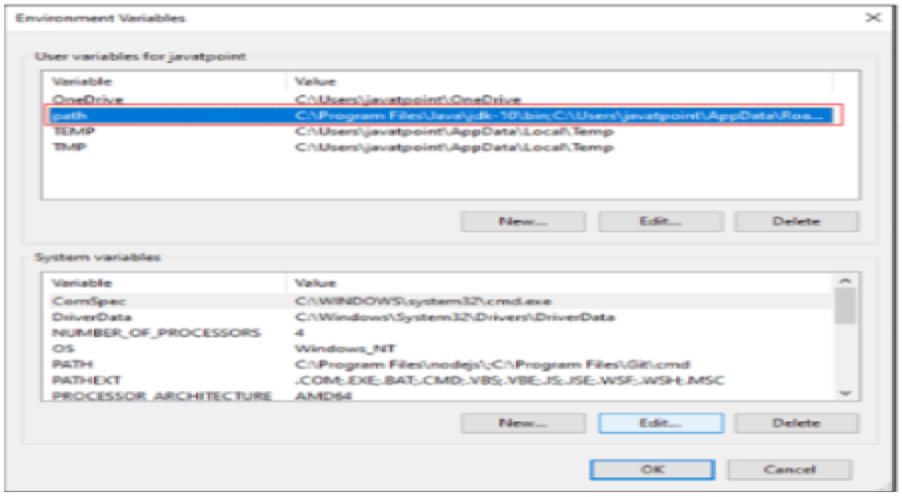
EXPERIMENT 1	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

Aim:Installation and Configuration of Flutter Environment.

Theory:The installation and configuration of the Flutter environment involve setting up the necessary tools to start developing cross-platform apps. It begins with downloading the Flutter SDK from the official Flutter website and extracting it to a suitable directory. The SDK's bin folder is then added to the system's PATH variable to enable access to Flutter commands globally. Developers also need to install an IDE like Visual Studio Code or Android Studio with the Flutter and Dart plugins. Running flutter doctor in the terminal helps verify the setup and highlights any missing dependencies. This setup ensures a ready environment for building and running Flutter apps on various platforms.

Output:

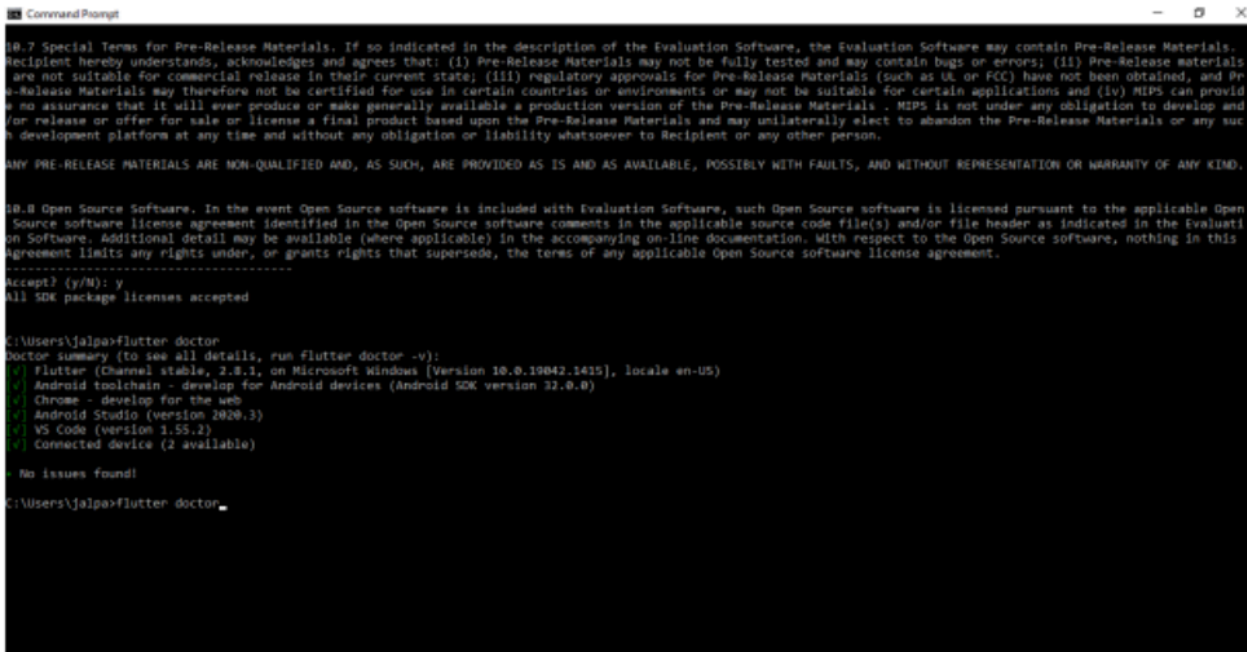


User variables for javatpoint

Variable	Value
OneDrive	C:\Users\javatpoint\OneDrive
path	C:\Program Files\Java\jdk-10\bin;C:\Users\javatpoint\AppData\Roa...
TEMP	C:\Users\javatpoint\AppData\Local\Temp
TMP	C:\Users\javatpoint\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
PATH	C:\Program Files\nodejs\;C:\Program Files\Git\cmd
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64



```
10.7 Special Terms for Pre-Release Materials. If so indicated in the description of the Evaluation Software, the Evaluation Software may contain Pre-Release Materials.
Recipient hereby understands, acknowledges and agrees that: (i) Pre-Release Materials may not be fully tested and may contain bugs or errors; (ii) Pre-Release materials
 are not suitable for commercial release in their current state; (iii) regulatory approvals for Pre-Release Materials (such as UL or FCC) have not been obtained, and Pr
e-Release Materials may therefore not be certified for use in certain countries or environments or may not be suitable for certain applications and (iv) MIPS can provid
e no assurance that it will ever produce or make generally available a production version of the Pre-Release Materials . MIPS is not under any obligation to develop and
/or release or offer for sale or license a final product based upon the Pre-Release Materials and may unilaterally elect to abandon the Pre-Release Materials or any suc
h development platform at any time and without any obligation or liability whatsoever to Recipient or any other person.

ANY PRE-RELEASE MATERIALS ARE NON-QUALIFIED AND, AS SUCH, ARE PROVIDED AS IS AND AS AVAILABLE, POSSIBLY WITH FAULTS, AND WITHOUT REPRESENTATION OR WARRANTY OF ANY KIND.


10.8 Open Source Software. In the event Open Source software is included with Evaluation Software, such Open Source software is licensed pursuant to the applicable Open
 Source software license agreement identified in the Open Source software comments in the applicable source code file(s) and/or file header as indicated in the Evaluati
on Software. Additional detail may be available (where applicable) in the accompanying on-line documentation. With respect to the Open Source software, nothing in this
Agreement limits any rights under, or grants rights that supersede, the terms of any applicable Open Source software license agreement.
---------------------------------------
Accept? (y/N): y
All SDK package licenses accepted


C:\Users\jalpa>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.19042.1415], locale en-US)
[√] Android toolchain - develop for Android devices (Android SDK version 32.0.0)
[√] Chrome - develop for the web
[√] Android Studio (version 2020.3)
[√] VS Code (version 1.55.2)
[√] Connected device (2 available)

• No issues found!

C:\Users\jalpa>flutter doctor
```

<u>Conclusion</u>:In conclusion, properly installing and configuring the Flutter environment is a crucial first step in mobile and cross-platform app development. It ensures that all necessary tools and dependencies are in place, allowing developers to build, test, and run Flutter applications smoothly. With the right setup, developers can fully leverage Flutter's powerful features and create high-performance apps with a single codebase.

