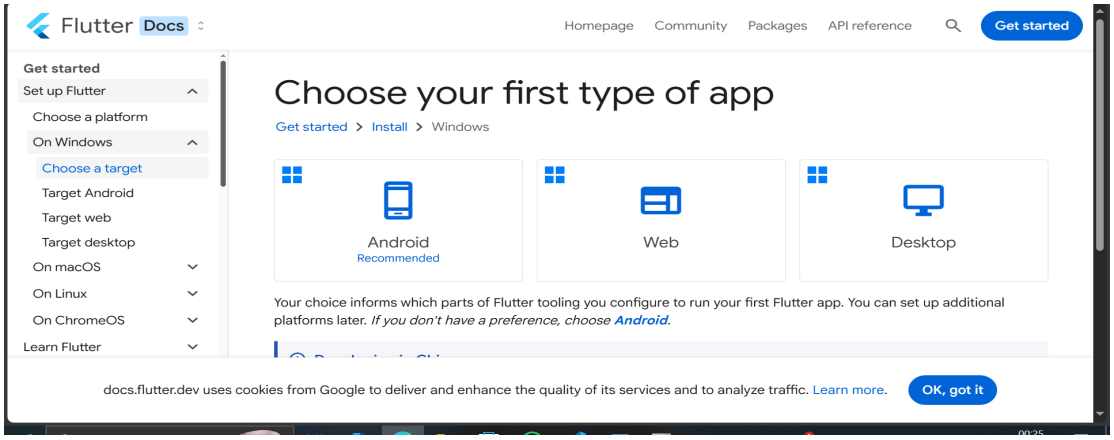
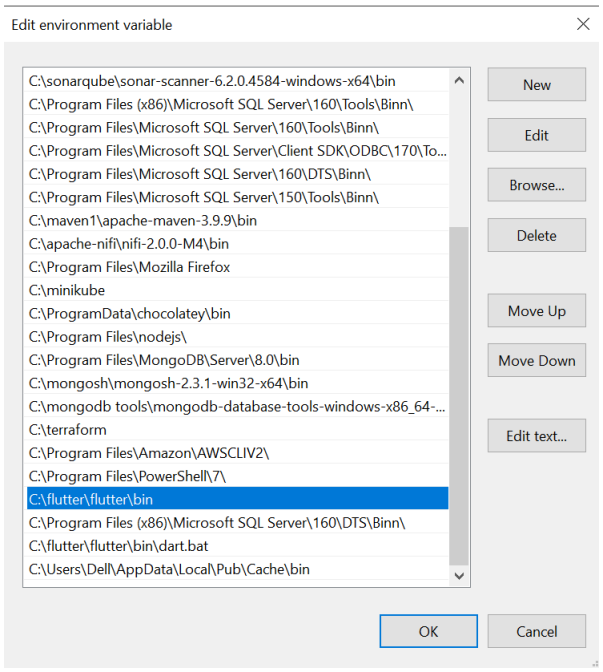
Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 01
Aim: To install and configure the Flutter Environment

Theory:

Flutter is an open-source UI toolkit from Google used for building cross-platform apps. Setting up Flutter involves installing the SDK, configuring environment variables, and setting up an IDE like VS Code or Android Studio.





```
Command Prompt - flutter
Microsoft Windows [Version 10.0.19045.5679]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dell>flutter

 A new version of Flutter is available!

 To update to the latest version, run "flutter upgrade".

Manage your Flutter app development.

Common commands:

  flutter create <output directory>
    Create a new Flutter project in the specified directory.

  flutter run [options]
    Run your Flutter application on an attached device or in an emulator.

Usage: flutter <command> [arguments]

Global options:
-h, --help                  Print this usage information.
-v, --verbose               Noisy logging, including all shell commands executed.
                            If used with "--help", shows hidden options. If used with "flutter doctor", shows additional                        diagnostic information.
(Use "-vv" to force verbose logging in those cases.)
-d, --device-id             Target device id or name (prefixes allowed).
    --version               Reports the version of this tool.
    --enable-analytics      Enable telemetry reporting each time a flutter or dart command runs.
    --disable-analytics     Disable telemetry reporting each time a flutter or dart command runs, until it is
                            re-enabled.
    --suppress-analytics    Suppress analytics reporting for the current CLI invocation.

Available commands:

Flutter SDK
  bash-completion   Output command line shell completion setup scripts.
  channel           List or switch Flutter channels.
  config            Configure Flutter settings.
  doctor            Show information about the installed tooling.
```

