College Xerox Center App

Introduction

The **College Xerox Center App** is designed to simplify the document printing process for students and administrators. It provides an easy-to-use platform where students can upload, customize, and pay for their prints, while admins can efficiently manage orders and process print requests.

Key Features

1. **Document Upload & Print Customization**
    - Students can upload documents (PDF, DOCX) and customize print options like **paper size, color, and copies** before submitting a print request.
2. **Secure Online Payment**
    - Integrated **payment gateway (Razorpay/Stripe)** for hassle-free online payments before processing print jobs.
3. **Order Tracking & Notifications**
    - Users receive **real-time updates** on order status (pending, printing, completed) and notifications when their prints are ready for pickup.
4. **Admin Dashboard & Advanced Features**
    - **Order Management**: Admin can **view, accept, and process print jobs** efficiently.
    - **Pricing Control**: Set and update print prices based on page type and color options.
    - **User Management**: Admin can **manage student accounts and order history**.

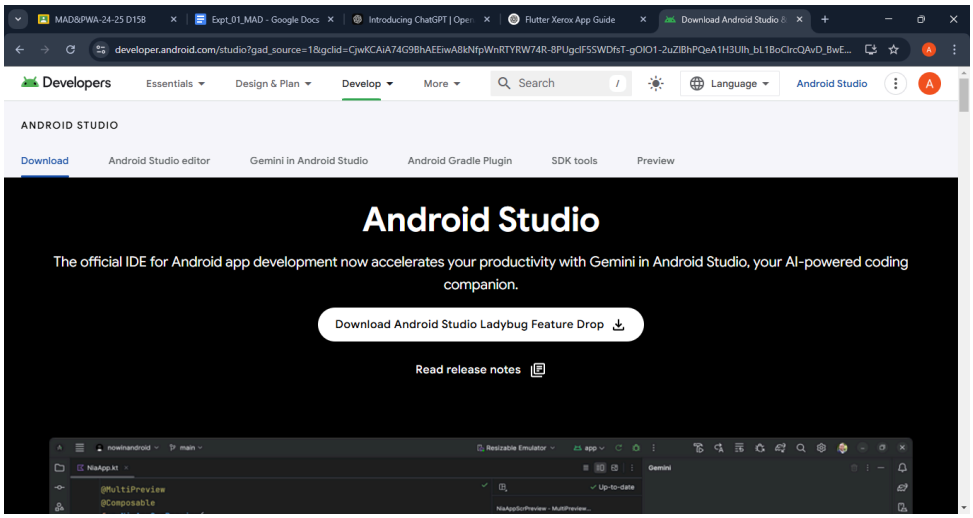
Conclusion

This app will modernize the traditional Xerox center by making printing more **efficient, fast, and user-friendly**. With seamless payment, order tracking, and a powerful admin dashboard, it enhances convenience for both students and admins. The app will **reduce waiting time, minimize errors, and improve overall service quality**, making it an essential solution for any college Xerox center.

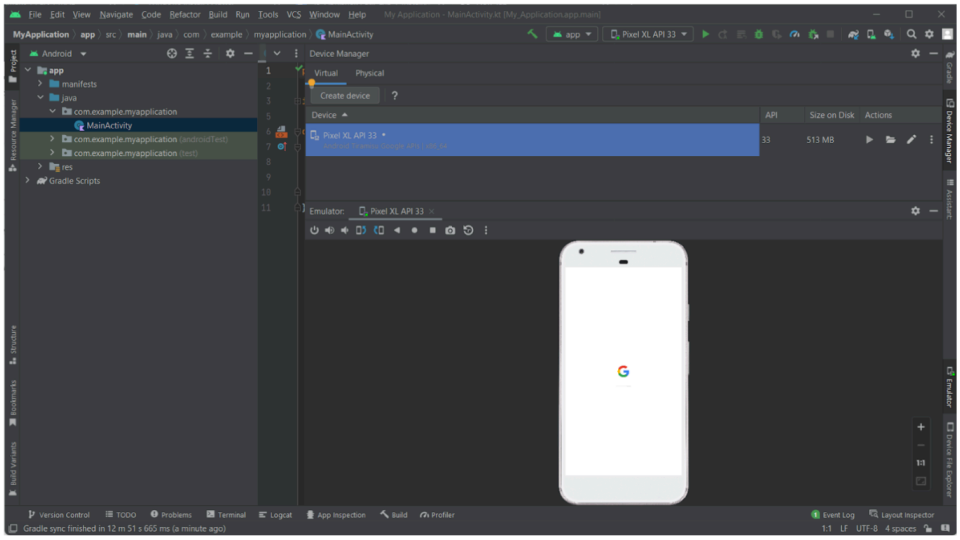
MAD & PWA
Experiment No: 01

Aim: To setup Android App Development Environment.

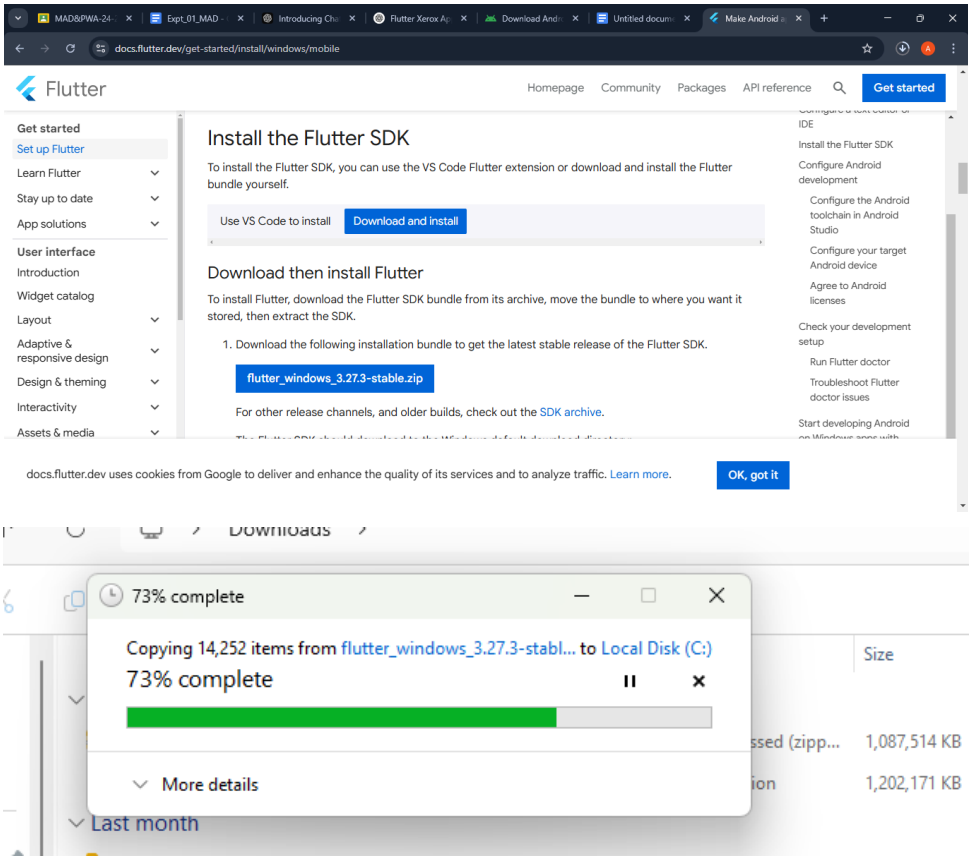
Step 1: Download Android Studio, run exe file and follow on screen steps to install it.



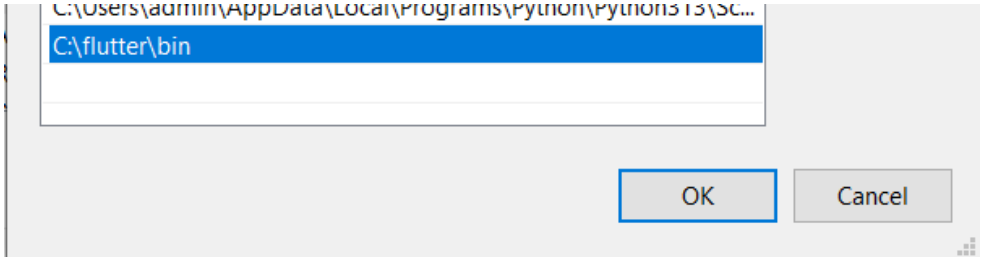
Step 2: set up an Android emulator in Setting>Virtual Device Manager.



Step 3: Download and install Flutter SDK by Extracting zip file(preferably in location C:\flutter).



Step 4: Set path of flutter sdk bin in System path to access flutter in command prompt.