.

EXPERIMENT 2	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

Aim:To design Flutter UI by including common widgets.

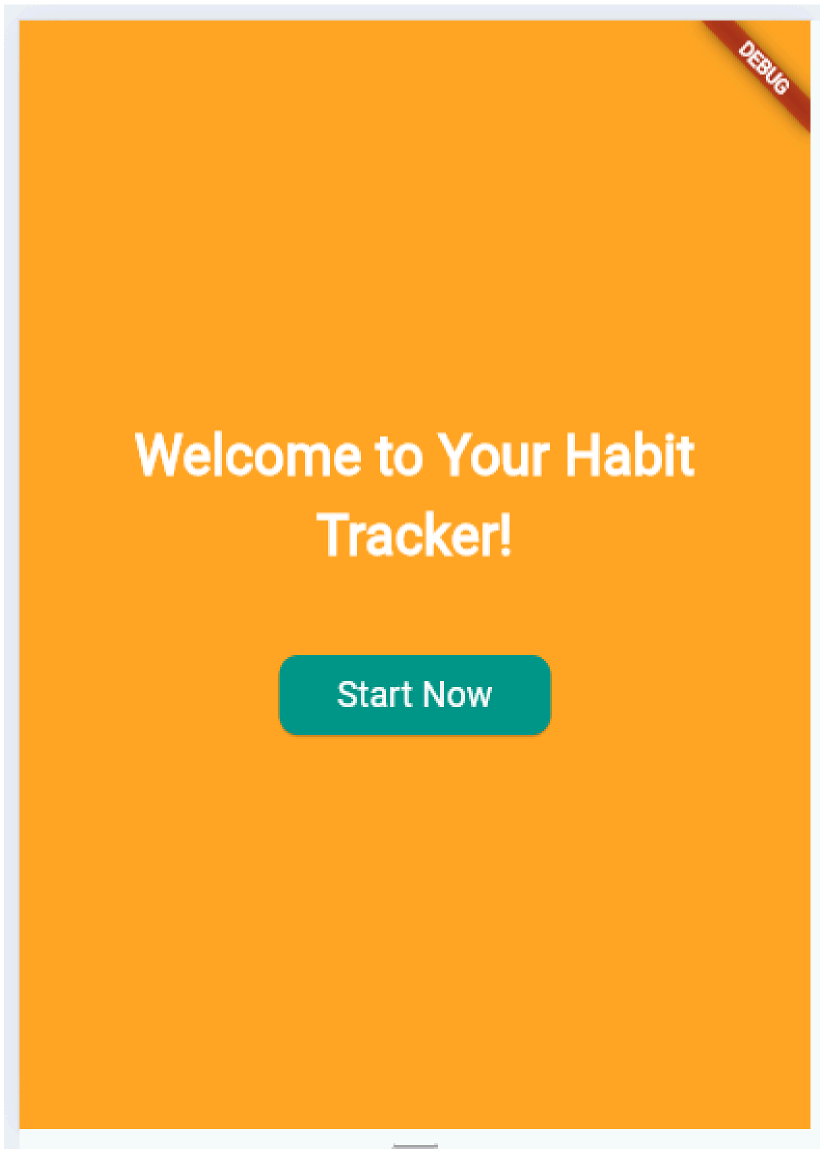
Github link: https://github.com/dimpledalwani/mpl.git

Theory :The **LandingPage** in the **FitSync app** serves as the first point of interaction for users and reflects the goal of designing intuitive and visually appealing Flutter UIs using commonly used widgets. By utilizing core Flutter widgets such as Scaffold, Column, Text, Image.asset, and ElevatedButton, the page is structured to offer both functionality and a welcoming aesthetic.

The Scaffold provides the foundational layout, ensuring consistency across the app. The Column widget arranges the content vertically, organizing elements like the welcome text, logo or image, and action buttons in a clean and readable manner. The Text widget delivers a friendly greeting or branding message that introduces users to the app's purpose—syncing their fitness goals and progress. The use of Image.asset adds a visual appeal, often displaying a relevant illustration or logo that reinforces the app's identity. An ElevatedButton is prominently placed to allow users to proceed further—typically by navigating to the login or signup screen.

This combination of widgets not only contributes to a smooth and interactive user experience but also sets the tone for the rest of the application. It establishes clarity, ease of navigation, and visual consistency—important principles in UI design. Overall, the **LandingPage** plays a vital role in engaging users from the moment they open the app, making it a key element in FitSync's user interface strategy.

Output:



Conclusion:In conclusion, the LandingPage of the FitSync app effectively demonstrates how Flutter’s common widgets can be used to design a clean, interactive, and user-friendly UI. By combining layout, text, images, and navigation elements, the page sets the foundation for a smooth user experience. This approach highlights the importance of using standard Flutter widgets to build responsive and visually appealing app interfaces.

