

Experiment No. 1

Aim:

To install and configure the Flutter Environment.

Theory:

The goal of this experiment was to install and set up the Flutter SDK along with the required tools to develop mobile applications using a single codebase. For our Attendance App, this setup was the foundation that enabled the smooth development of features across Android and iOS platforms.

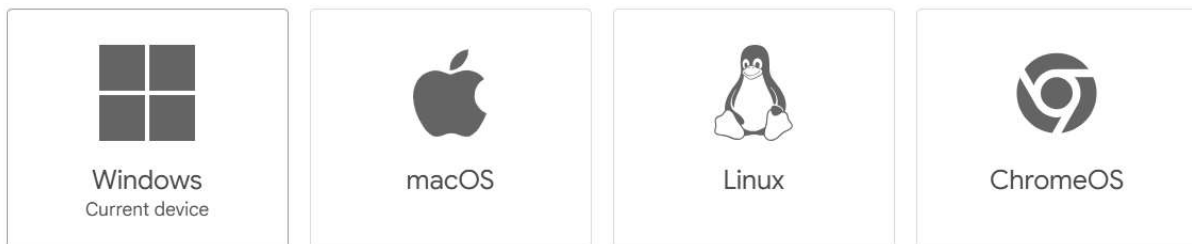
We began by downloading and installing the Flutter SDK, setting the environment path, and running the `flutter doctor` command to verify the setup. Visual Studio Code was chosen as the IDE, with necessary extensions such as Flutter and Dart installed. The Android emulator and a physical device were configured for testing the app during the development cycle.

This configuration allowed us to start building the core UI and logic for our app and ensured that our development workflow was stable and efficient.

Screenshot:

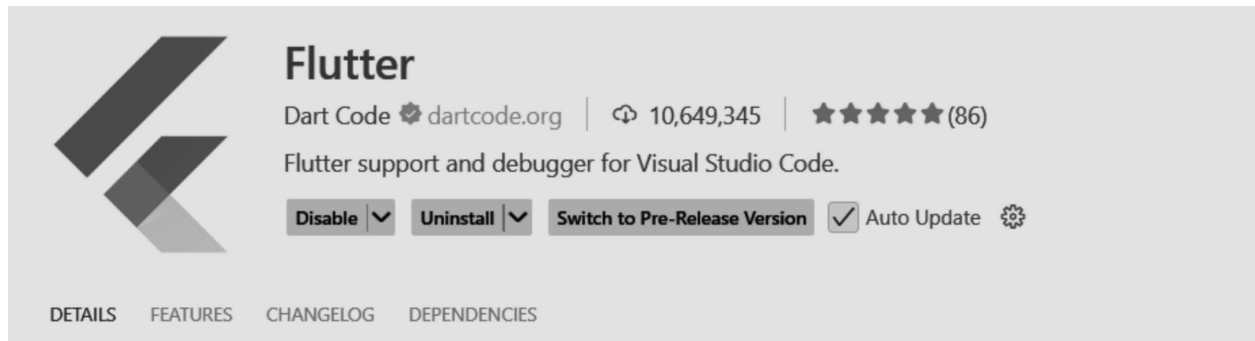
Choose your development platform to get started

[Get started](#) > [Install](#)



```
C:\Users\athar>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 3.29.0, on Microsoft Windows [Version 10.0.22621.4317], locale en-IN)
[✓] Windows Version (11 Home Single Language 64-bit, 22H2, 2009)
[✓] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop Windows apps (Visual Studio Build Tools 2019 16.11.35)
[✓] Android Studio (version 2024.2)
[✓] VS Code (version 1.99.0)
[✓] Connected device (3 available)
[✓] Network resources

• No issues found!
```



Conclusion:

Flutter environment was successfully installed and configured. It provided the necessary setup for beginning the development of our Manual Attendance App and ensured that the app could be built and tested across platforms seamlessly.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

Experiment No. 2

Aim:

To design Flutter UI by including common widgets.