Step 5: Run flutter and flutter doctor command to ensure everything is setup for development.

```
C:\Users\admin>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.27.4, on Microsoft Windows [Version 10.0.19045.5371], locale en-IN)
[√] Windows Version (Installed version of Windows is version 10 or higher)
[√] Android toolchain - develop for Android devices (Android SDK version 35.0.1)
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2024.2)
[√] VS Code (version 1.97.0)
[√] Connected device (3 available)
[√] Network resources

! Doctor found issues in 1 category.

C:\Users\admin>
```

Experiment 2

Aim: To design flutter UI using common widgets

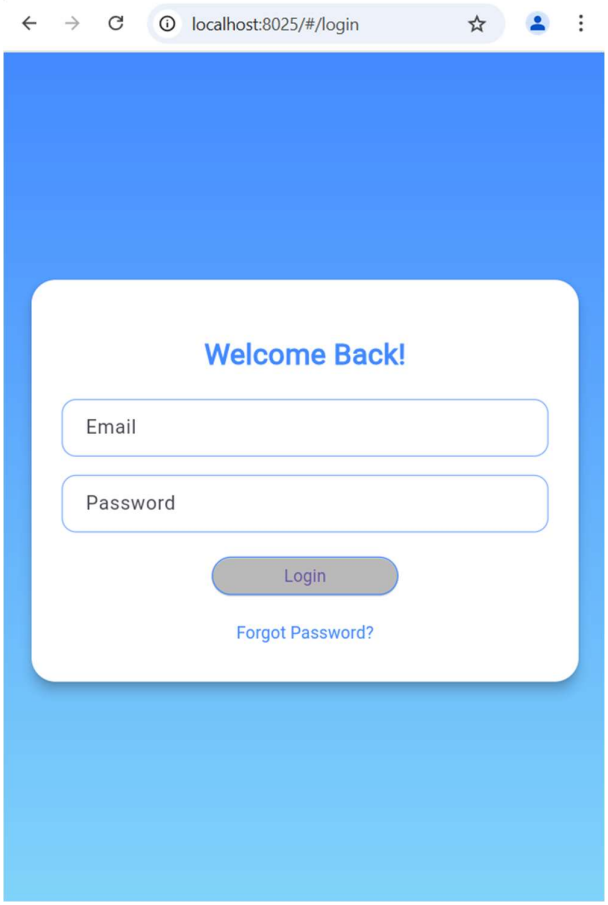
Code :

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

class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  bool isLoading = false;

  void loginUser(BuildContext context) async {
    setState(() => isLoading = true);
    try {
      await Provider.of<AuthProvider>(context, listen: false)
          .login(emailController.text, passwordController.text);
      Navigator.pushReplacementNamed(context, '/home');
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Login Failed: ${e.toString()}"),
          backgroundColor: Colors.redAccent,
        ),
      );
    }
    setState(() => isLoading = false);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        width: double.infinity,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [Colors.blueAccent, Colors.lightBlue.shade200],
          ),
        ),
```

```
child: Center(
  child: Padding(
    padding: EdgeInsets.all(20.0),
    child: Card(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(20),
      ),
      elevation: 8,
      color: Colors.white,
      child: Padding(
        padding: EdgeInsets.all(25.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            SizedBox(height: 20),
            Text(
              "Welcome Back!",
              style: TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                color: Colors.blueAccent,
              ),
            ),
            SizedBox(height: 20),
            _buildTextField(
                controller: emailController,
                label: "Email"),
            SizedBox(height: 15),
            _buildTextField(
                controller: passwordController,
                label: "Password",
                isPassword: true),
            SizedBox(height: 20),
            isLoading
                ? CircularProgressIndicator()
                : ElevatedButton(
                    onPressed: () => loginUser(context),
                    child: Text("Login"),
                    style: ElevatedButton.styleFrom(
                      padding: EdgeInsets.symmetric(vertical: 12, horizontal: 60),
                      backgroundColor: Colors.transparent, // Make background transparent
                      side: BorderSide(color: Colors.blueAccent), // Optional: Adds a border to the button
                    ),
                  ),
            SizedBox(height: 15),
            TextButton(
              onPressed: () {}, // Add forgot password logic
              child: Text(
```

```
                  "Forgot Password?",
                  style: TextStyle(color: Colors.blueAccent),
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  ),
 );
}

Widget _buildTextField(
  {required TextEditingController controller, required String label, bool isPassword = false}) {
 return TextField(
  controller: controller,
  obscureText: isPassword,
  decoration: InputDecoration(
   labelText: label,
   filled: true,
   fillColor: Colors.white,
   contentPadding: EdgeInsets.symmetric(horizontal: 20, vertical: 14),
   border: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: BorderSide(color: Colors.blueAccent),
   ),
   enabledBorder: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: BorderSide(color: Colors.blueAccent.shade100),
   ),
   focusedBorder: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: BorderSide(color: Colors.blueAccent, width: 2),
   ),
  ),
 );
}
}
```

Output:


localhost:8025/#/login
Welcome Back!
Email
Password
Login
Forgot Password?

Experiment 3

Aim: To include icons, images and fonts in App.

Code :

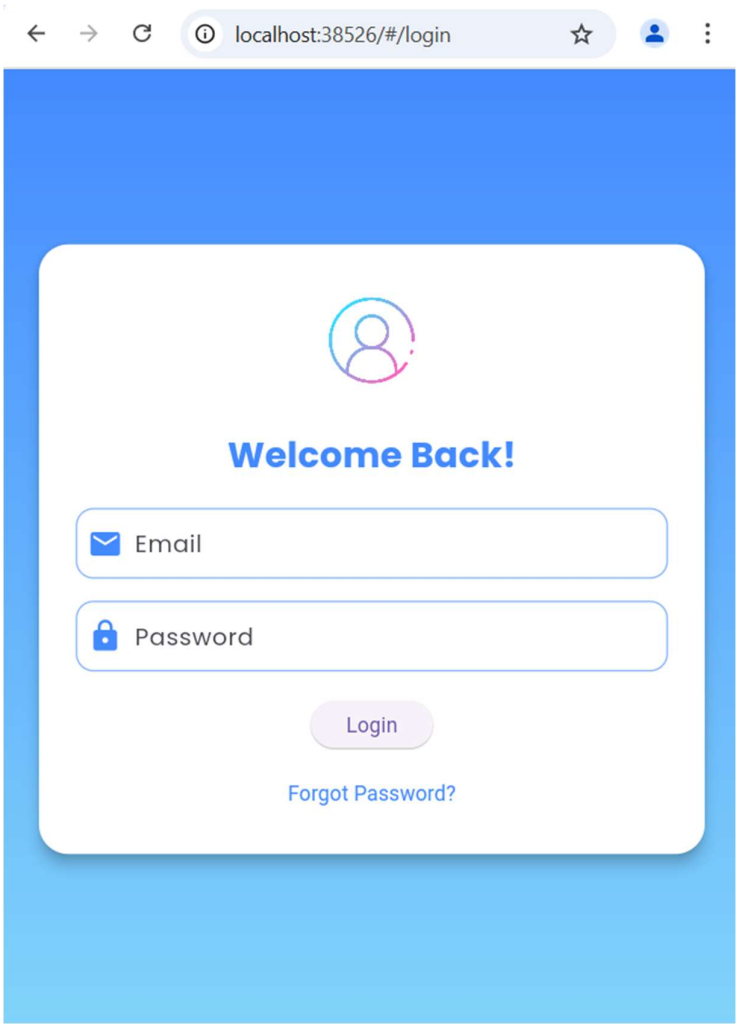
Login Page:

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

class LoginScreen extends StatefulWidget {
  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  bool isLoading = false;

  void loginUser(BuildContext context) async {
    setState(() => isLoading = true);
    try {
      await Provider.of<AuthProvider>(context, listen: false)
          .login(emailController.text, passwordController.text);
      Navigator.pushReplacementNamed(context, '/home');
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Login Failed: ${e.toString()}"),
          backgroundColor: Colors.redAccent,
        ),
      );
    }
    setState(() => isLoading = false);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        width: double.infinity,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
```

```
    end: Alignment.bottomCenter,
    colors: [Colors.blueAccent, Colors.lightBlue.shade200],
   ),
  ),
  child: Center(
   child: Padding(
    padding: EdgeInsets.all(20.0),
    child: Card(
     shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20),
     ),
     elevation: 8,
     color: Colors.white,
     child: Padding(
      padding: EdgeInsets.all(25.0),
      child: Column(
       mainAxisSize: MainAxisSize.min,
       children: [
        // Use a web image (URL)
        Image.network('https://cdn-icons-png.flaticon.com/512/5087/5087579.png', height: 80),
        SizedBox(height: 20),
        Text(
         "Welcome Back!",
         style: GoogleFonts.poppins(
          fontSize: 24,
          fontWeight: FontWeight.bold,
          color: Colors.blueAccent,
         ),
        ),
        SizedBox(height: 20),
        _buildTextField(
          controller: emailController,
          label: "Email",
          icon: Icons.email),
        SizedBox(height: 15),
        _buildTextField(
          controller: passwordController,
          label: "Password",
          isPassword: true,
          icon: Icons.lock),
        SizedBox(height: 20),
        isLoading
          ? CircularProgressIndicator()
          : CustomButton(
            text: "Login",
            onPressed: () => loginUser(context),
           ),
        SizedBox(height: 15),
```

```
                TextButton(
                  onPressed: () {}, // Add forgot password logic
                  child: Text(
                    "Forgot Password?",
                    style: TextStyle(color: Colors.blueAccent),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    ),
  );
}

Widget _buildTextField(
    {required TextEditingController controller,
    required String label,
    bool isPassword = false,
    IconData? icon}) {
  return TextField(
    controller: controller,
    obscureText: isPassword,
    decoration: InputDecoration(
      prefixIcon: icon != null ? Icon(icon, color: Colors.blueAccent) : null,
      labelText: label,
      labelStyle: GoogleFonts.poppins(fontSize: 16),
      filled: true,
      fillColor: Colors.white,
      contentPadding: EdgeInsets.symmetric(horizontal: 20, vertical: 14),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.blueAccent),
      ),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.blueAccent.shade100),
      ),
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.blueAccent, width: 2),
      ),
    ),
  );
 }
}
```

Output:


localhost:38526/#/login
Welcome Back!
Email
Password
Login
Forgot Password?