EXPERIMENT 3	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

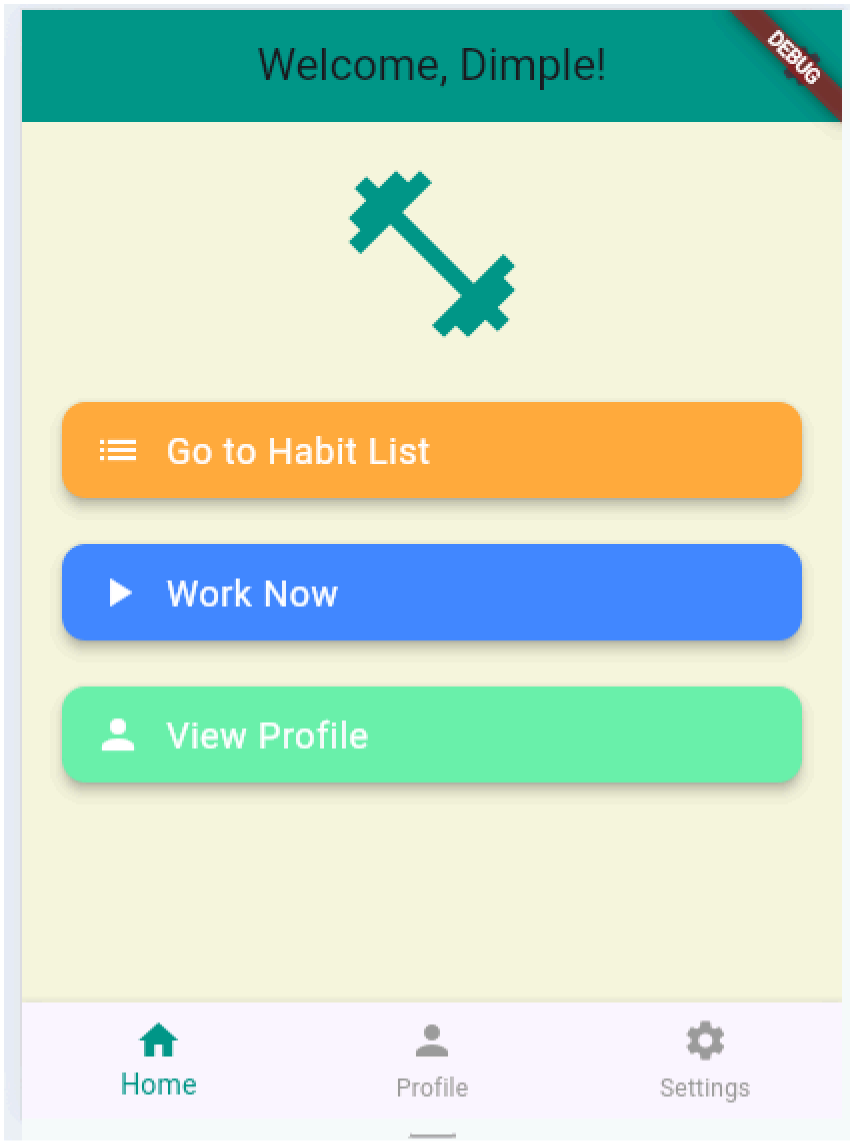
Github link: https://github.com/dimpledalwani/mpl.git

Theory:The aim of including **icons, images, and custom fonts** is clearly demonstrated in the **FitSync app**, contributing to a visually appealing and user-friendly interface. On the **HomePage**, multiple icons are used strategically to enhance navigation and help users quickly understand the function of different features. These icons improve usability and give the app a modern, intuitive feel.

Custom fonts are applied throughout the app using styled `Text` widgets, adding a sense of branding and ensuring visual consistency across different screens. The **LandingPage** effectively incorporates an asset image, which serves as a warm and engaging visual introduction to the app, helping to establish an emotional connection with users.

Together, these elements not only improve the aesthetics of the UI but also enrich the overall user experience. This reflects a thoughtful approach to design and highlights Flutter's powerful widget and styling capabilities in building polished and interactive mobile applications.

Output:



Conclusion:In conclusion, the FitSync app successfully demonstrates the use of icons, images, and fonts to enhance the overall user interface. By integrating these elements, the app becomes more visually appealing, user-friendly, and interactive. This implementation reflects good Flutter design practices and contributes to an engaging user experience.

EXPERIMENT 4	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

Aim:To create an interactive Form using form widget

Github Link:https://github.com/dimpledalwani/mpl.git

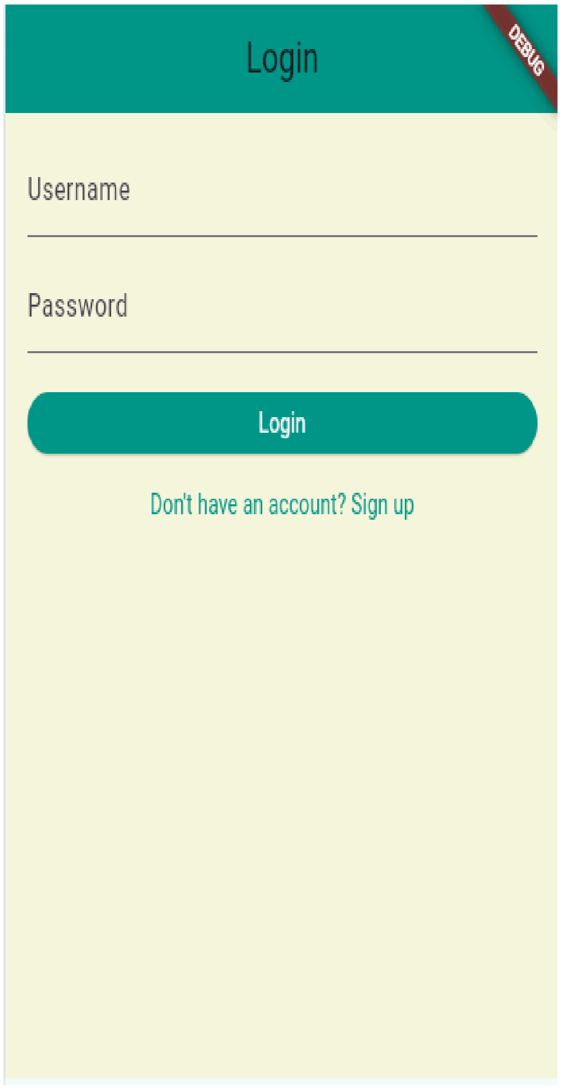
Theory :The aim **"To create an interactive Form using the Form widget"** is effectively implemented in the **FitSync app** through its user authentication process. The AuthPage utilizes the Form widget to gather essential user information such as **username, email, and password** in a structured and user-friendly manner.