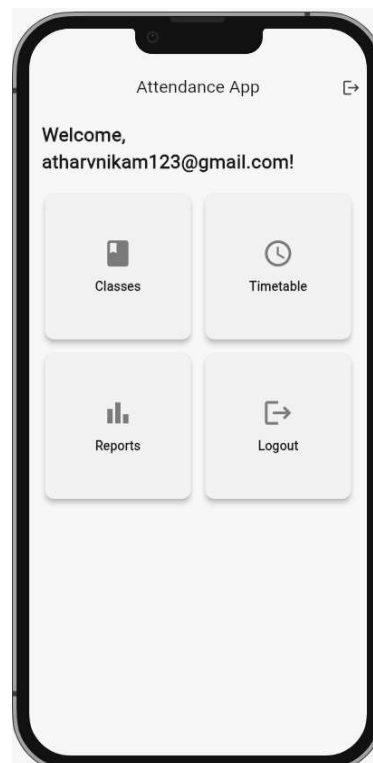
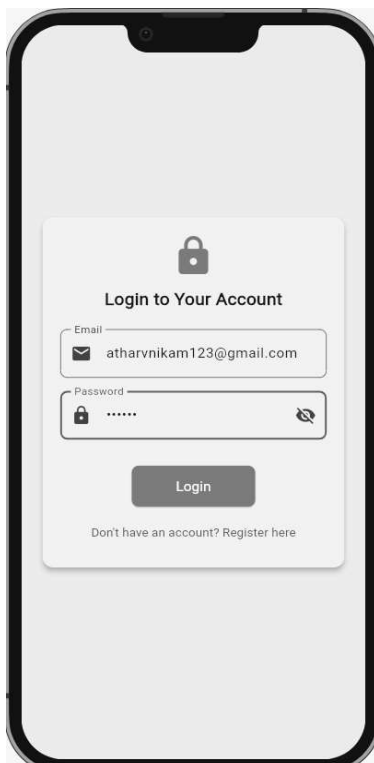
Theory:

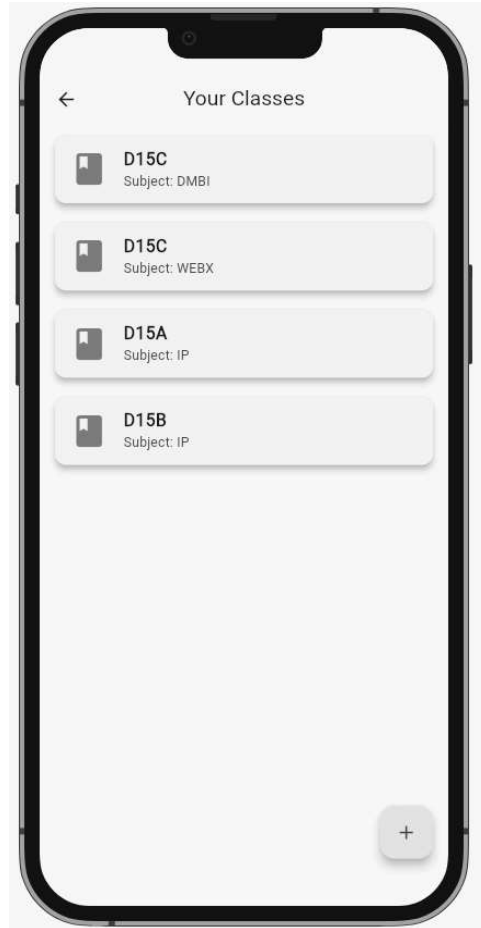
In this experiment, we focused on designing the user interface of our Attendance App using common Flutter widgets. Flutter provides a rich set of pre-designed widgets that make UI development fast and expressive.

We utilized widgets like Scaffold, AppBar, Text, TextField, ListView, Container, Card, and ElevatedButton to create different parts of the UI, such as the homepage, class list, student list, and attendance mark screen. These widgets allowed us to maintain a clean layout, responsive design, and consistent theme throughout the application.

Implementing these widgets made the app user-friendly and ensured that the navigation and functionality were clearly accessible to the teacher using the app.