Conclusion: The Flutter environment was successfully installed and configured, providing a solid foundation for cross-platform mobile app development. With Flutter, Android Studio, and the required SDKs in place, we are now equipped to start building and testing mobile applications efficiently. The setup ensures a smooth workflow for writing, debugging, and deploying apps on both Android and iOS platforms.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

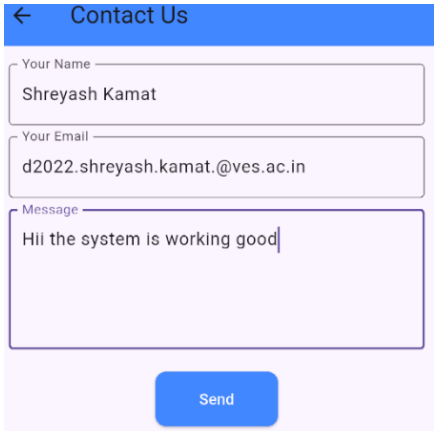
Experiment 02

Aim: To design Flutter UI by including common widgets.

Theory:

Flutter provides a rich set of widgets such as Text, Container, Row, Column, ListView, etc., that allow flexible UI design. These widgets form the building blocks of any Flutter interface.In the contact us form, multiple commonly used Flutter widgets have been utilized to create an interactive and well-structured UI:

- **TextField**: Used to capture user input such as email, name, and flat number. These are interactive fields allowing text input.
- **DropdownButtonFormField**: Used to allow users to select options from a dropdown list — for example, selecting "Resident" or "Manager" as the user type.
- **Column**: Used to align widgets vertically — making sure the form fields are placed one below the other in a clean stacked layout.
- **Row**: Used wherever horizontal alignment is required, such as displaying form elements side-by-side or organizing parts of the layout.
- **Padding and SizedBox**: Used for spacing and improving layout aesthetics — giving proper margin and breathing room between widgets.
- **ElevatedButton**: A clickable button that triggers actions, such as submitting the registration form.
- **Form and GlobalKey<FormState>**: While not visual widgets, they are crucial for validating the form and maintaining form state.
- **Responsive Design Considerations**: The use of flexible layouts and widgets ensures that the form looks good on different screen sizes.



Conclusion:

The experiment introduced **Flutter widgets** and their role in UI building. Using basic widgets like **Text**, **Container**, **Row**, **Column**, **AppBar**, and **ElevatedButton**, a simple interactive UI was created—laying the groundwork for more advanced Flutter designs.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 03

Aim: To include icons, Prerequisite, fonts in Flutter app

Theory:

In Flutter, visual consistency and user engagement are enhanced by incorporating icons and custom fonts into an application. These elements contribute to the overall branding and aesthetic appeal of the app.

To include such assets, the pubspec.yaml file acts as the central configuration hub.

Required Flutter packages such as cupertino_icons must be specified in the dependencies section of the pubspec.yaml.