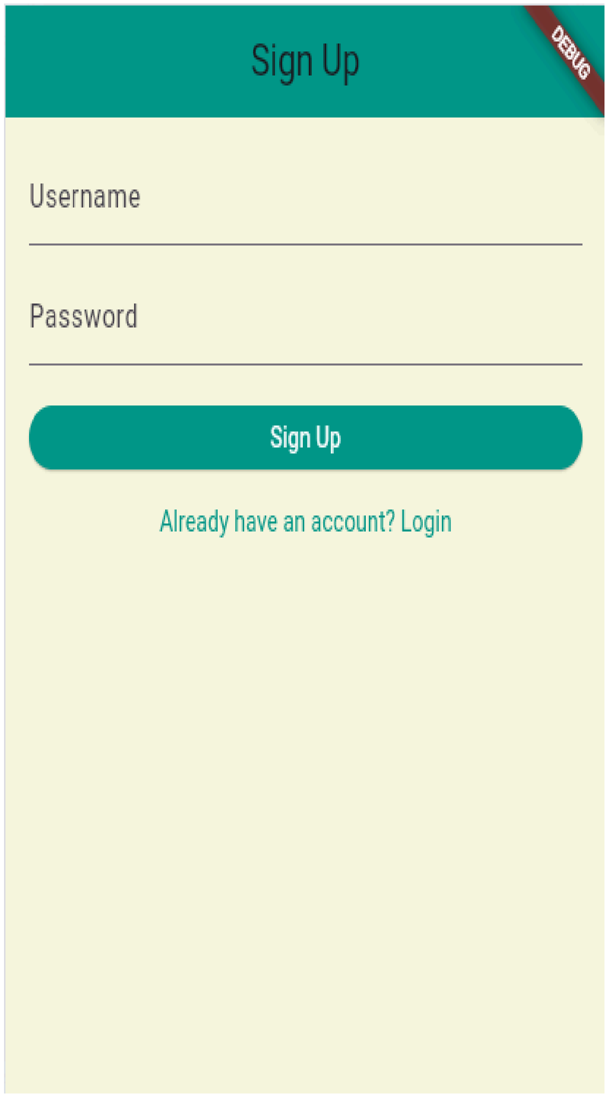
Inside the form, multiple TextFormField widgets are used to accept input, each equipped with **validation logic** to ensure users enter valid data. For example, the email field checks for proper formatting, and the password field may enforce minimum length or character requirements. This helps prevent errors and guides users in real-time as they fill out the form.

Interactive buttons are also included to **submit the form or navigate** between login and signup screens, enhancing the overall flow. The combination of validation, feedback, and clear navigation makes the form highly interactive and user-focused.

Overall, this implementation showcases Flutter's form handling capabilities and highlights how thoughtful UI design can improve user experience by ensuring a **smooth, reliable, and intuitive registration and login process**.

Output:





Conclusion:In conclusion, the FitSync app effectively utilizes the Form widget to create interactive and user-friendly authentication forms. By incorporating input fields with validation and submission logic, the app ensures accurate data collection and smooth user interaction. This implementation highlights the importance of form handling in building secure and responsive Flutter applications.

EXPERIMENT 5	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

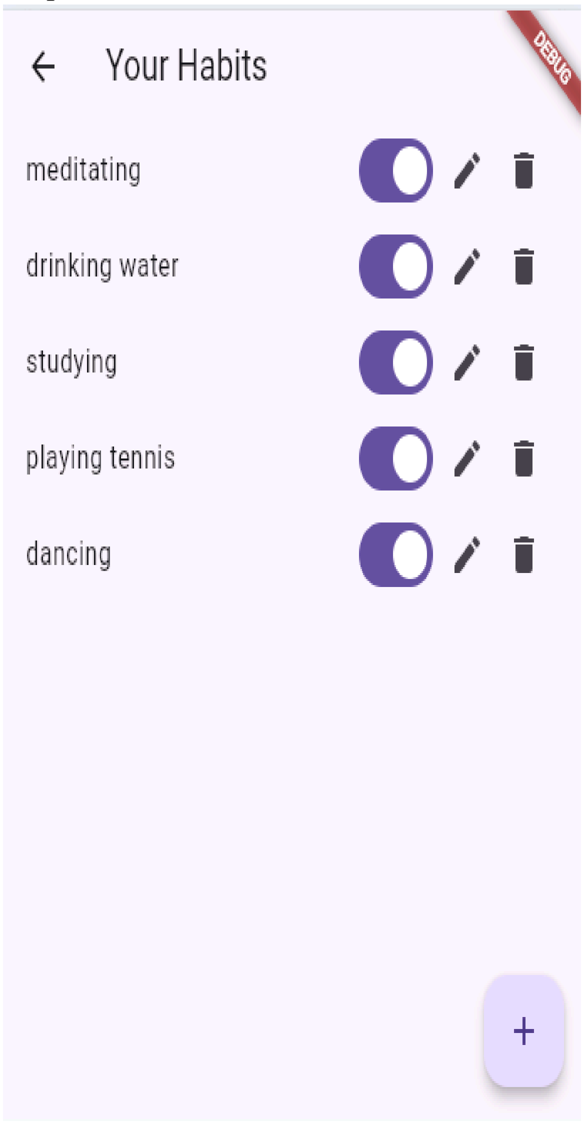
Github link:https://github.com/dimpledalwani/mpl.git

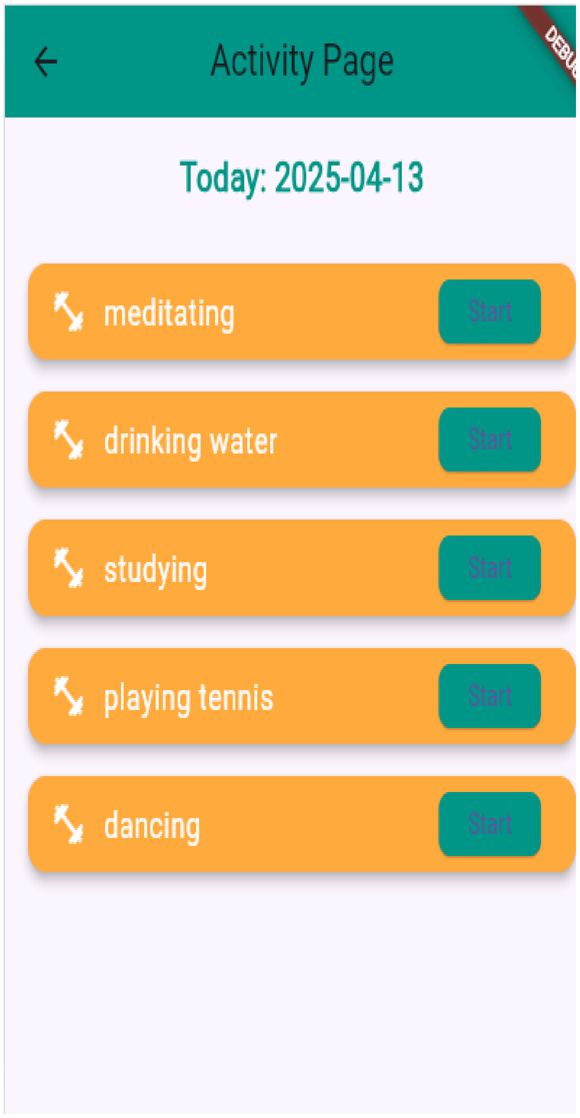
Theory:The aim of applying **navigation, routing, and gestures** is effectively implemented in the **HabitListPage** of the **FitSync app**, enhancing the overall interactivity and user experience. This page allows users to manage their habits with ease by utilizing **tap gestures** on list items and action buttons. Users can tap on a habit to **view its details**, **edit information**, or **delete it**, making the process of habit tracking seamless and engaging.