Screenshot:





Conclusion:

Common Flutter widgets were effectively used to build a functional and attractive UI for the Manual Attendance App. These widgets enhanced the user experience and helped in implementing a structured and scalable app layout.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

Experiment No. 3

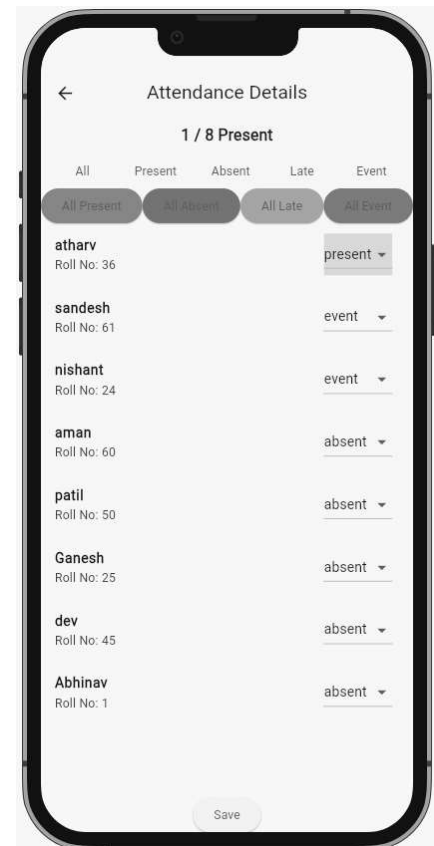
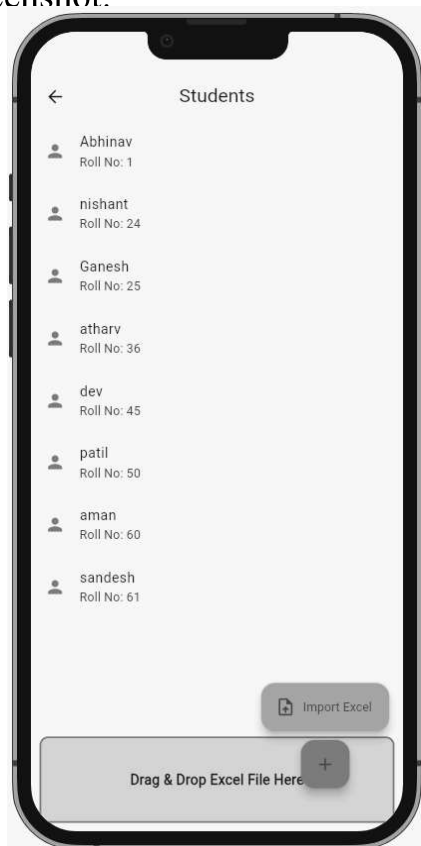
Aim: To include icons, images, and fonts in a Flutter app.

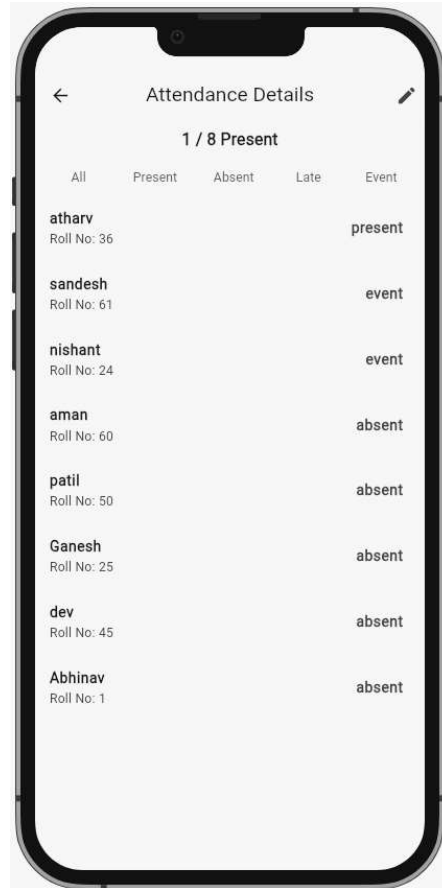
Theory: In this experiment, we explored how to enhance the visual appeal and user experience of our Attendance App by adding custom icons, images, and fonts. Flutter allows developers to easily include and manage these assets using the `pubspec.yaml` file.