Assets like fonts or icon files must be placed in appropriate folders within the project directory.

```
dependencies:

  cupertino_icons: ^1.0.8
```

- ➔ This line ensures that Cupertino icons are available, enabling the use of iOS-style icons throughout the app.
- ➔ uses-material-design: true under the flutter: section enables the use of Google's Material Icons in the app.

```
flutter:

  uses-material-design: true
```

To include fonts

```
flutter:

  uses-material-design: true

  fonts:

    - family: Roboto

      fonts:

        - asset: assets/fonts/Roboto-Regular.ttf
```

Conclusion:

By integrating icons, images, and custom fonts, we took a big step toward making our Flutter app visually appealing and polished. These elements go beyond basic layouts — they bring personality, brand identity, and clarity to the user interface. Understanding how to add and manage assets properly sets the stage for designing apps that *look* professional and *feel* complete.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 04

Aim: To create an interactive Form using form widget

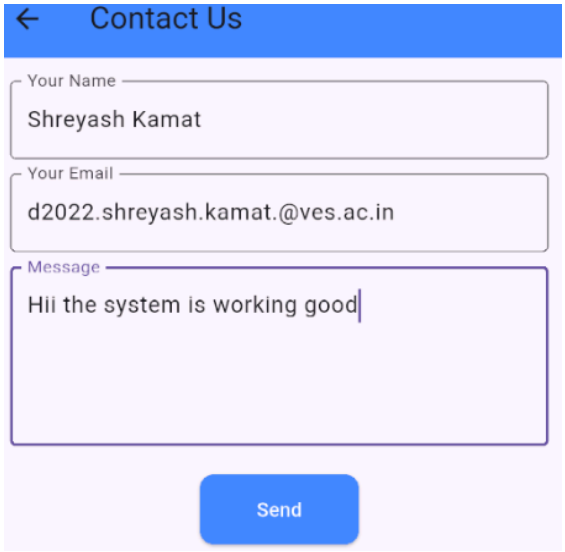
Theory:

Flutter forms use Form, TextFormField, and GlobalKey<FormState> for validation and user input handling. They are essential for collecting structured user data.

- I am using a Form widget with GlobalKey<FormState>.
- Validation, conditional fields (e.g., email messages), and user input are handled properly.

In registration form:

- A Form widget is wrapped around the entire input structure to maintain a unified validation context.
- A GlobalKey<FormState> is used to validate and manage form state globally when the form is submitted.
- Various TextFormField widgets are used to collect user inputs like name, email and messages.
- Validation logic is applied to fields to ensure correctness of data (e.g., checking if email is valid, or mandatory fields are not left blank).



Conclusion:

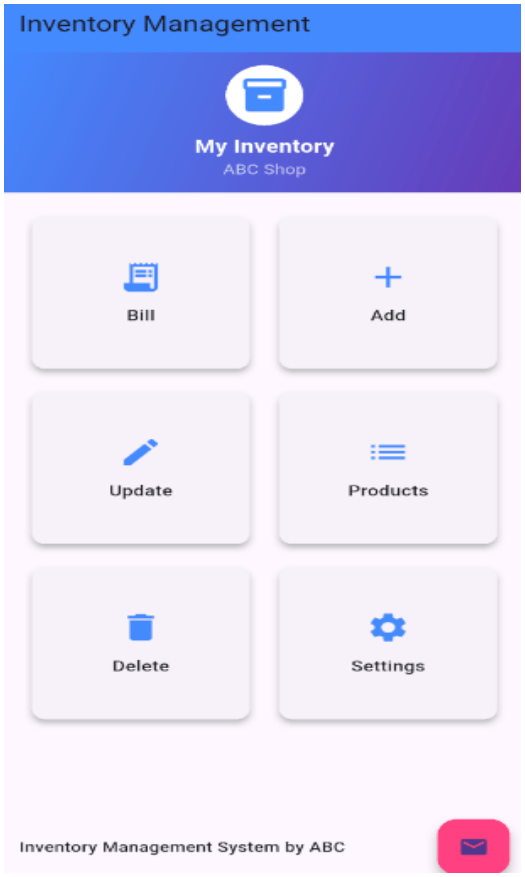
Creating interactive forms in Flutter is crucial for collecting structured user input. This experiment helped in understanding how to use Form, TextFormField, and validation techniques to ensure the input is accurate and complete. Mastering forms is a must-have skill for any real-world app that involves user interaction.