Navigation between different pages, such as returning to the **HomePage** or moving to an **update screen**, is handled using Flutter's built-in Navigator class. This enables smooth routing and page transitions, maintaining a natural flow throughout the app. For example, tapping an "Add Habit" or "Edit" button triggers the Navigator.push method to take the user to the respective form pages, while Navigator.pop helps in returning to the previous screen after an action is completed.

These interactive elements—combined with gesture detection and efficient routing—create an intuitive user interface where users feel in control. The thoughtful use of navigation and gestures not only streamlines the habit management process but also highlights the flexibility and power of Flutter's navigation and gesture handling capabilities.

Output:





Conclusion:In conclusion, the FitSync app effectively demonstrates the use of navigation, routing, and gesture handling. By using Flutter’s Navigator and gesture-responsive widgets like ListTile and ElevatedButton, the app ensures a smooth and interactive user experience. This implementation enhances usability by allowing users to navigate seamlessly between pages and perform actions through intuitive touch gestures.

EXPERIMENT 6	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

Aim:To set up Firebase with Flutter for iOS and Android Apps.

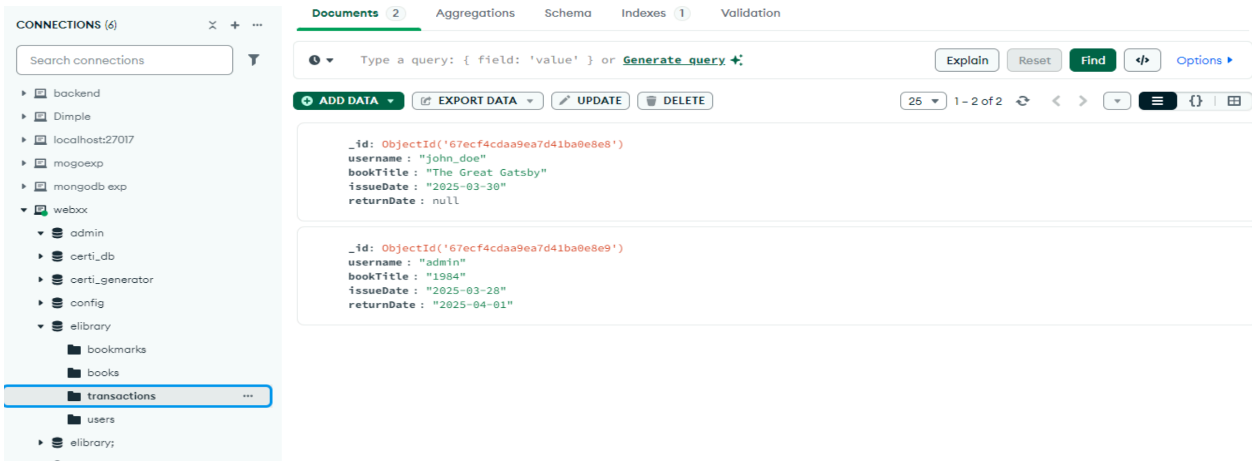
Github link:https://github.com/dimpledalwani/mpl.git

Theory:The aim of **setting up Firebase with Flutter** is successfully achieved in the **FitSync app**, providing robust backend support for **user authentication** and **data storage**. By integrating Firebase, the app benefits from a reliable and scalable cloud-based platform that simplifies the management of user accounts and habit-related data across both **Android and iOS** devices.

Firebase Authentication is used to handle **user signups and logins**, offering secure and streamlined access. It validates credentials and manages user sessions efficiently, ensuring that only authenticated users can access personalized content. The app also leverages **Firebase Firestore** (or Realtime Database) to **store, retrieve, and sync habit data** in real time, enabling a smooth and consistent experience across devices.

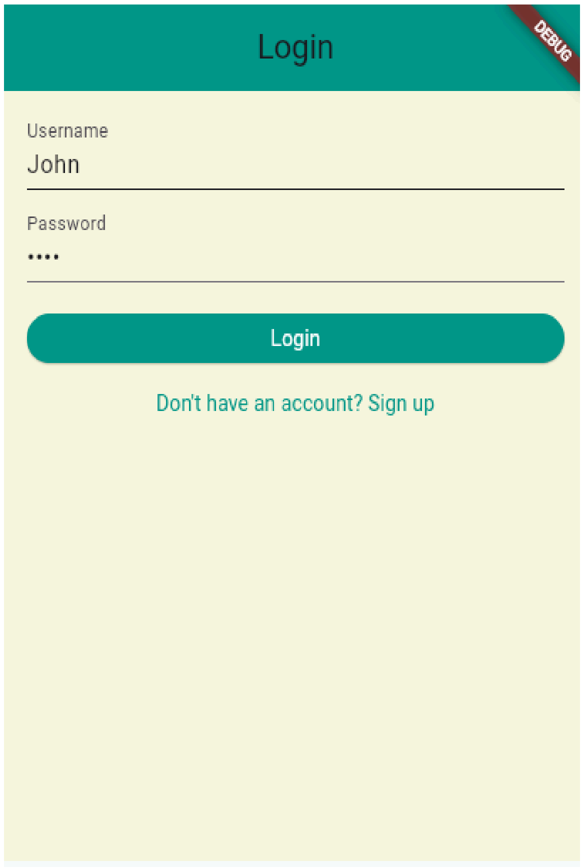
This integration not only enhances the security and reliability of the app but also supports **scalability**, allowing the app to grow with an increasing user base. By using Firebase, FitSync ensures real-time updates, secure data handling, and a unified backend infrastructure, laying a solid foundation for expanding features and maintaining a responsive and connected user experience.

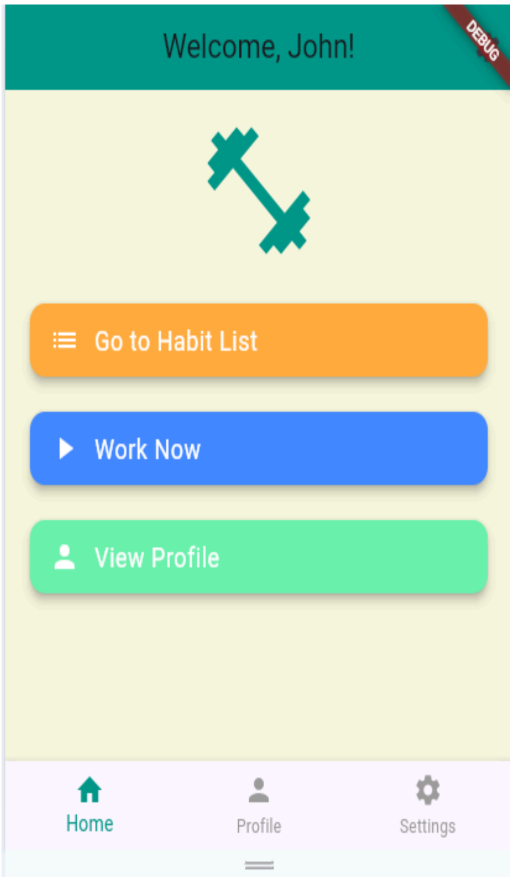
Output:



```
_id: ObjectId('67ecf4cdaa9ea7d41ba0e8e8')
username : "john_doe"
bookTitle : "The Great Gatsby"
issueDate : "2025-03-30"
returnDate : null
```

```
_id: ObjectId('67ecf4cdaa9ea7d41ba0e8e9')
username : "admin"
bookTitle : "1984"
issueDate : "2025-03-28"
returnDate : "2025-04-01"
```





Conclusion:In conclusion, setting up Firebase with Flutter in the FitSync app successfully enabled secure authentication and backend support for both Android and iOS platforms. With Firebase Authentication integrated, users can sign up and log in seamlessly, ensuring that their habit data is securely stored and managed. This integration establishes a reliable cloud-based infrastructure, making the app scalable, secure, and ready for real-world deployment.