MPL
Experiment 4

Aim: To create an interactive Form using form widget.
Program:

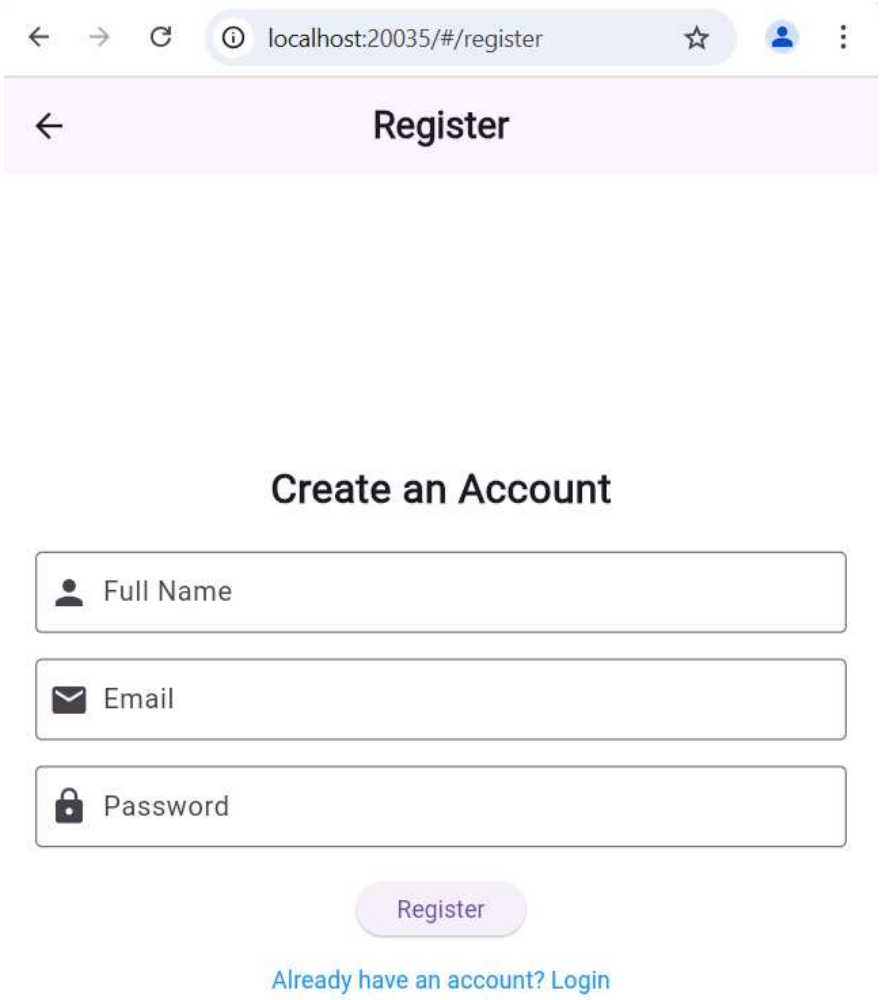
```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../providers/auth_provider.dart';
import '../widgets/custom_button.dart';

class RegisterScreen extends StatefulWidget {
  @override
  _RegisterScreenState createState() => _RegisterScreenState();
}

class _RegisterScreenState extends State<RegisterScreen> {
  final TextEditingController nameController = TextEditingController();
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  final _formKey = GlobalKey<FormState>();
  bool isLoading = false;

  void registerUser(BuildContext context) async {
    if (!_formKey.currentState!.validate()) return;

    setState(() => isLoading = true);

    try {
      await Provider.of<AuthProvider>(context, listen: false).register(
        nameController.text.trim(),
        emailController.text.trim(),
        passwordController.text.trim(),
      );
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Registration Successful! Please login.")),
      );
      Navigator.pushReplacementNamed(context, '/login');
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Error: ${e.toString()}")),
      );
    }

    setState(() => isLoading = false);
```

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.white,
    appBar: AppBar(
      title: Text('Register', style: TextStyle(fontWeight: FontWeight.bold)),
      centerTitle: true,
      elevation: 0,
    ),
    body: Padding(
      padding: EdgeInsets.all(20),
      child: Form(
        key: _formKey,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              "Create an Account",
              style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 20),

            // Name Field
            TextFormField(
              controller: nameController,
              decoration: InputDecoration(
                labelText: "Full Name",
                prefixIcon: Icon(Icons.person),
                border: OutlineInputBorder(),
              ),
              validator: (value) =>
                  value!.isEmpty ? "Enter your name" : null,
            ),
            SizedBox(height: 15),

            // Email Field
            TextFormField(
              controller: emailController,
              decoration: InputDecoration(
                labelText: "Email",
                prefixIcon: Icon(Icons.email),
                border: OutlineInputBorder(),
```

```
    ),
    validator: (value) {
      if (value!.isEmpty) return "Enter your email";
      if (!RegExp(r"^[a-zA-Z0-9.]+@[a-zA-Z0-9]+\.[a-zA-Z]+")
          .hasMatch(value)) {
        return "Enter a valid email";
      }
      return null;
    },
  ),
  SizedBox(height: 15),

  // Password Field
  TextFormField(
    controller: passwordController,
    obscureText: true,
    decoration: InputDecoration(
      labelText: "Password",
      prefixIcon: Icon(Icons.lock),
      border: OutlineInputBorder(),
    ),
    validator: (value) =>
        value!.length < 6 ? "Password must be 6+ chars" : null,
  ),
  SizedBox(height: 20),

  // Register Button
  isLoading
      ? CircularProgressIndicator()
      : CustomButton(
          text: "Register",
          onPressed: () => registerUser(context),
        ),
  SizedBox(height: 10),