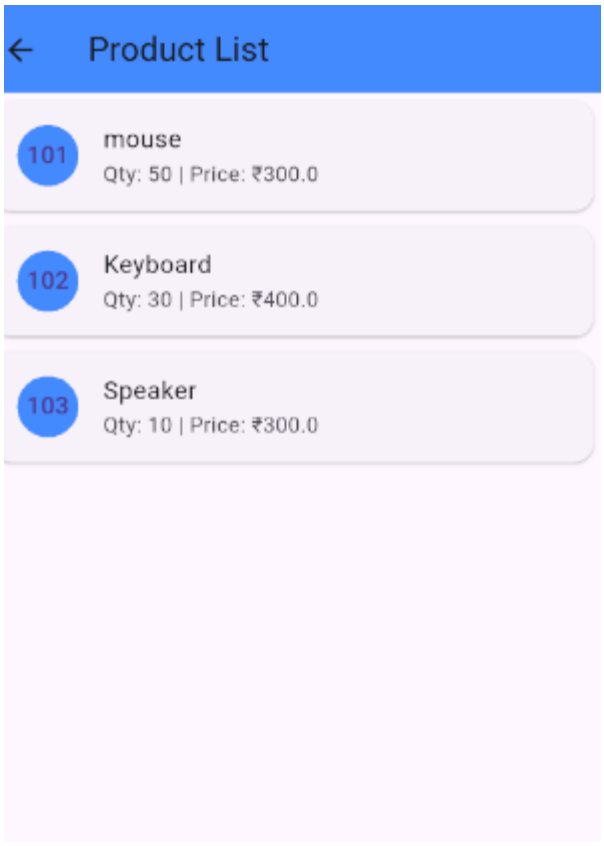
Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 05
Aim: To apply navigation, routing and gestures in Flutter App

Theory:
Flutter uses Navigator for route management and gesture widgets like GestureDetector for user interaction. This provides seamless screen transitions and intuitive UX.

- Navigation to Different pages after clicking on the buttons.
- You might also have gestures like onTap or onPressed, though not clearly visible here.





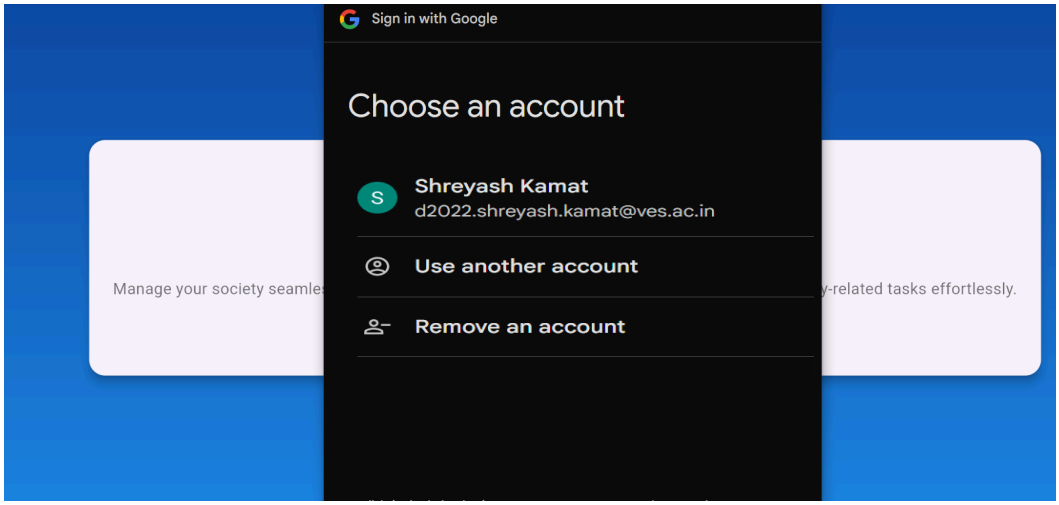
Conclusion:
Navigation and gestures are what make an app feel alive — not just a bunch of screens. In this experiment, we learned how to switch between pages smoothly using routing and how to respond to user actions like taps or swipes using gesture detectors. These features are the backbone of any real-world app — without them, you're just stuck on a single screen with zero vibes. Now, we're building apps that actually *flow*.