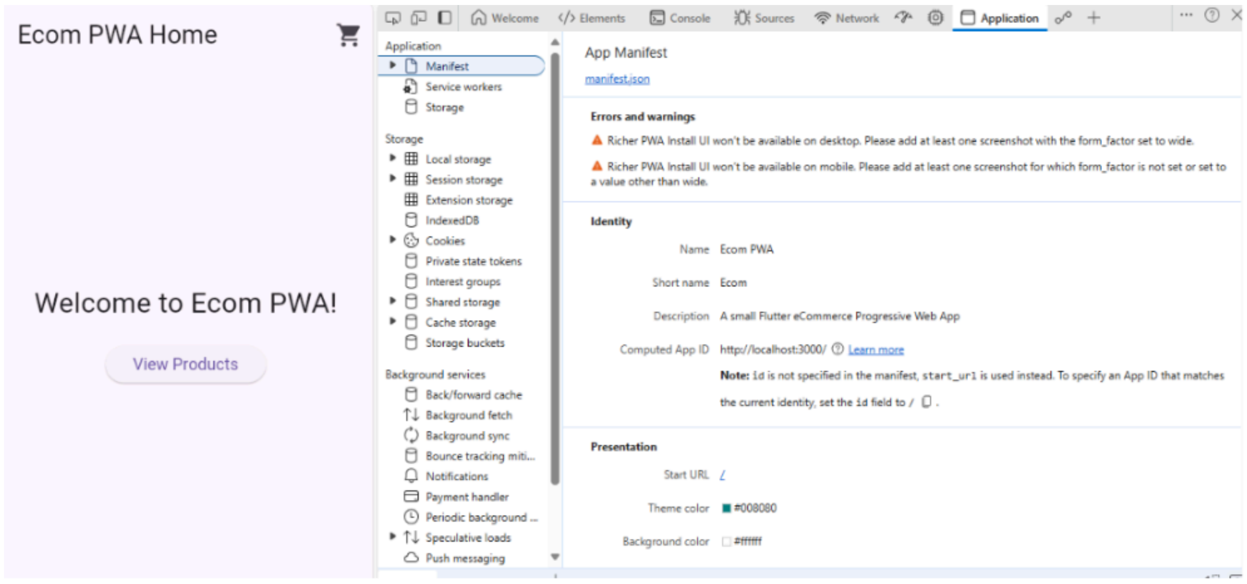
EXPERIMENT 7	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

Github link:https://github.com/dimpledalwani/flutter-ecom-pwa1.git

Theory:The **Web App Manifest** is a crucial component in turning an e-commerce website into a Progressive Web App (PWA). It is a JSON file that contains metadata about the app, such as the name, short name, start URL, display mode, icons, and theme colors. These details allow browsers to recognize the app as installable, enabling users to add it directly to their device's home screen. For an e-commerce PWA, this enhances the shopping experience by offering a more app-like interface and faster access. Properly configured, the manifest ensures the app opens in a standalone window and uses the defined branding elements, making it feel like a native app.

Output:



App Manifest

manifest.json

Errors and warnings

Richer PWA Install UI won't be available on desktop. Please add at least one screenshot with the form_factor set to wide.

Richer PWA Install UI won't be available on mobile. Please add at least one screenshot for which form_factor is not set or set to a value other than wide.

Identity

Name	Ecom PWA
Short name	Ecom
Description	A small Flutter eCommerce Progressive Web App
Computed App ID	http://localhost:3000/ Learn more
	Note: `id` is not specified in the manifest, `start_url` is used instead. To specify an App ID that matches the current identity, set the `id` field to `/`.

Presentation

Start URL	/
Theme color	#008080
Background color	#ffffff

Conclusion:In conclusion, writing the web app manifest with accurate and meaningful metadata is essential for enabling the “Add to Homescreen” feature in an e-commerce PWA. It not only improves user engagement but also provides a consistent and seamless experience across different devices. This simple file plays a significant role in enhancing the app’s visibility, usability, and overall performance as a reliable, installable web application.

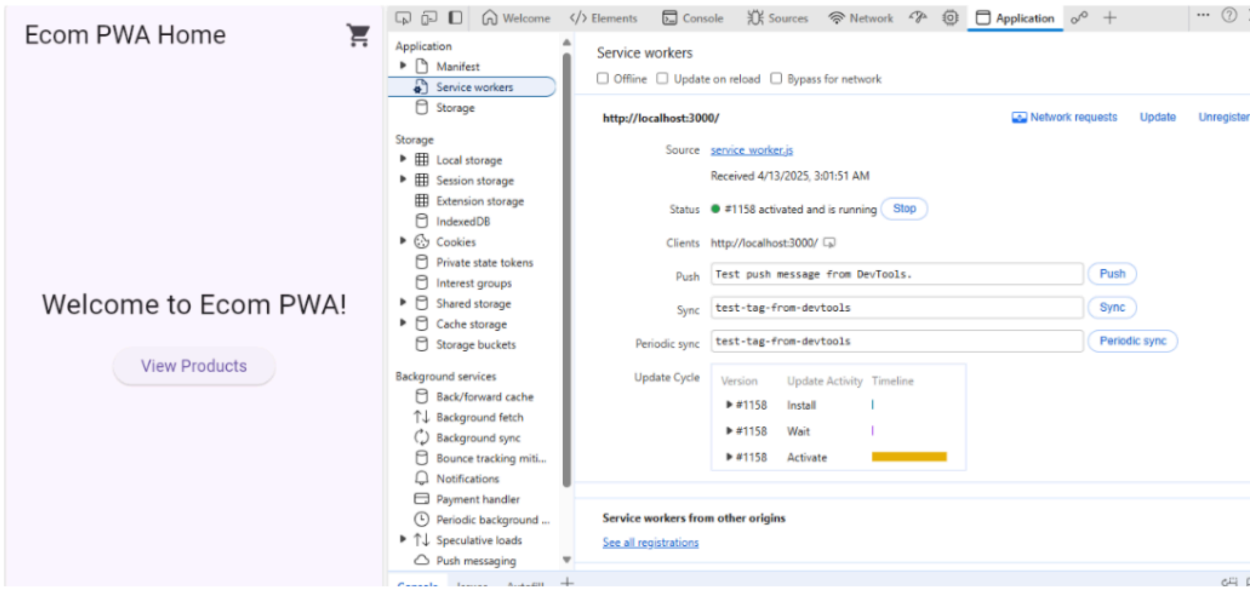
EXPERIMENT 8	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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.