  // Already have an account? Login
  TextButton(
    onPressed: () {
      Navigator.pushReplacementNamed(context, '/login');
    },
    child: Text("Already have an account? Login",
        style: TextStyle(color: Colors.blue)),
  ),
],
```

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

Output:


localhost:20035/#/register
Register
Create an Account
Full Name
Email
Password
Register
Already have an account? Login

MPL

Experiment 5

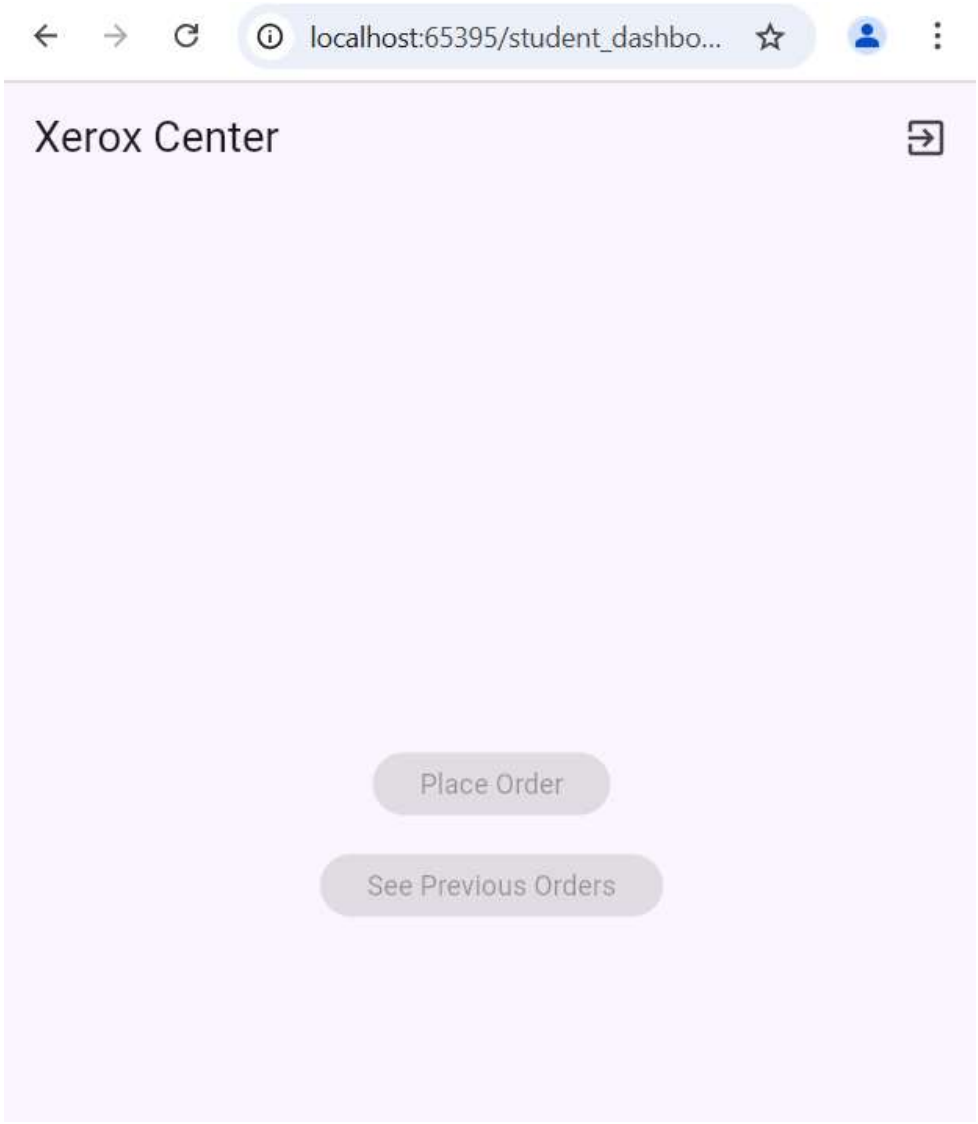
Aim: To apply navigation, routing and gestures in Flutter App

Program:

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

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Xerox Center"),
        actions: [
          GestureDetector(
            onTap: () {
              Navigator.pop(context);
            },
            child: Padding(
              padding: EdgeInsets.symmetric(horizontal: 16),
              child: Icon(Icons.exit_to_app),
            ),
          ),
        ],
      ),
      body: GestureDetector(
        onDoubleTap: () {
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text("Double Tap Detected")),
          );
        },
        onLongPress: () {
          ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text("Long Press Detected")),
          );
        },
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              GestureDetector(
                onTap: () {
                  Navigator.push(
```

```
                context,
                MaterialPageRoute(builder: (context) => OrderScreen()),
              );
            },
            child: ElevatedButton(
              onPressed: null, // Button is controlled by GestureDetector
              child: Text("Place Order"),
            ),
          ),
          SizedBox(height: 20),
          GestureDetector(
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => PreviousOrdersScreen()),
              );
            },
            child: ElevatedButton(
              onPressed: null, // Controlled by GestureDetector
              child: Text("See Previous Orders"),
            ),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

Output:


localhost:65395/student_dashbo...
Xerox Center
Place Order
See Previous Orders

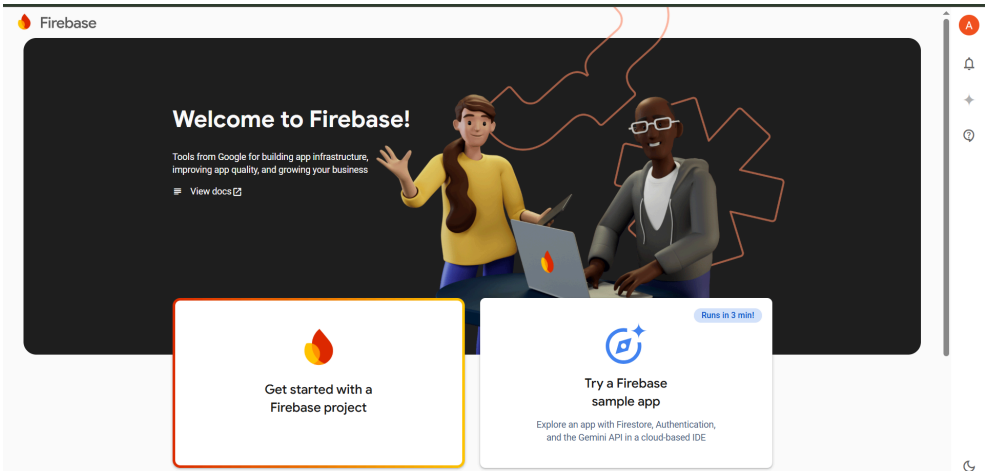
MPL

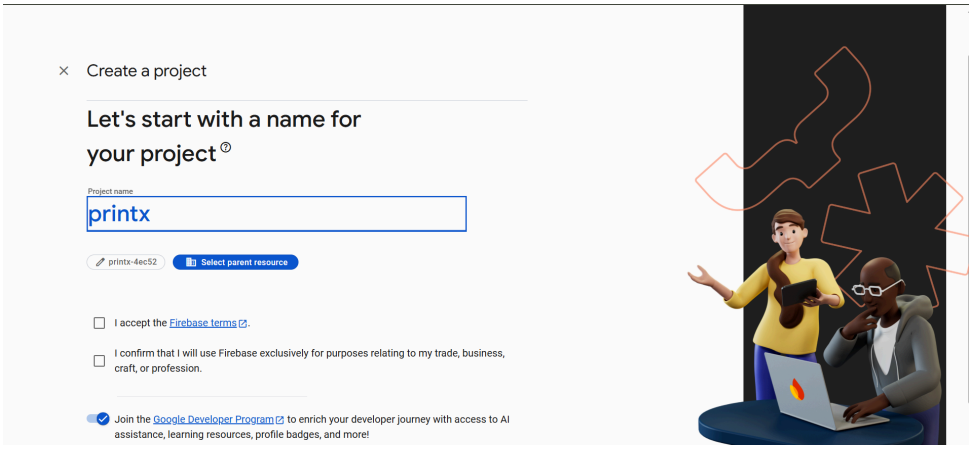
Experiment 6

Aim: To Set Up Firebase with Flutter for iOS and Android Apps

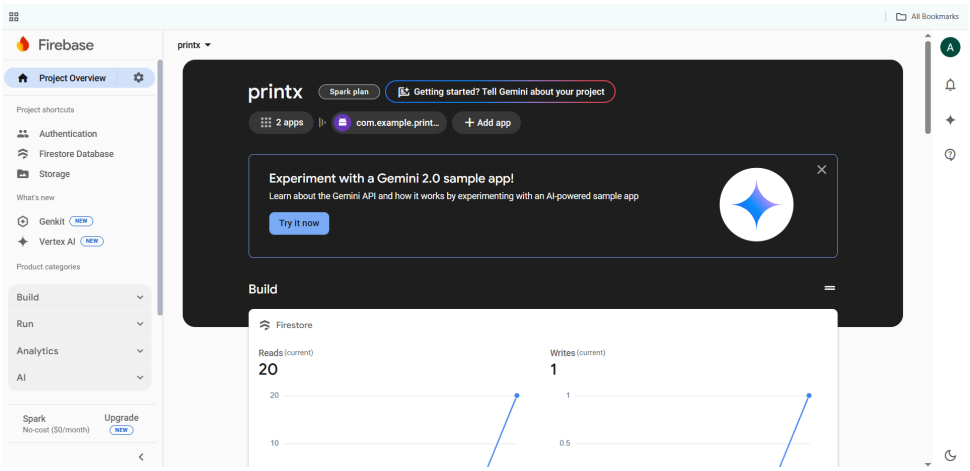
1)Create firebase account.

2)create project in firebase.





Download and add the `google-services.json` (Android) and `GoogleService-Info.plist` (iOS) files to the project.



Initialize firebase in flutter app (main.dart)

Experiment No: 07

Aim: To Converting your MERN website into a Progressive Web App (PWA) that users can "Add to Homescreen" just like a mobile app.

Steps:

Step 1: Create and Add a `manifest.json` file

In your React app (inside the `public/` folder), create a file called `manifest.json` with this content:

```
{
  "name": "Your App Name",
  "short_name": "AppShort",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#000000",
  "description": "A sample PWA for experiment.",
  "icons": [
    {
      "src": "icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

*****Make sure to place `icon-192x192.png` and `icon-512x512.png` inside `public/icons/`.

Step 2: Add Manifest Link in `public/index.html`

Add this inside the <head> tag of your public/index.html:

```
<link rel="manifest" href="./manifest.json" />
<meta name="theme-color" content="#000000" />
```

Step 3: Add the Service Worker (Optional if not using CRA)

If you're using **Create React App (CRA)**, it's already PWA-ready with serviceWorkerRegistration.js.

If **not** using CRA, create a serviceworker.js in public/:

```
// public/serviceworker.js
const CACHE_NAME = 'pwa-cache';
const urlsToCache = ['/', '/index.html'];

self.addEventListener('install', event => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => cache.addAll(urlsToCache))
  );
});

self.addEventListener('fetch', event => {
  event.respondWith(
    caches.match(event.request)
      .then(resp => resp || fetch(event.request))
  );
});
```

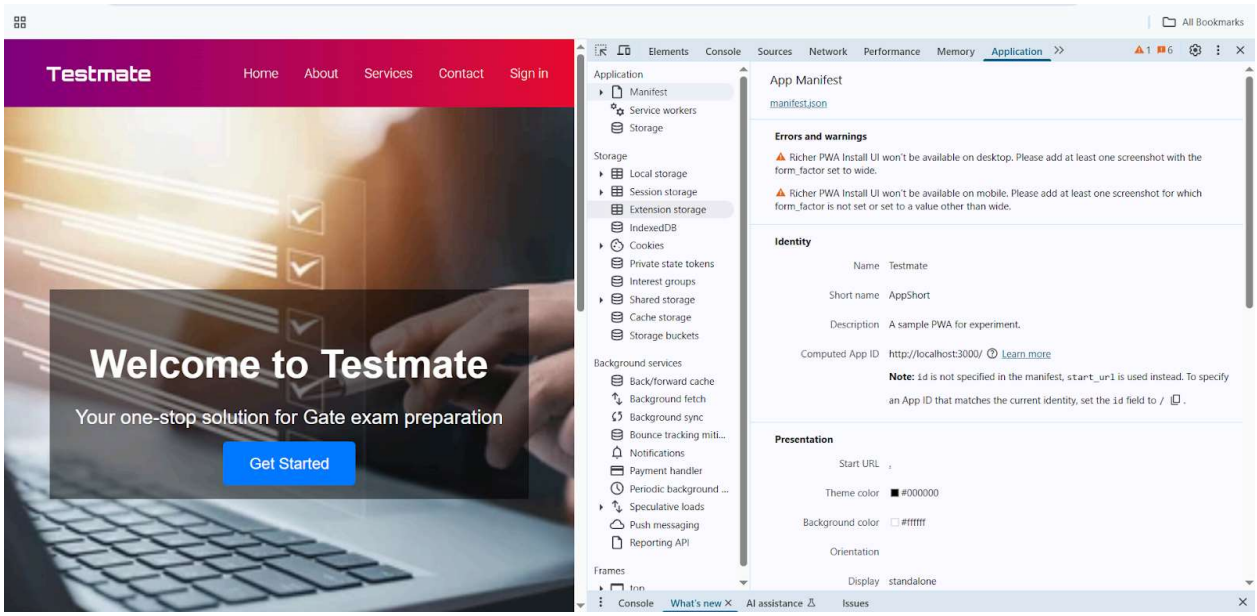
Step 4: Register Service Worker in React