We imported custom fonts to maintain a professional and clean typography across the app, especially for headings and section titles. Icons from the `Icons` class and third-party libraries like `font_awesome_flutter` were used to represent actions like attendance marking, settings, and user profiles. Relevant images such as app logo, splash screen graphics, and empty state illustrations were added to make the interface more engaging and user-friendly.

All assets were placed in the appropriate folders and referenced efficiently to keep the codebase clean. This helped in giving a consistent branding feel to the application and made the user interface more intuitive.

Screenshot:





Conclusion:

By integrating custom fonts, icons, and images, the Manual Attendance App achieved a more polished and professional look. These elements significantly improved the overall design and usability of the app. Additionally, using organized asset management ensured better scalability and maintainability of the project in the long run.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

Experiment No. 4

Aim:

To create an interactive Form using Form widget.

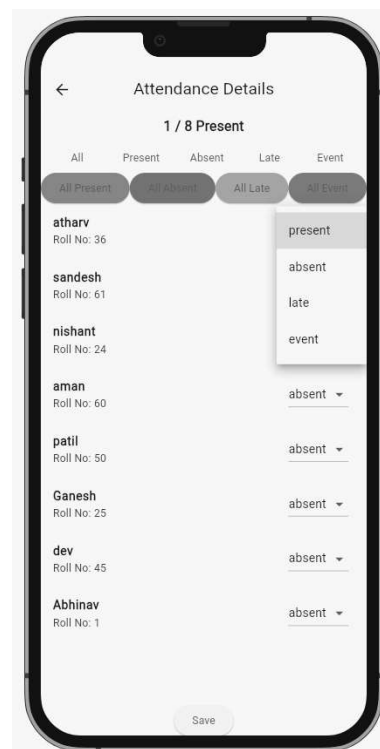
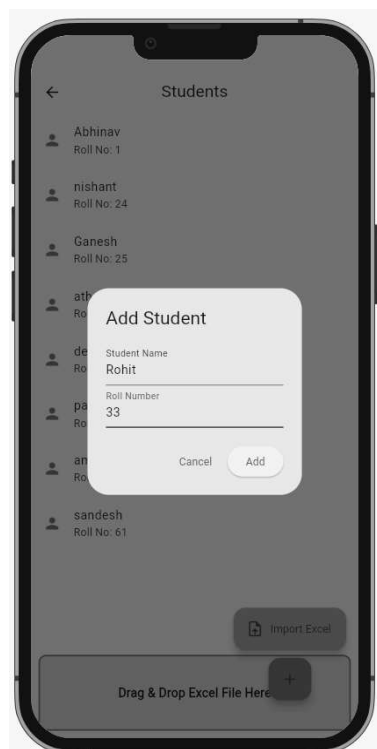
Theory:

In this experiment, we implemented interactive form elements in our Manual Attendance App to collect and validate user input. Flutter provides a powerful Form widget along with `TextFormField` and built-in validation support, which was leveraged to design a user-friendly and secure data input system.

In the app, the form was used in features like the Add Student screen, where the teacher inputs student details such as name and roll number. Each input field was wrapped in a Form widget, managed by a `GlobalKey`, and included validators to check for empty inputs and valid formats. Additionally, buttons were conditionally enabled only when the form was valid, improving overall data integrity.

We also added helpful hints, error messages, and UI feedback to guide users in real-time while entering data. This form design minimized errors and ensured clean, structured data entry throughout the app.

Screenshot:



Conclusion:

The use of Flutter's Form and TextFormField widgets allowed us to implement dynamic and validated user input features in our app. This significantly improved the app's reliability and provided a smooth experience for teachers when adding or managing student data. The form's structure ensured data consistency and prepared the app for future expansion, such as data storage and user authentication.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

Experiment No. 5

Aim:

To apply navigation, routing, and gestures in a Flutter App.