Github link:https://github.com/dimpledalwani/flutter-ecom-pwa1.git

Theory:The **service worker** is a JavaScript file that runs in the background and plays a vital role in enabling offline capabilities, caching, and background processes in a **Progressive Web App (PWA)**. In an e-commerce PWA, coding and registering a service worker involves defining its lifecycle events—**install**, **activate**, and **fetch**—to handle asset caching and content delivery. The registration is done in the main JavaScript file using navigator.serviceWorker.register(). During the **install phase**, essential files are cached, and in the **activation phase**, old caches are cleared. This ensures that users can access the e-commerce site even without an internet connection, improving speed, reliability, and performance.

Output:



Service workers

☐ Offline ☐ Update on reload ☐ Bypass for network

http://localhost:3000/

Source service worker.js

Received 4/13/2025, 3:01:51 AM

Status ● #1158 activated and is running

Clients http://localhost:3000/

Push `Test push message from DevTools.`

Sync `test-tag-from-devtools`

Periodic sync `test-tag-from-devtools`

Update Cycle

Version	Update Activity	Timeline
▸ #1158	Install	
▸ #1158	Wait	
▸ #1158	Activate	

Service workers from other origins

See all registrations

Conclusion:In conclusion, successfully coding and registering a service worker, along with completing its install and activation phases, is a fundamental step in turning an e-commerce site into a PWA. It significantly enhances the app's functionality by enabling offline access, faster load times, and smoother user interactions. This strengthens user trust and retention, making the e-commerce platform more robust and user-friendly.

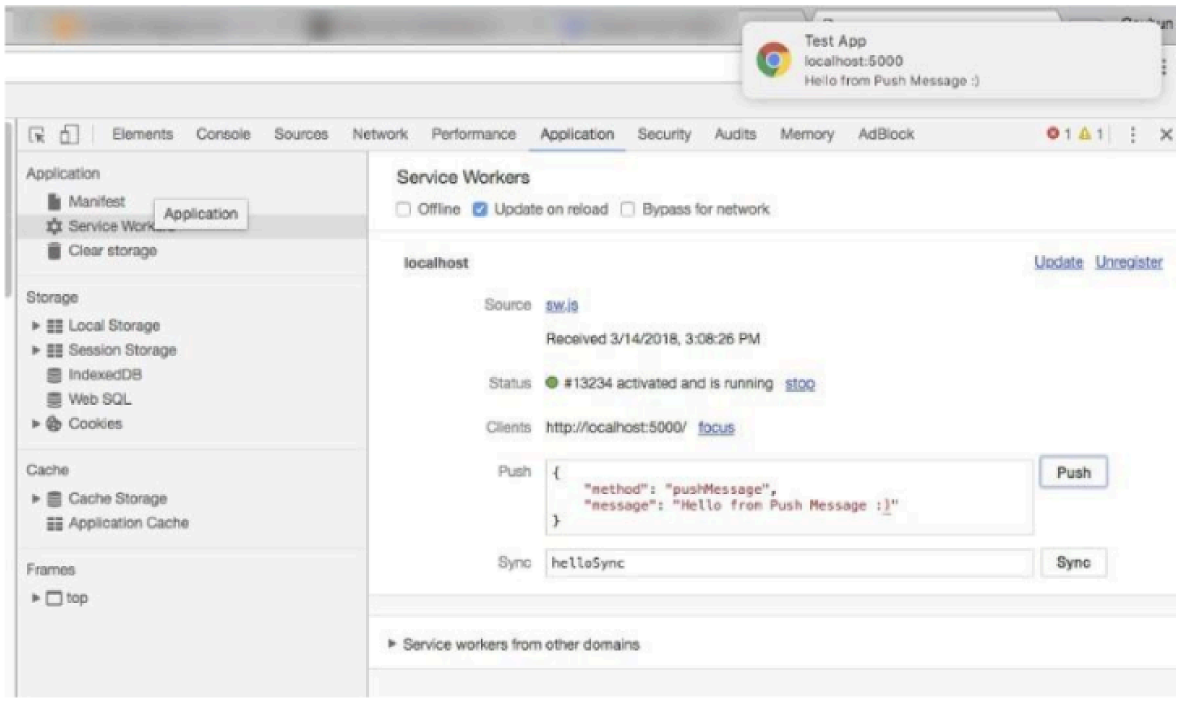
EXPERIMENT 9	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

Github link:https://github.com/dimpledalwani/flutter-ecom-pwa1.git

Theory:In an **E-commerce PWA**, implementing service worker events like **fetch**, **sync**, and **push** enhances user experience by enabling advanced background functionalities. The **fetch** event intercepts network requests and serves cached responses when offline, improving performance and reliability. The **sync** event allows background synchronization of data (e.g., submitting orders or saving cart items) when the device regains connectivity. The **push** event enables the app to receive and display notifications, such as order updates or promotional alerts, even when the app isn't open. These events are handled within the service worker script and contribute to making the e-commerce PWA more dynamic, responsive, and user-centric.

Output:



Test App
localhost:5000
Hello from Push Message :)

Service Workers

☐ Offline ☑ Update on reload ☐ Bypass for network

localhost

Source sw.js

Received 3/14/2018, 3:08:26 PM

Status ● #13234 activated and is running stop

Clients http://localhost:5000/ focus

Push
```
{
    "method": "pushMessage",
    "message": "Hello from Push Message :)"
}
```

Sync helloSync

Conclusion:In conclusion, implementing fetch, sync, and push events in the service worker is essential for delivering a seamless and engaging experience in an e-commerce PWA. These events ensure real-time updates, offline functionality, and personalized user interactions, ultimately making the platform more reliable and competitive in the digital marketplace.