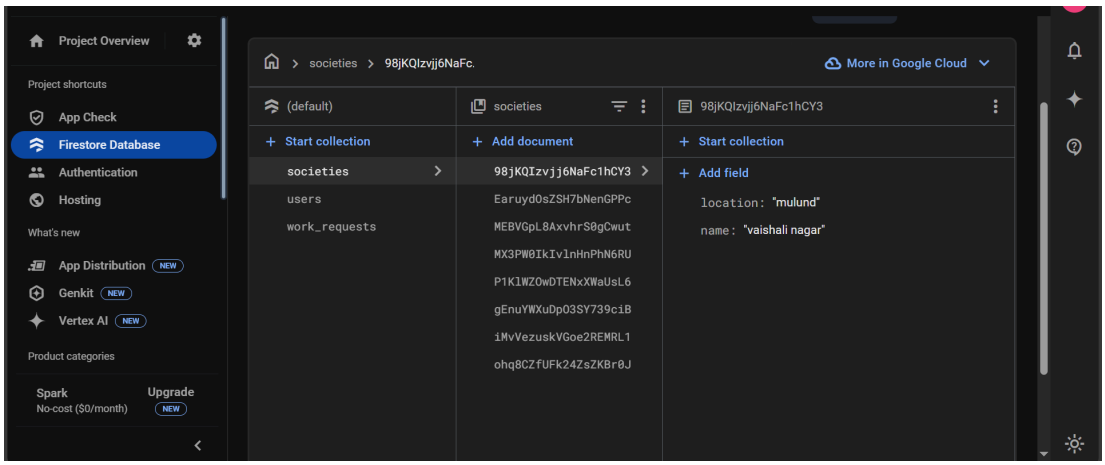
Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 06
Aim: To Connect Flutter UI with fireBase database

Theory:
Firebase offers a backend-as-a-service including authentication and Firestore database. Integration allows storing, updating, and retrieving user data in real-time.

- Firebase Auth is used for sign-in.
- Firestore is used to store product details.





Conclusion:
Integrating Firebase with Flutter enabled real-time, cloud-based features. In this experiment, we connected the app to Firebase and performed data store and retrieve operations—paving the way for dynamic, user-focused apps with instant updates across devices.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 07

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

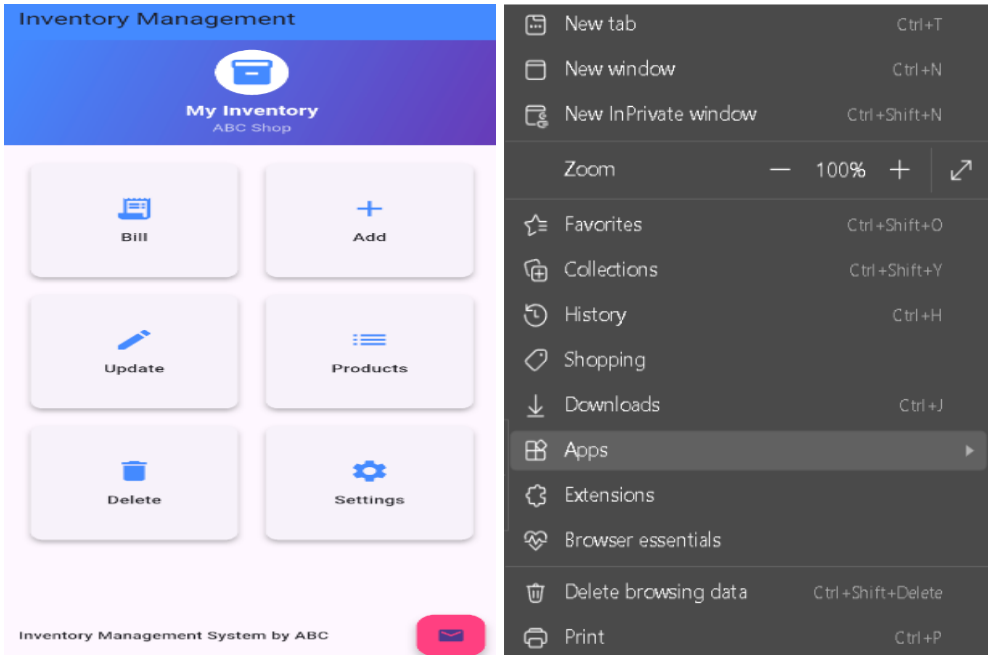
Theory:

Progressive Web Apps (PWAs) use a **Web App Manifest**, a JSON file that includes key metadata like:

- App name & short name
- Start URL & scope
- Icons for various devices
- Theme & background colors
- Display mode (e.g., standalone)

Linking manifest.json to the HTML makes the app installable and adds "Add to Home Screen" functionality, giving the Ecommerce platform an app-like experience and boosting engagement and branding.

Output:



Conclusion:

In this experiment, we created and integrated a Web App Manifest for the PWA, adding metadata like name, icons, theme color, and display mode. This enabled "Add to Home Screen", offering a native-like experience and boosting user engagement.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 08

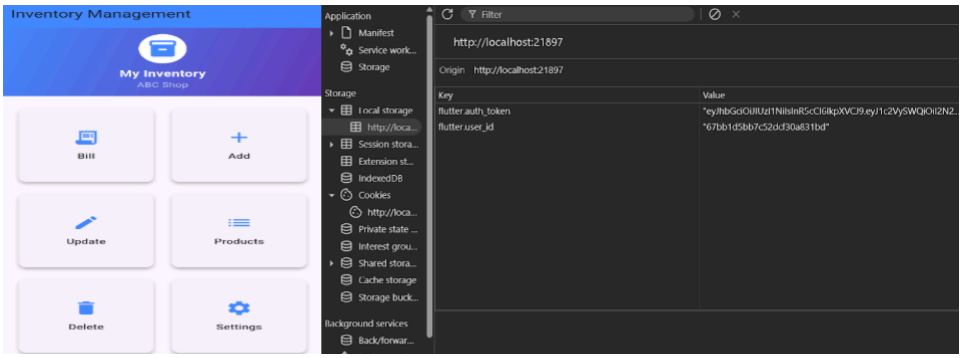
Aim: To code and register a service worker, and complete the install and activation process for a new service worker for the E-commerce PWA

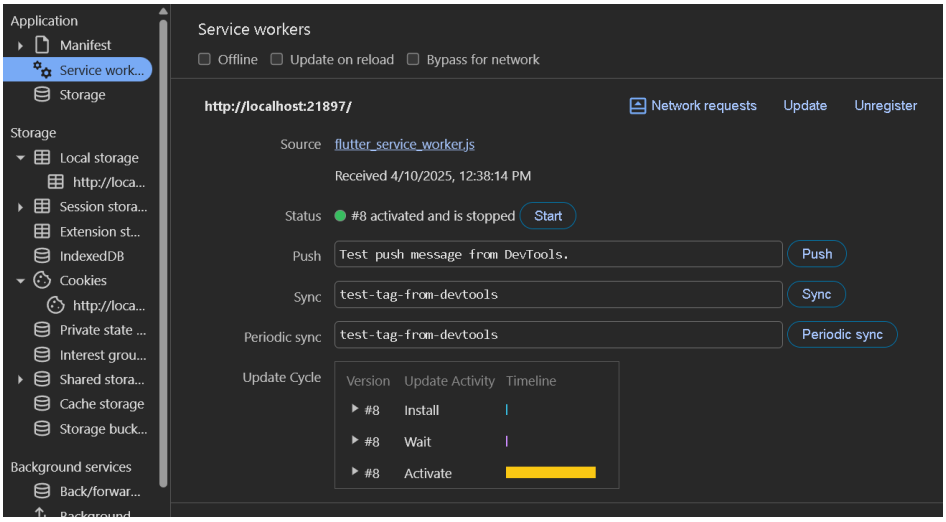
Theory:

Service Worker is a background script that powers key PWA features like offline support, background sync, and push notifications. It goes through three phases:

- Install: Caches essential assets.
- Activate: Cleans up old caches.
- Fetch: Intercepts requests to serve cached content, enabling offline use.

Output:





Conclusion:

In this experiment, we implemented a Service Worker in our PWA to enhance performance and offline functionality. We successfully coded, registered, and completed the install and activate phases. This setup ensures that our app can cache key resources, load faster, and work in low or no network conditions, ultimately offering a more reliable and app-like experience to users.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 09

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

Theory:

A **Service Worker** is a background JavaScript file in a PWA that acts as a proxy between the app and the network. It enables:

- Offline caching (fetch event)
- Background sync (sync event)
- Push notifications (push event)

These features enhance speed, updates, and user engagement, even with poor or no internet.

Output:

Fetch Event:

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

Sync Event:

```
self.addEventListener('sync', (event) => {
  if (event.tag === 'sync-data') {
    event.waitUntil(syncDataWithServer());
  }
});
```

Push Event

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

Conclusion:

In this experiment, we successfully implemented the core Service Worker events (fetch, sync, and push). This enhanced the app's ability to:

- Work offline using cache (fetch)
- Automatically sync data in the background (sync)
- Engage users with notifications (push)

These features are crucial for improving reliability, performance, and user engagement in modern web applications.

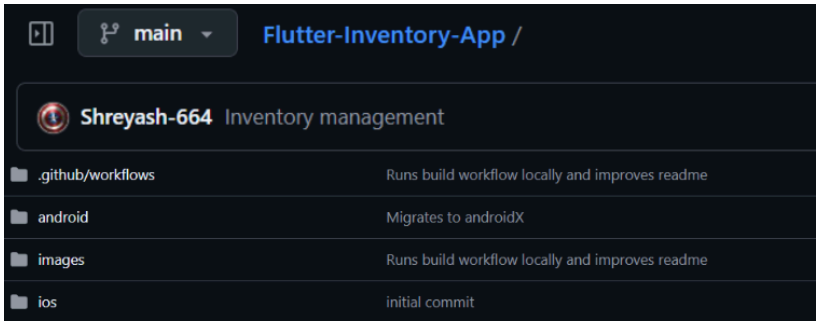
Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

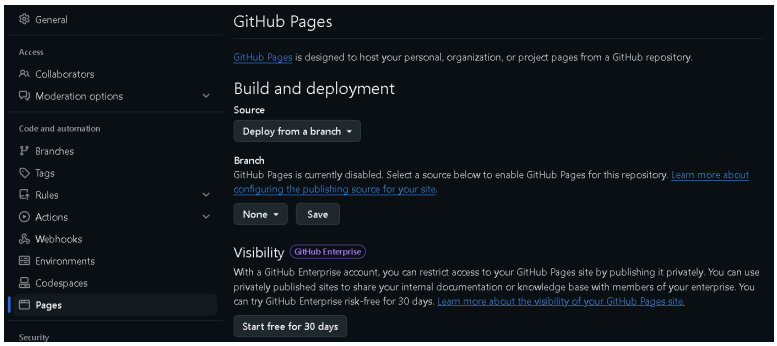
Experiment 10

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

Theory:

GitHub Pages is a free hosting platform by GitHub for static websites. It can serve HTML, CSS, JavaScript, and service worker files, making it a suitable option for deploying Progressive Web Apps (PWAs).