Theory:

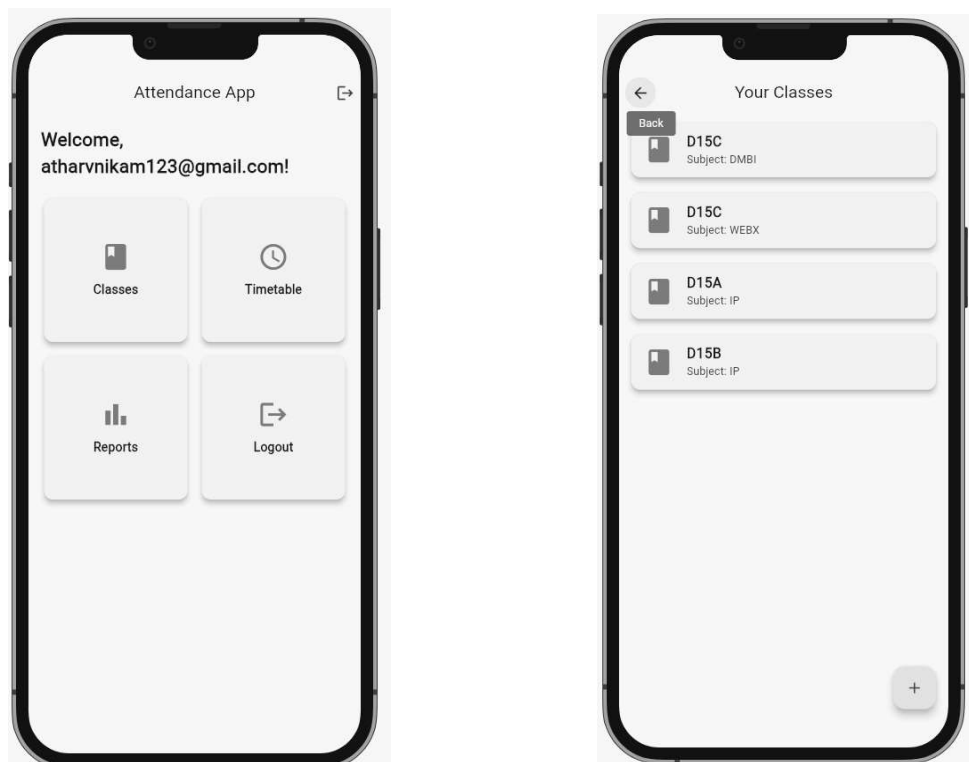
In this experiment, we implemented navigation, routing, and gesture handling in our Attendance App to provide a seamless and interactive user experience.

Navigation is a key part of any app's user flow, and Flutter offers multiple ways to move between screens, including named and anonymous routes.

We structured the app using named routes for better scalability and maintainability. Screens like Login, Dashboard, Class List, and Attendance Marking were connected using Flutter's `Navigator` and `MaterialPageRoute`. This allowed users to easily transition between functionalities with a smooth and intuitive flow.

Gesture detectors and built-in gesture widgets such as `InkWell` and `GestureDetector` were used to capture tap actions on UI components like cards and buttons. For example, tapping on a class card navigates to the student list, and tapping on a student toggles their attendance status. These interactions made the app more dynamic and reduced the need for additional UI clutter.

Screenshot:



Conclusion:

The implementation of navigation and gestures played a crucial role in enhancing the usability of the Manual Attendance App. It allowed users to quickly access different parts of the app while maintaining a clean and responsive UI. The routing structure also made the codebase more organized and future-proof for adding more features. Gesture handling added a natural and interactive feel to the app, making it intuitive even for non-technical users.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

Experiment No. 6

Aim:

How to set up Firebase with Flutter for iOS and Android Apps.