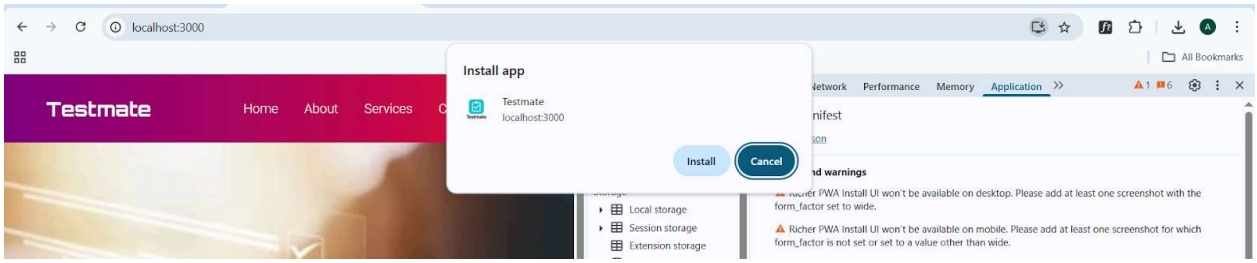
Inside your main JS file (e.g. src/index.js):

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/serviceworker.js')
      .then(reg => console.log("Service Worker registered", reg))
      .catch(err => console.log("Service Worker registration failed",
err));
  });
}
```

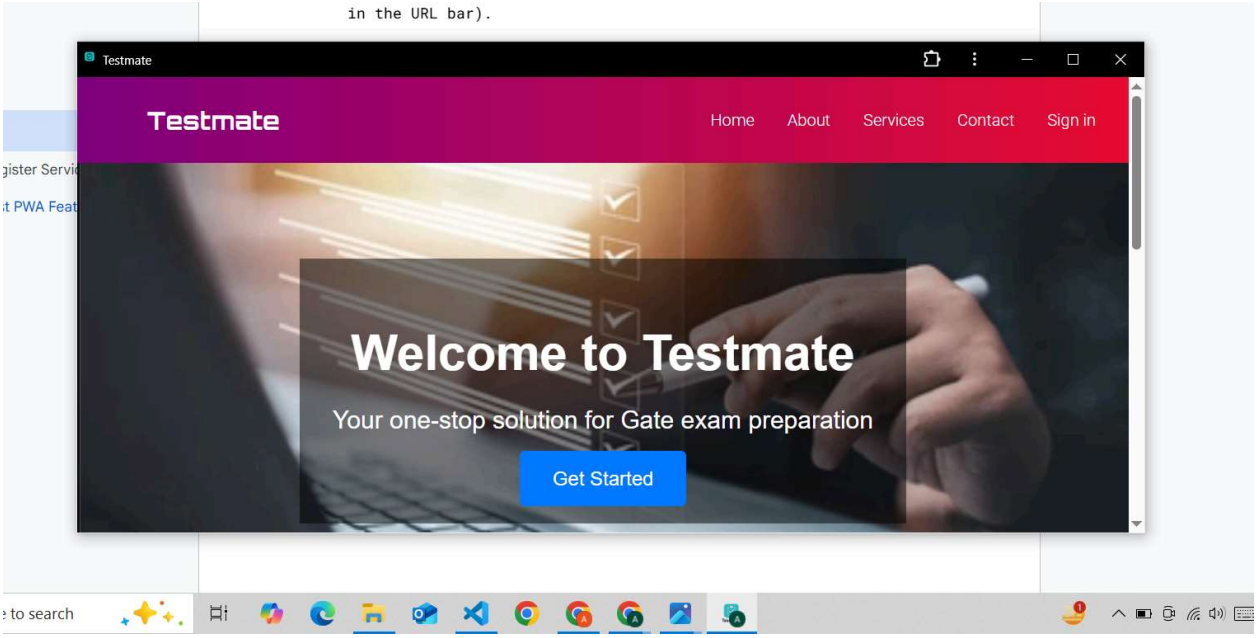
Step 5: Test PWA Feature

1. Start your React app locally (or deploy it on Netlify/Vercel).
2. Open in Chrome browser.
3. Open DevTools → Application tab → Manifest.
4. Check if:
    - Manifest is loaded.
    - Icons are valid.
    - Service Worker is registered.
5. You should see an option to “Install App” (from the 3-dot menu or in the URL bar).





in the URL bar).



Experiment No: 08

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

- A Service Worker (SW) is a script that runs in the background, separate from your web page.
- It enables features like offline caching, push notifications, and background sync.
- It doesn't have access to the DOM directly.
- Runs only on HTTPS (except on `localhost` during development).

Step 1: Code in `main.js` (Service Worker Registration)

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

- Place this script in your `main.js` and make sure it is **linked in your HTML file** using `<script src="main.js"></script>`.

Step 2: Code in `sw.js` (Install + Activate + Fetch Handling)

```
// Caching important files
const filesToCache = [
  './components/Login.js',
'./components/Home.js',
];
```

```
const CACHE_NAME = 'offline';

// Install event
self.addEventListener("install", function (event) {
  event.waitUntil(
    caches.open(CACHE_NAME).then(function (cache) {
      return cache.addAll(filesToCache);
    })
  );
});

// Activate event (clean up old caches if needed)
self.addEventListener("activate", function (event) {
  console.log("Service worker activated.");
});

// Fetch event
self.addEventListener("fetch", function (event) {
  event.respondWith(
    fetch(event.request).then(function (response) {
      if (response.status !== 404) {
        return response;
      } else {
        return caches.match("offline.html");
      }
    }).catch(function () {
      return caches.match(event.request).then(function (response) {
        return response || caches.match("offline.html");
      });
    })
  );

  event.waitUntil(
    caches.open(CACHE_NAME).then(function (cache) {
      return fetch(event.request).then(function (response) {
        return cache.put(event.request, response.clone());
      });
    })
  );
});
```

💻 Step 3: Run Your Project Locally

You can use a **local server** (HTTPS not required on localhost):

🛠️ If using VS Code:

Install Live Server Extension and run your project with **Right click → "Open with Live Server"**

Or use:

```
npx serve
```

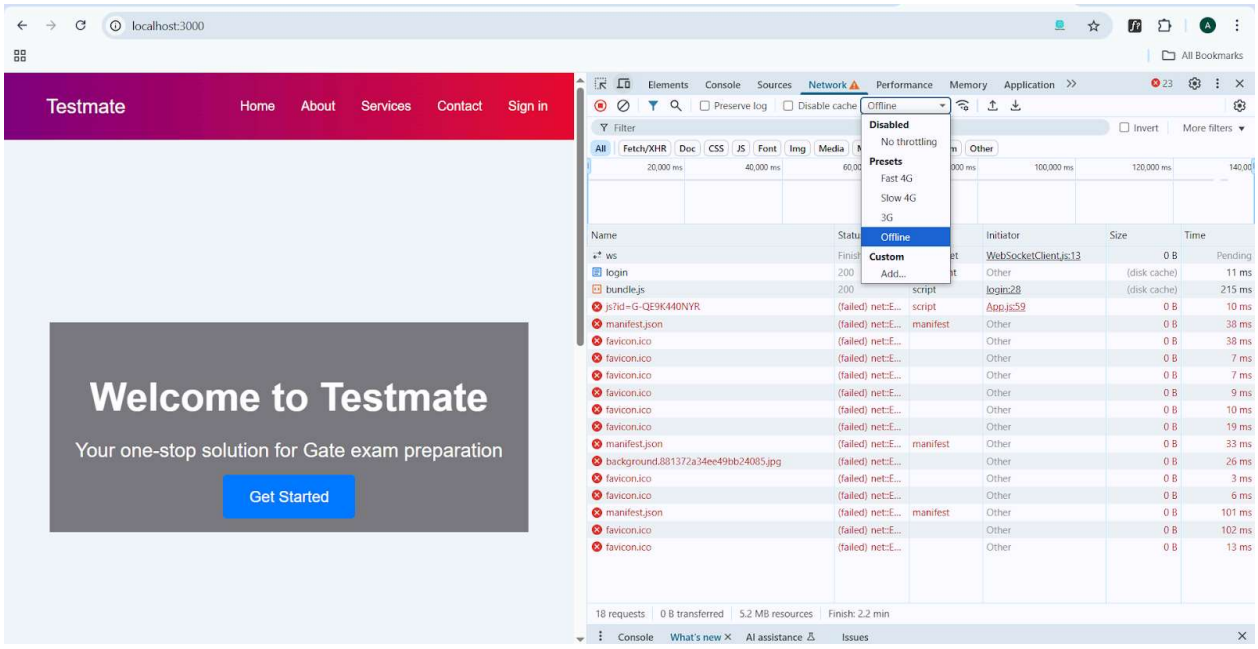
or

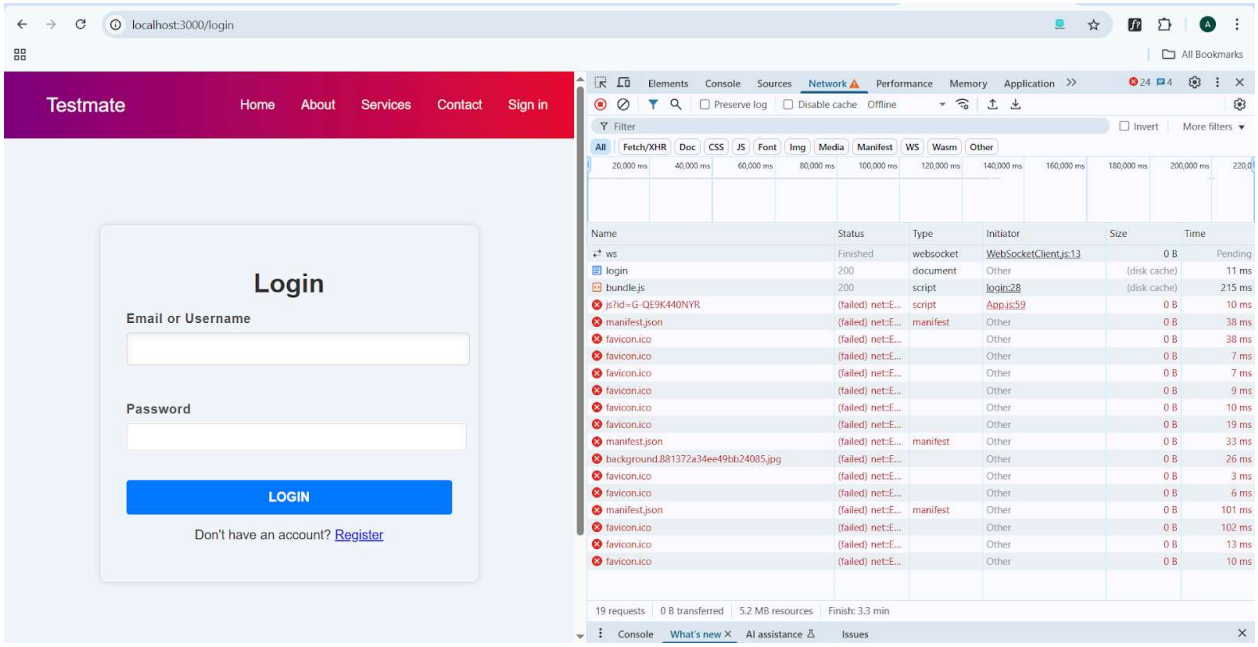
```
python3 -m http.server 8080
```

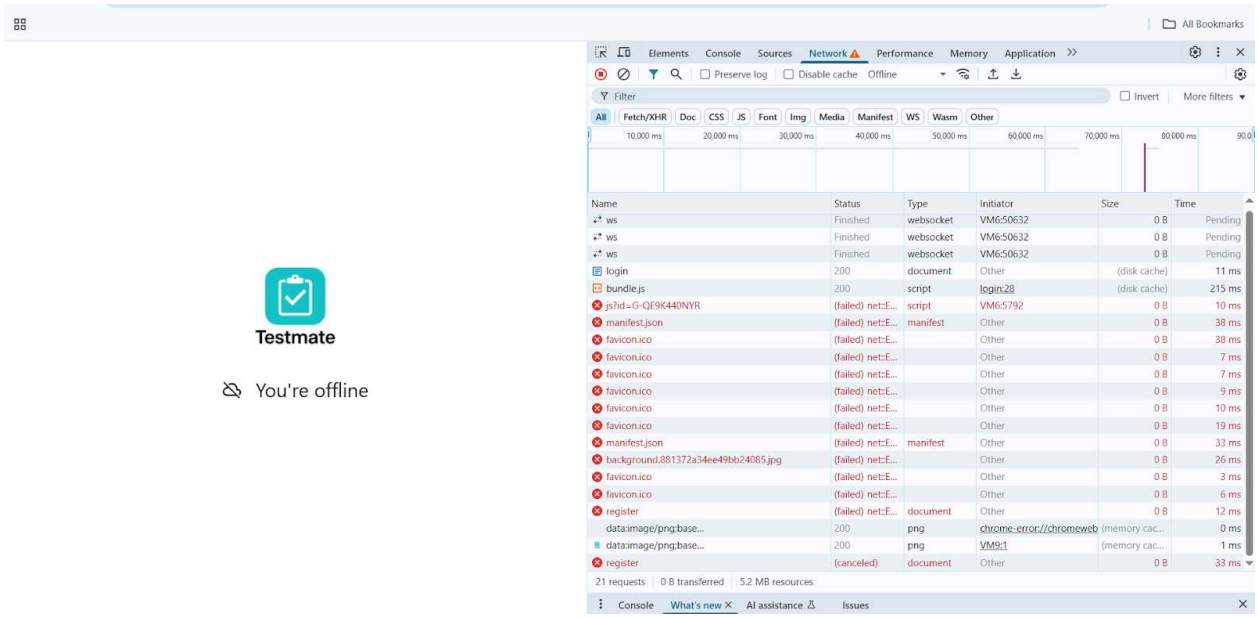
Then go to `http://localhost:8080`

Step 4: Test It

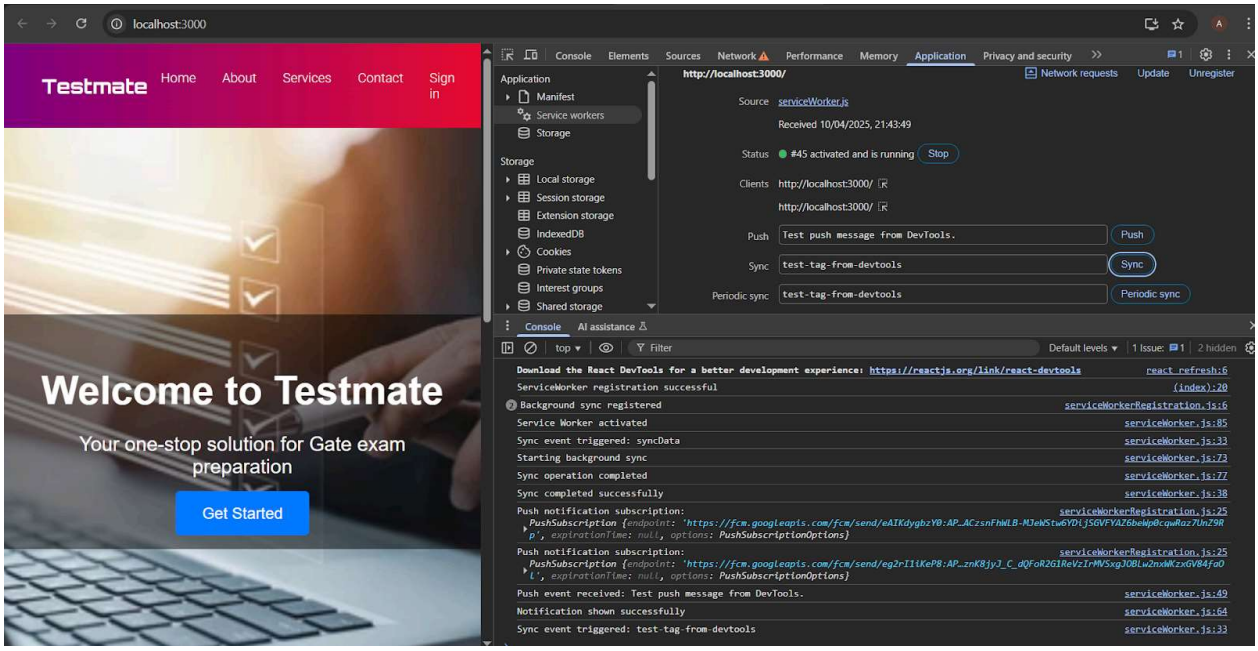
1. Open DevTools → Application tab → Check if service worker is registered.
2. Go Offline (DevTools → Network → Offline).
3. Visit `/home` or `/login` → Page should load from cache.
4. Try a non-cached URL → Should show `offline.html`.


localhost:3000
Testmate
Home
About
Services
Contact
Sign in
Welcome to Testmate
Your one-stop solution for Gate exam preparation
Get Started
Elements
Console
Sources
Network
Performance
Memory
Application
Offline
Disabled
No throttling
Presets
Fast 4G
Slow 4G
3G
Offline
Custom
Add...
18 requests
0 B transferred
5.2 MB resources
Finish: 2.2 min

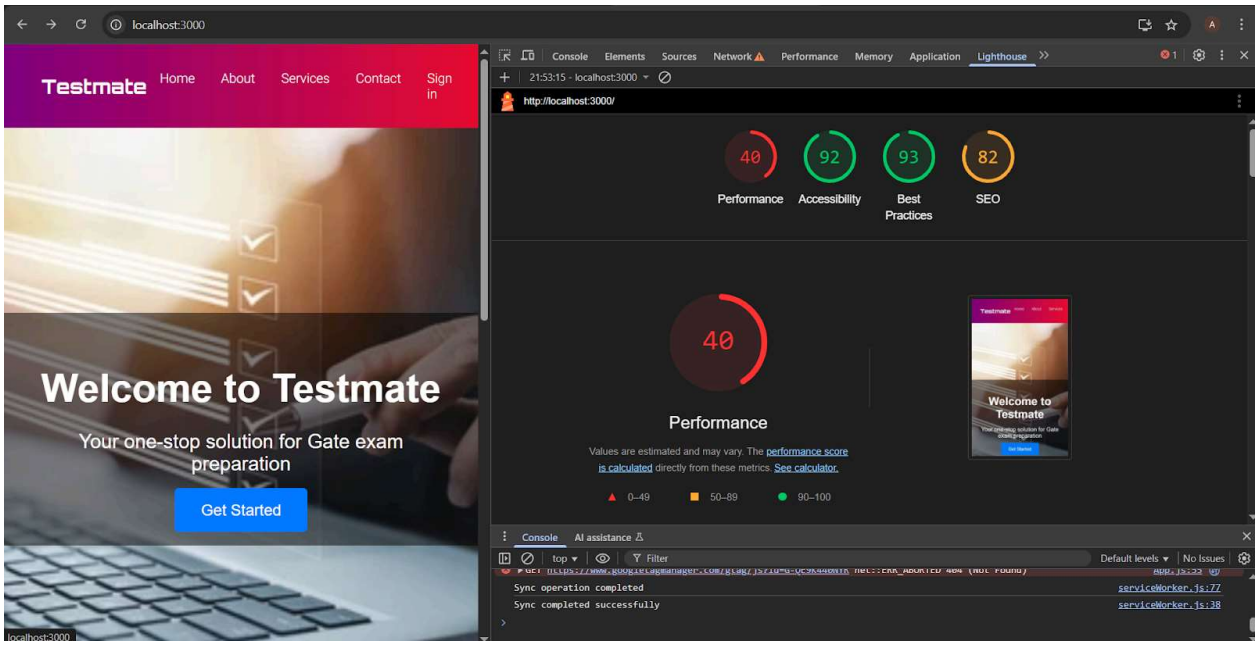

localhost:3000/login
Testmate
Home
About
Services
Contact
Sign in
Login
Email or Username
Password
LOGIN
Don't have an account? Register
19 requests
0 B transferred
5.2 MB resources
Finish: 3.3 min


All Bookmarks
Testmate
You're offline
Elements
Console
Sources
Network
Performance
Memory
Application
Preserve log
Disable cache
Offline
Filter
Invert
More filters
All
Fetch/XHR
Doc
CSS
JS
Font
Img
Media
Manifest
WS
Wasm
Other
10,000 ms
20,000 ms
30,000 ms
40,000 ms
50,000 ms
60,000 ms
70,000 ms
80,000 ms
Name
Status
Type
Initiator
Size
Time
ws Finished websocket VM6:50632 0 B Pending
ws Finished websocket VM6:50632 0 B Pending
ws Finished websocket VM6:50632 0 B Pending
login 200 document Other (disk cache) 11 ms
bundle.js 200 script login:28 (disk cache) 215 ms
js?id=G-QE9K440NYR (failed) net::E... script VM6:5792 0 B 10 ms
manifest.json (failed) net::E... manifest Other 0 B 38 ms
favicon.ico (failed) net::E... Other 0 B 38 ms
favicon.ico (failed) net::E... Other 0 B 7 ms
favicon.ico (failed) net::E... Other 0 B 7 ms
favicon.ico (failed) net::E... Other 0 B 9 ms
favicon.ico (failed) net::E... Other 0 B 10 ms
favicon.ico (failed) net::E... Other 0 B 19 ms
manifest.json (failed) net::E... manifest Other 0 B 33 ms
background.881372a34ee49bb24085.jpg (failed) net::E... Other 0 B 26 ms
favicon.ico (failed) net::E... Other 0 B 3 ms
favicon.ico (failed) net::E... Other 0 B 6 ms
register (failed) net::E... document Other 0 B 12 ms
data:image/png;base... 200 png chrome-error://chromeweb (memory cac... 0 ms
data:image/png;base... 200 png VM9:1 (memory cac... 1 ms
register (canceled) document Other 0 B 33 ms
21 requests
0 B transferred
5.2 MB resources
Console
What's new
AI assistance
Issues

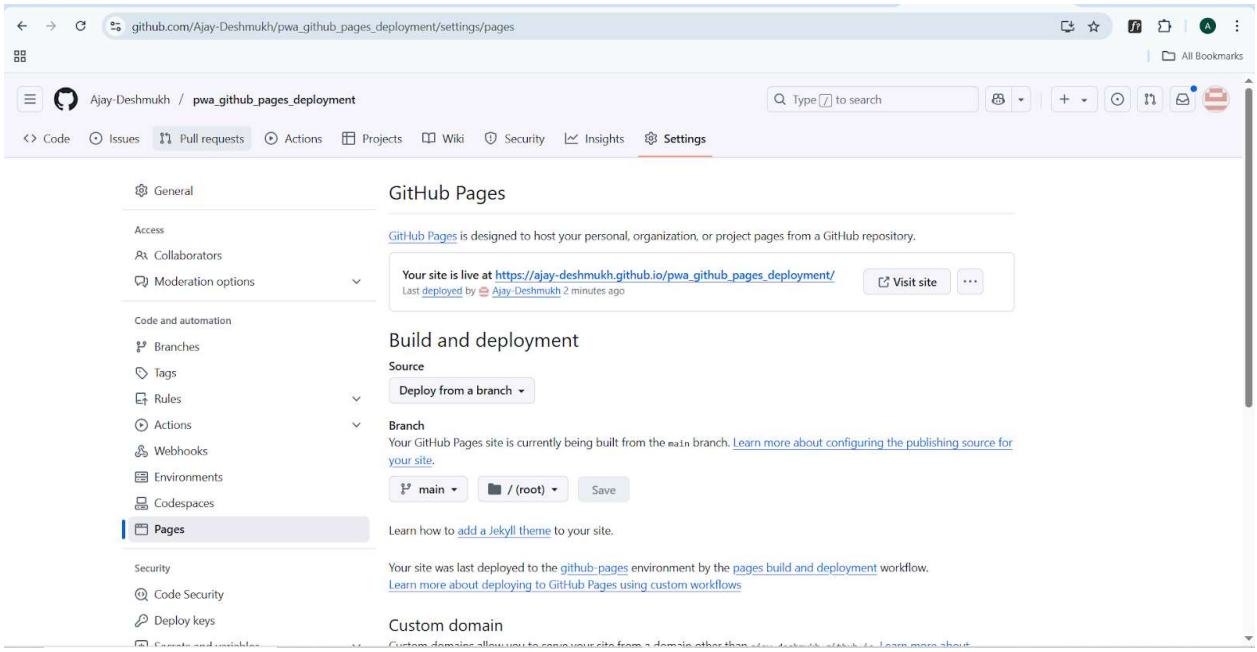
Experiment No: 09

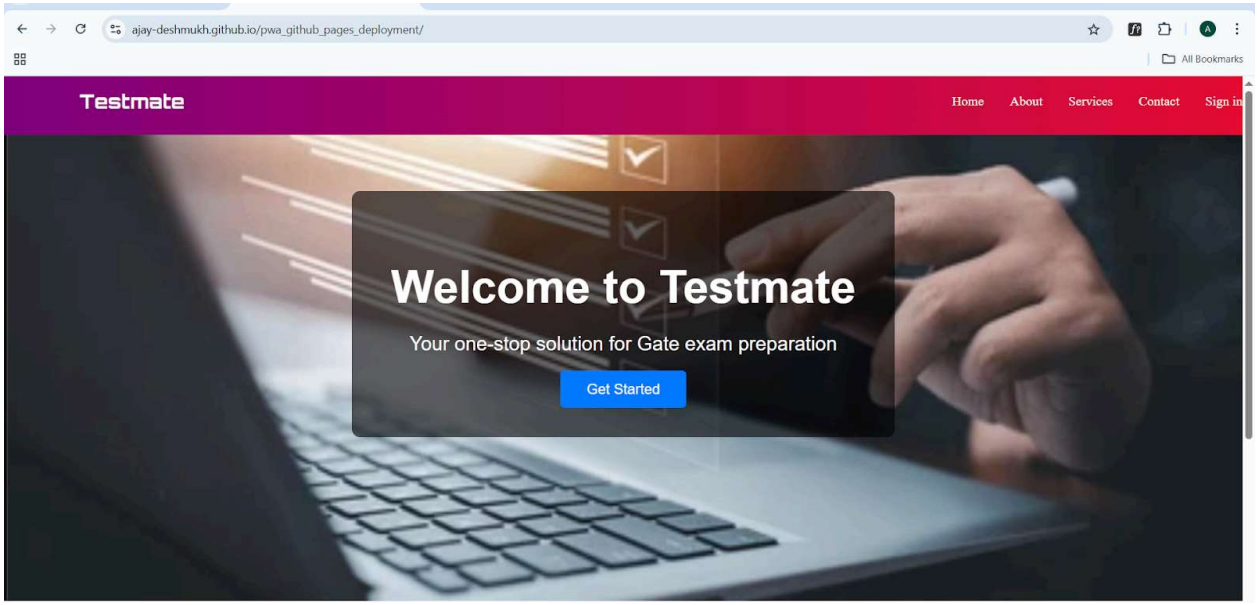


Experiment No: 10



Experiment No: 11





Name:- Ajay Deshmukh
Div :- D15B
Roll no:- 11

MPL
Assignment No:01

DOS :- 28/2/2025

1) Explain the key features and advantage of using Flutter for mobile app development

Ans:- Flutter is an open-source UI framework developed by google for building natively compiled applications for mobile, web, and desktop from a Single Codebase.

*key features:-

1) Single Codebase:- Write one codebase and deploy it on Android, ios, web and Desktop.
2) Hot Reload:- Instantly see changes made in the code without restarting the app.
3) Widget-Based UI:- Everything in flutter is a Widget, making UI creation flexible and reusable.
4) Fast Performance:- Uses Dart's ahead-of-Time Compilation for high performance.
5) Customizable UI:- Rich set of material and cupertino Widgets for Seamless Android and ios design.
6) Strong Community & Google Support:- Regular updates and Strong community backing.