Conclusion: Deploying the PWA on GitHub Pages made it publicly accessible with offline access and fast loading.It also simplified updates, version control, and sharing for real-world testing.

Name: **Shreyash Kamat**	Div-Roll no: **D15C-22**
DOP:	DOS:
Sign:	Grade:

Experiment 11

Aim: To use google Lighthouse PWA Analysis Tool to test the PWA functioning.

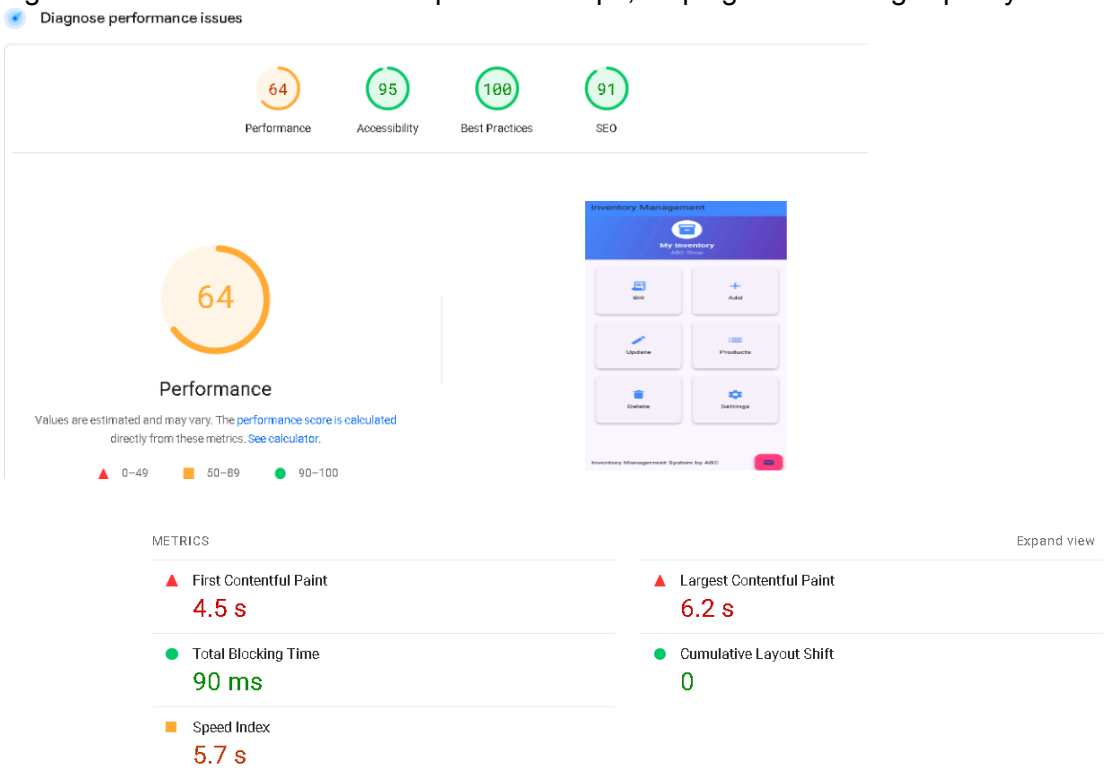
Theory:

Google Lighthouse is an open-source tool that audits web apps for performance, accessibility, SEO, best practices, and PWA standards.

For PWAs, it checks:

- Web App Manifest
- Service Worker setup
- HTTPS usage
- Responsive & offline functionality
- “Add to Home Screen” support

It gives a score out of 100 with improvement tips, helping ensure a high-quality user experience.



Conclusion:

Using Google Lighthouse, we successfully analyzed the PWA capabilities of App The tool helped us verify important aspects like offline access, manifest configuration, and service worker functionality. It also provided valuable suggestions to optimize user experience and app performance, making it a vital step in PWA development and deployment.