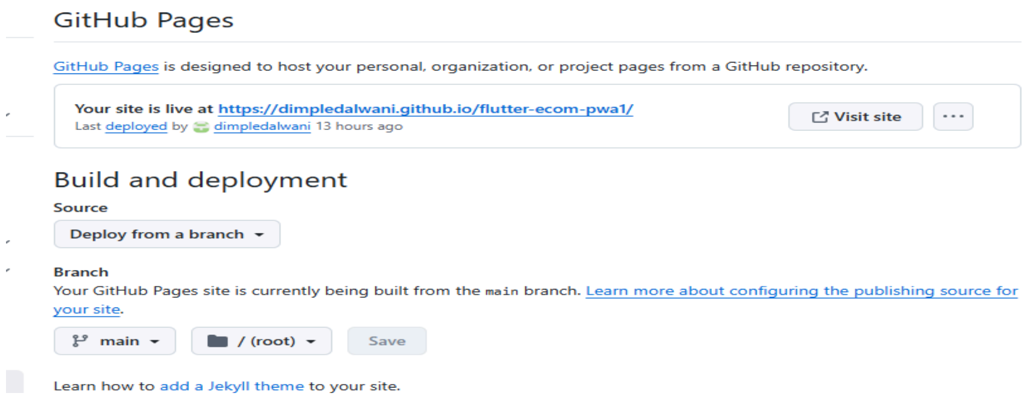
EXPERIMENT 10	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

Github link:https://github.com/dimpledalwani/flutter-ecom-pwa1.git

Theory:Deploying an **E-commerce PWA** to **GitHub Pages** involves hosting the web app using GitHub's static site hosting feature. The process includes building the project (especially if using frameworks like Flutter, React, or Angular), ensuring that all **PWA requirements** like the manifest file, service worker, and icons are properly configured, and pushing the final production files to a GitHub repository. GitHub Pages serves files from the `main` or `gh-pages` branch, so developers often use deployment tools or commands (like `npm run deploy`) to publish the build to the appropriate branch. Proper routing and `404.html` support must also be handled for single-page applications to function correctly.

Output:



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://dimpledalwani.github.io/flutter-ecom-pwa1/
Last deployed by dimpledalwani 13 hours ago

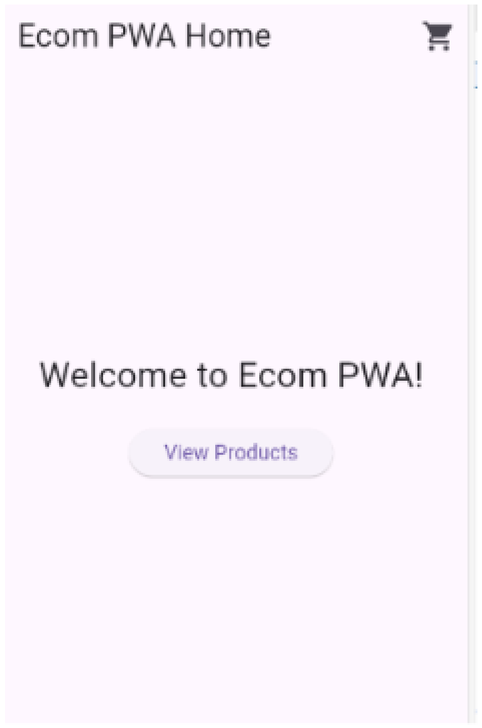
Build and deployment

Source

Branch

Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.

Learn how to add a Jekyll theme to your site.



Conclusion:In conclusion, deploying an E-commerce PWA to GitHub Pages is an efficient way to make the app publicly accessible and test its PWA capabilities. It provides a free, reliable hosting solution for static content while supporting core PWA features like offline access and “Add to Homescreen.” This deployment method is ideal for showcasing the app, gathering user feedback, and maintaining updates directly from a GitHub repository.

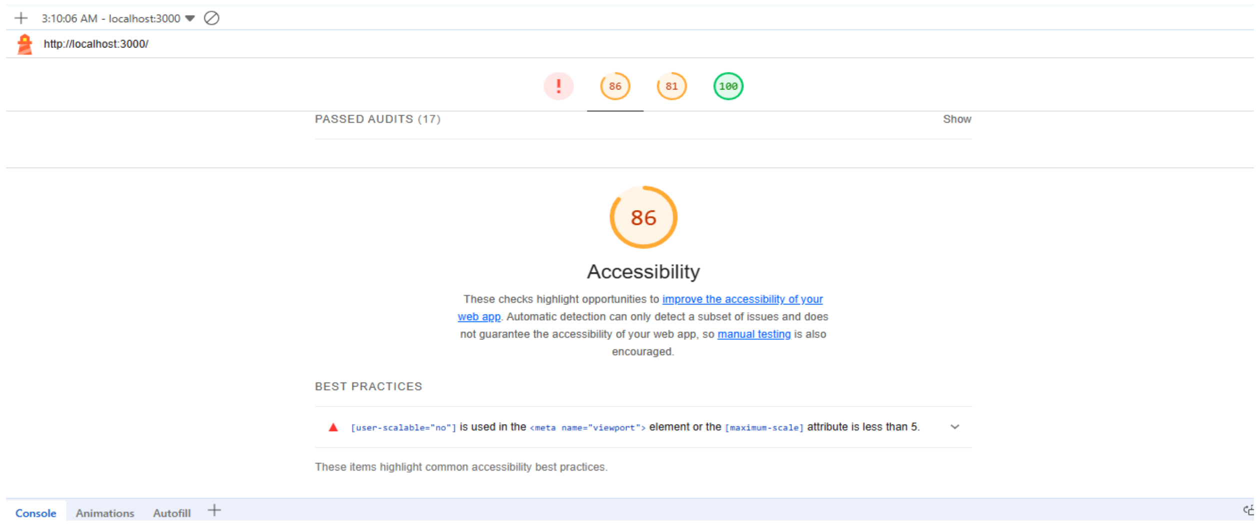
EXPERIMENT 11	
Name	DIMPLE DALWANI
Class_Roll no	D15C_8
DOP	
DOS	
Grade	
Sign	

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

Github link:https://github.com/dimpledalwani/flutter-ecom-pwa1.git

Theory:Using **Google Lighthouse** for PWA analysis is an essential step in testing and improving the performance and functionality of an **E-commerce PWA**. Lighthouse is a free, automated tool available in Chrome DevTools that audits various aspects of a web application, including performance, accessibility, best practices, SEO, and PWA compliance. When used for a PWA, it checks whether the app meets critical PWA requirements such as service worker registration, valid manifest file, offline capability, fast load time, and installability. The tool provides a detailed report with scores and actionable suggestions, helping developers identify issues and optimize the app for a better user experience.

Output:



Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

BEST PRACTICES

[user-scalable="no"] is used in the <meta name="viewport"> element or the [maximum-scale] attribute is less than 5.

These items highlight common accessibility best practices.

Conclusion:In conclusion, using the Google Lighthouse PWA analysis tool is a valuable practice for evaluating and enhancing the quality of an E-commerce PWA. It ensures that the app is not only functional but also reliable, fast, and user-friendly across different devices. This step helps developers align with modern web standards and deliver a high-quality shopping experience to users.