*Advantages:-

1) Fast Development:- Hot reload and a rich set of pre-built widgets speed up development
2) Consists UI Across Platform:- No need for Platform Specific UI coding.
3) Lower Development Cost:- A single team can handle multiple platforms.
4) Better Performance:- No reliance on Javascript bridges leading to smooth UI interactions.

Teacher's Sign:

2.1.b) How Flutter Differs from Traditional Approach & its Properties

→ Ans:- * Tradition Approach (Native Development)
→ Android :- Uses Java/Kotlin and XML for UI
→ Ios :- Uses Swift/objective C with storyboards or Sw
→ Separate Codebases :- Developers write and maintain Separa code for different platforms

* Flutter's Approach :-
→ Uses Dart for both frontend and backend logic
→ Employs a widget-based UI rather than XML-based design
→ Single codebase for both Android and Ios.
→ Own Rendering Engine (skia), eliminating platf dependency.

* Reasons for Flutter's Popularity
1) Faster Development with Hot Reload
2) Cross-Platform Development without Performance Trade-offs
3) Rich UI Capabilities with Pre-built widgets.
4) Backed by Google & strong Open-source Community.
5) Growing Adoption in Enterprises & startups.

Teacher's Sign.:

Concept of Widget Tree & Widget Composition in Flutter.

Ans:- Widget Tree in Flutter

-> In Flutter, everything is a widget (buttons, text, images, Containers etc)

-> These widgets are arranged in a tree-like structure called the widget Tree.

-> The tree structure determines the layout and behavior of UI elements.

* Widget Composition:-

-> Complex UIs are built by nesting widgets within each other.

-> StatelessWidgets are immutable, while StatefulWidgets maintain dynamic state.

Commonly Used Widgets & Their Roles in Widget Tree

*Basic Widgets:-

1) Container: Used for layout, padding, and Styling.
2) Row and Column: Arrange widgets horizontally and Vertically.
3) Text: Displays text Content.
4) Image Displays images from assets or network.

* Structural Widgets

1) Scaffold:- Provides a default layout structure with an app bar, body, and floating button.

Teacher's Sign.:

2) AppBar :- Creates a material-design-style app bar.
3) ListView :- Creates a scrollable list of widgets.

* Interactive Widgets :-
1) GestureDetector :- Detects user gestures like taps and swipes.
2) ElevatedButton :- A button with elevation effects.
3) TextField :- An input field for user text input.

Q.3.a) Importance of State Management in Flutter Applications.

→ State management controls the behavior and data flow in an app. it ensures
→ UI updates dynamically based on user actions.
→ Efficient data handling and component reusability
→ Maintainability of Complex applications.

Q.3.b) Comparison of State Management Approaches.

→ * Scenarios for Each Approach :-
→ SetState :- Small applications where minimal state changes occur.
→ Provider :- Medium-scale apps needing optimized UI Updates
→ Riverpod :- Large-scale applications requiring dependency injection and modular state management.

Approach	Best for	Pros	Cons
setState	Small apps	Simple to use	UI overhaills frequently
Provider	Medium apps	Optimized for performance	Require understanding of
Riverpod	Large apps	Scalable & Testable	Learning curve is higher

Teacher's Sign.:

Firebase Integration & Benefits

* Steps to Integrate Firebase in Flutter

1) Create a firebase project at Firebase Console.
2) Add an Android/ios app to the firebase Project
3) Download and add the google-service.json (for Android) or GoogleService-Info.plist (for ios) to the Flutter Project

4) Install Firebase dependencies (firebase_core, cloud_firestore)
5) Initialize Firebase in main.dart:

```
import 'package:firebase_core/firebase_core.dart';

Void main() async {
    widgetsFlutterBinding.ensureInitialized();
    await Firebase.initializeApp();
    runApp(MyApp());
}
```

* Benefits of firebase as a backend Solution

-> No need to manage backend Servers
-> Real-time database Synchronization
-> Integrated authentication (Google, Facebook)
-> Cloud Functions for business logic.

Q.4 b) Common Firebase Services Used in Flutter and Data Synchronization.

→ Common Firebase Services :-

1) Firebase Authentication :- User login / signup using email, Google, Facebook, etc.
2) Cloud Firebase :- NoSQL real-time database for storing app data.
3) Firebase Storage :- For storing images, videos and files.
4) Firebase Cloud Messaging (FCM) - Push notifications.
5) Firebase Analytics - Tracks user behaviour

* Data Synchronization in Firebase

→ Firestore uses real-time listeners to automatically update data when changes occur.

Fetching and listening to real time Firestore updates

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

Experiment No.	
Title.	
Roll No	
Name	
Class	
Subject	
Grade:	

Teacher's Sign.:

05/05

Name :- Ajay Deshmukh
Roll no : 11
Div :- D15B

MPL Assignment 02

Define progressive webapp (PWA) and explain its significance in modern web development. Discuss the key characteristics of PWAs from traditional mobile apps.

⇒ A progressive webapp (PWA) is a type of web application that works like a mobile app but runs in a browser.

Significance of PWA in modern web development:

1) Cross-platform Compatibility.
2) Offline Support.
3) Fast performance
4) No app store required
5) Lower Development Cost

Difference in PWA & traditional apps:

Features	PWA	Traditional App
Installation	Direct from browser	Download from app store
Internet Required	work offline with Caching	Usually requires internet
Performance	Fast with Service workers	Fast but need installation
updates	Automatic	Manual
Development Cost	Lower	Higher

Q.2. Define ~~regu~~ responsive web design and explain its importance in the context of progressive web apps. Compare and contrast responsive, fluid and adaptive web design approaches.

⇒ Definition of Responsive web design:-

Responsive web design (RWD) is a ~~techniq~~ technique that makes web Pages adjust automatically to different screen size and devices it ensures a good user experience on mobile on mobile, tablets and desktops without needing separate version of website.

Importance of Responsive Design in PWAs

1) Better user Experience
2) Faster Load time
3) SEO benefits
4) Cost effective

* Comparison :-

Approach	How it works	Pros	Cons
~~Res~~ Responsive	use flexible grids and css media queries	works on all devices	Com
Fluid	use % based widths and fixed pixels on different screen size	works well	?

Key Differences :-

1) Responsive adapts dynamically to all screens
2) Fluid resizes smoothly but may not be fully optimized
3) Adaptive loads different layouts based on device type.

Describes the lifecycle of service workers including registration, installation and activation phases

=> Lifecycle of Service workers :-

A services worker is a script that runs in the ~~browsers~~ background and helps a web app work offline, load faster and send push notifications, its lifecycle has three main phases

1) Registration Phase :- The browser registers the Service workers using javascript.

e.g.:

```
if ('serviceworker' in navigator) {
    navigator.serviceworker.register("/sw.js")
    .then () => console.log('Service Registered");
    .catch(error => console.log ("Registered Failed",
                                 errors));
}
```

2) Installation phase:
i) The service worker downloads necessary files (HTML, and stores the cache

ii) if successful, it moves to the activation phase.

Code eg:-

```
self.addEventListener('install', event => {
    event.waitUntil(
        caches.open('app-cache').then(cache =>
            return cache.addAll(['/', '/index.html',
                'style.css']);
        })
    );
});
```

3) Activation phase:
→ The old service worker is replaced with new o
→ unused cache files from the previous version are deleted.

Final step: Fetch & nine
Once activated, the service worker intercepts network request. Serves cached files and types data.

Explain the use of indexed DB in the service worker for data storage.

⟹ Use of IndexDB in service worker for Data storage:-
IndexedDB is a browser database that stores large amount of structured data like json objects.
It helps PWAs work offline by saving and networking data efficiency.

Why use Indexed DB in Service workers?

1) Offline Support:- store data when offline and sync it later.

2) Efficient storage:- Saves structured data like user setting, cart items or form inputs

3) Faster Access:- Retrive Data quickly without needing a network registers

4] Persistent Data:- Data remains saved even after the browser is closed.

How Service workers use IndexedDB?

Opening the Database

```
let db;
let request = indexedDB.open('MyDatabase', 1);
request.onsuccess = function(event) {
    db = event.target.result;
};
```

Creating a store and adding Data

```
request.onupgradeneeded = function(event) {
    let db = event.target.result;
    let store = db.createObjectStore('users', { keyPath
    store.add({ id: 1, name: 'John Doe', age: 25 });
};
```

Fetching Data in Service Worker.

```
let transaction = db.transaction('users', 'readonly');
let store = transaction.objectStore('user');
let getUser = store.get(1);
get user.onsuccess = function() {
    Console.log(get.User result);
};
```