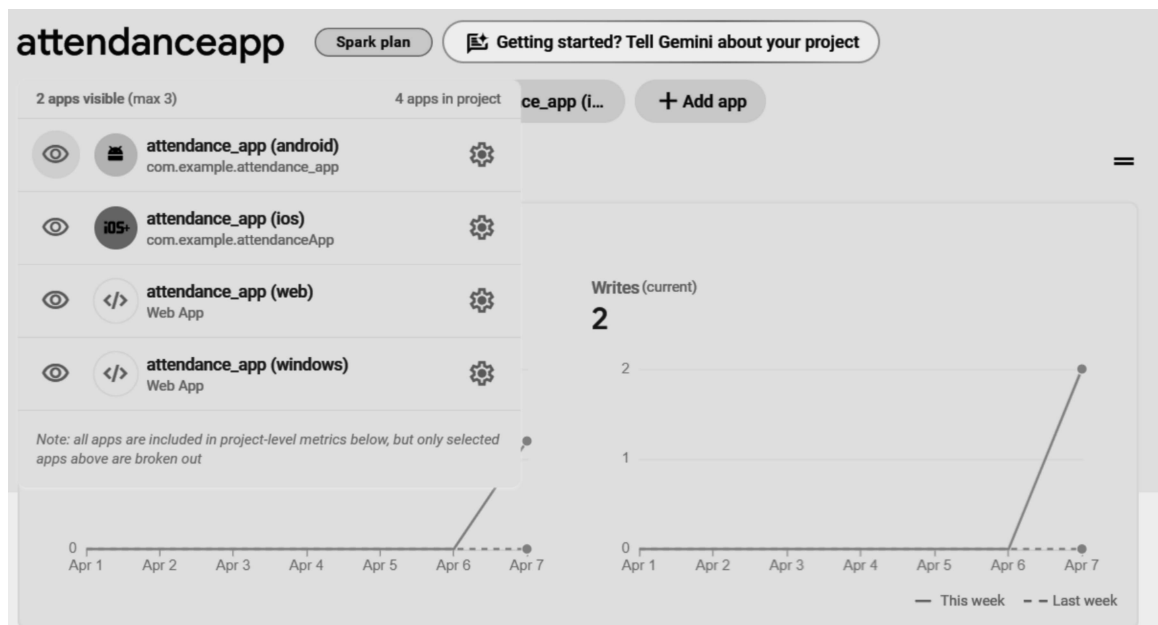
Theory:

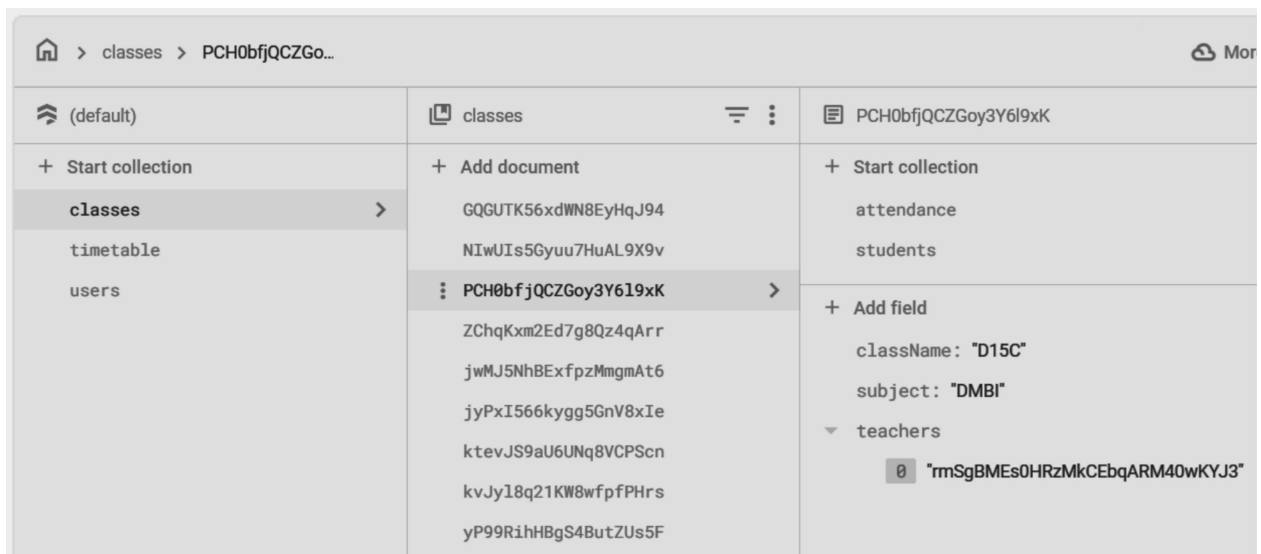
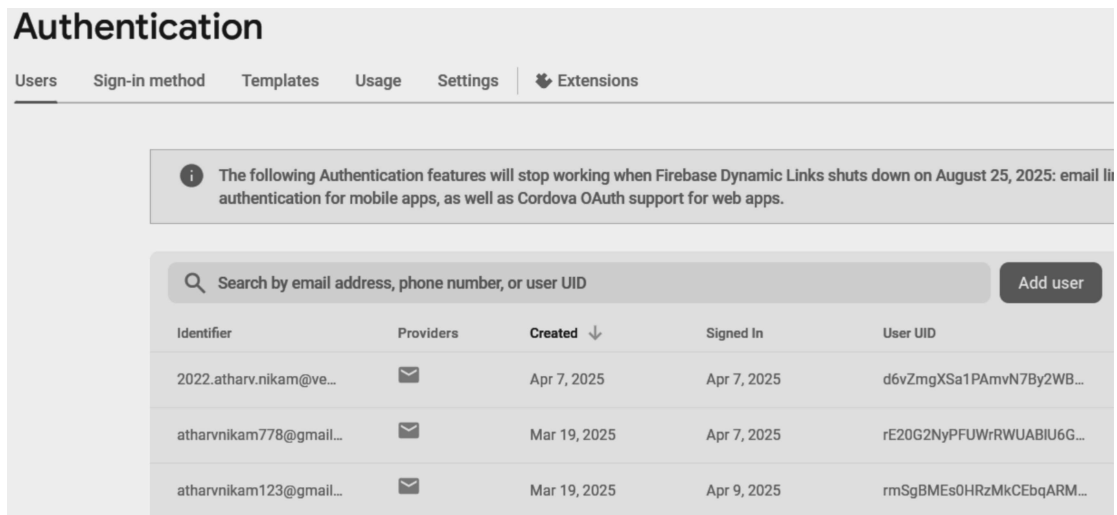
This experiment focused on integrating Firebase into our Manual Attendance App to enable backend services like authentication, data storage, and real-time updates. Firebase is a powerful backend-as-a-service (BaaS) that works seamlessly with Flutter, making it easier to implement essential features without building a custom backend.

We started by creating a Firebase project and registering both Android and iOS versions of our app. The `google-services.json` and `GoogleService-Info.plist` files were added to their respective platforms. Then, we configured Firebase in Flutter using dependencies like `firebase_core`, `cloud_firestore`, and `firebase_auth`.

In our app, Firebase was used to securely store class and attendance data. Teachers could log in, create classes, and mark attendance, with all records being stored in Firestore in real-time. This ensured that data remained persistent, accessible, and scalable. Firebase also opens the door for future features like analytics, push notifications, and user-specific data sync.

Screenshot:





Conclusion:

Setting up Firebase with Flutter significantly enhanced the functionality of our Manual Attendance App by adding secure data storage and real-time updates. The integration was smooth and scalable, allowing us to focus more on building features rather than managing servers. Firebase will be a crucial component as the app evolves, especially for handling authentication, cloud data, and analytics efficiently.

GitHub Link:

