1. Flutter "Hello World" Application

Aim:

To create a basic Flutter app displaying "Hello World" text.

Code:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Hello World App')),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Conclusion:

A simple Flutter application was successfully created displaying "Hello World" on the screen.

2. Increment and Decrement Counter using setstate

Aim:

To create a counter app where a number increases or decreases on button press using setState().

Code:

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

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

class CounterApp extends StatefulWidget {
  const CounterApp({super.key});

  @override
  State<CounterApp> createState() => _CounterAppState();
}

class _CounterAppState extends State<CounterApp> {
  int counter = 0;

  void increment() {
    setState(() {
      counter++;
    });
  }

  void decrement() {
    setState(() {
      counter--;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Counter App')),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
```

```
          children: [
            Text('Counter: $counter', style: const TextStyle(fontSize: 24)),
            const SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                ElevatedButton(onPressed: increment, child: const Text('Increment')),
                const SizedBox(width: 10),
                ElevatedButton(onPressed: decrement, child: const Text('Decrement')),
              ],
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Conclusion:

The counter application demonstrated updating UI based on user interactions using setState().

3. Flutter Form and Form Validator

Aim:

To create a simple form with validation in Flutter.

Code:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FormScreen(),
    );
  }
}

class FormScreen extends StatefulWidget {
  @override
  State<FormScreen> createState() => _FormScreenState();
}

class _FormScreenState extends State<FormScreen> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Form Validator')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                controller: _nameController,
                decoration: const InputDecoration(labelText: 'Enter your name'),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Please enter name';
                  }
                  return null;
                },
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
```

```
              if (_formKey.currentState!.validate()) {
                ScaffoldMessenger.of(context)
                    .showSnackBar(const SnackBar(content: Text('Form Submitted')));
              }
            },
            child: const Text('Submit'),
          )
        ],
      ),
    ),
  ),
 );
 }
}
```

Conclusion:

The form was successfully created with validation ensuring user input correctness.

4. State Management using BLoC and Provider

Aim:To understand and implement state management using Provider and BLoC.

Code :

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

void main() {
  runApp(ChangeNotifierProvider(
    create: (context) => Counter(),
    child: const ProviderApp(),
  ));
}

class Counter with ChangeNotifier {
  int value = 0;
  void increment() {
    value++;
    notifyListeners();
  }
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Provider Example')),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Consumer<Counter>(
                builder: (context, counter, _) =>
                    Text('Counter: ${counter.value}', style: const TextStyle(fontSize: 24)),
              ),
              ElevatedButton(
                onPressed: () => context.read<Counter>().increment(),
                child: const Text('Increment'),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

Conclusion: Provider was successfully used to manage and update state in a reactive way across widgets.

5. Flutter Animation Example

Aim:

To implement simple animations in Flutter.

Code:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: AnimatedScreen(),
    );
  }
}

class AnimatedScreen extends StatefulWidget {
  @override
  State<AnimatedScreen> createState() => _AnimatedScreenState();
}

class _AnimatedScreenState extends State<AnimatedScreen> {
  double size = 100.0;

  void animate() {
    setState(() {
      size = size == 100 ? 200 : 100;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Simple Animation')),
      body: Center(
        child: GestureDetector(
          onTap: animate,
          child: AnimatedContainer(
            width: size,
            height: size,
            color: Colors.blue,
            duration: const Duration(seconds: 1),
            curve: Curves.easeInOut,
          ),
        ),
      ),
```

```
    );
  }
}
```

Conclusion:

Animations were used to smoothly transition widget properties like size and color.

6. Routing, Passing Data using Navigator and NavigationRail

Aim:

To implement navigation between pages and pass data using Navigator and NavigationRail.

Code:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: NavigationHome(),
    );
  }
}

class NavigationHome extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Home Page')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.push(context, MaterialPageRoute(
              builder: (context) => SecondPage(data: "Hello from Home!"),
            ));
          },
          child: const Text('Go to Second Page'),
        ),
      ),
    );
  }
}

class SecondPage extends StatelessWidget {
  final String data;
  const SecondPage({super.key, required this.data});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Second Page')),
```

```
      body: Center(child: Text(data, style: const TextStyle(fontSize: 24))),
    );
  }
}
```

Conclusion:

Navigation and data passing between screens was successfully demonstrated.

7. Simple Hello World PWA

Aim:

To create a basic Progressive Web App that says "Hello World".

Code (index.html + manifest.json):

Html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World PWA</title>
  <link rel="manifest" href="manifest.json">
</head>
<body>
  <h1>Hello World PWA</h1>
  <script>
    if ('serviceWorker' in navigator) {
      navigator.serviceWorker.register('/service-worker.js');
    }
  </script>
</body>
</html>
```

Json:

```
// manifest.json
{
  "name": "Hello World PWA",
  "short_name": "PWA",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#000000",
  "icons": [{
    "src": "icon.png",
    "sizes": "192x192",
    "type": "image/png"
  }]
}
```

Conclusion:

Basic PWA setup was achieved with manifest and service worker registration.

8. Service Worker Install and Activate (E-commerce PWA)

Aim:

To code and register a service worker with install and activate events.

Code (service-worker.js):

```
self.addEventListener('install', (event) => {
  console.log('Service Worker Installed');
});

self.addEventListener('activate', (event) => {
  console.log('Service Worker Activated');
});
```

Conclusion:

Service worker registered, installed, and activated successfully.

9. Service Worker Events: Fetch, Sync, Push (E-commerce PWA)

Aim:

To implement Fetch, Sync, and Push events in service worker for offline-first support.

Code (service-worker.js):

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

self.addEventListener('sync', (event) => {
  if (event.tag === 'syncData') {
    console.log('Sync event triggered');
    // Background sync logic here
  }
});

self.addEventListener('push', (event) => {
  const data = event.data.json();
  self.registration.showNotification(data.title, {
    body: data.body,
  });
});
```

Conclusion:

Fetch for offline, Sync for background updates, and Push notifications were integrated into the PWA successfully.