<https://github.com/atharvnikam38/attendanceapp>

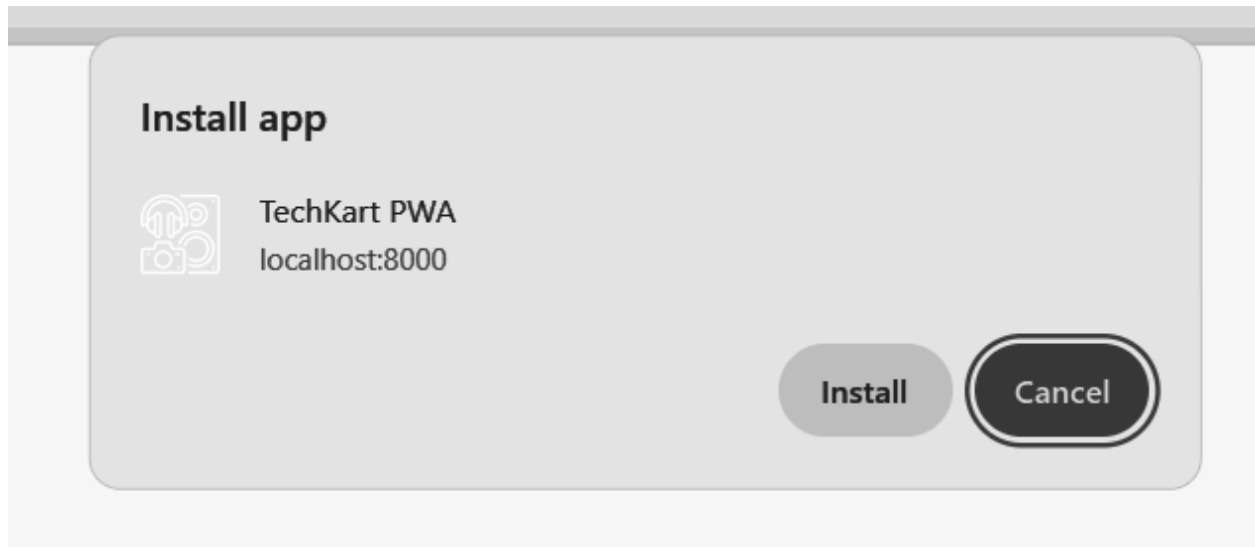
EXPERIMENT NO : 07**MAD and PWA Lab****1. Aim:**

To learn about Progressive Web Apps (PWAs) and their difference from standard websites.

2. Description:

Progressive Web Apps (PWAs) are web applications that function like native apps, offering features like offline access, push notifications, and home screen installation. They bridge the gap between websites and native apps by using standard web technologies (HTML, CSS, JavaScript) to deliver an enhanced user experience. Key differences include direct access (like native apps), increased engagement options, offline functionality, and improved performance metrics compared to traditional websites.

```
{ } manifest.json X
{ } manifest.json > ...
1  {
2    "name": "TechKart PWA",
3    "short_name": "TechKart",
4    "start_url": ".",
5    "display": "standalone",
6    "background_color": "#f4f4f4",
7    "theme_color": "#0a192f",
8    "description": "Buy gadgets & accessories online at TechKart",
9    "icons": [
10     {
11       "src": "icon-192.png",
12       "sizes": "192x192",
13       "type": "image/png"
14     },
15     {
16       "src": "icon-512.png",
17       "sizes": "512x512",
18       "type": "image/png"
19     }
20   ]
21 }
```



3. Conclusion:

PWAs offer a compelling blend of web and native capabilities, providing advantages like broader reach, lower development costs, and enhanced user engagement.

4. GitHub Link:

<https://github.com/atharvnikam38/techkart-pwa>

EXPERIMENT NO : 08**MAD and PWA Lab****1. 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.

2. Description:

This experiment focuses on Service Workers, which are essential for enabling advanced PWA features. A Service Worker is a JavaScript script that runs in the background of a web browser, independent of the web page. It acts as a programmable network proxy, allowing control over network requests, caching, and other background tasks.

- Key Concepts:
 - Service Worker: A background script that handles network requests, caching, and push notifications.
 - Registration: The process of telling the browser where the Service Worker script is located.
 - Installation: The event when the browser installs the Service Worker, allowing for caching of assets.
 - Activation: The stage where the Service Worker becomes active and takes control of the pages within its scope.
 - Caching: Storing assets (e.g., HTML, CSS, images) to provide offline access and improve performance.
- Service Worker Lifecycle:
 - Registration: The Service Worker is registered with the browser.
 - Installation: The `install` event is fired, and assets can be cached.
 - Activation: The `activate` event is fired, and the Service Worker takes control.
 - Fetch: Intercepts network requests.

Service workers

☐ Offline ☐ Update on reload ☐ Bypass for network

http://localhost:8000/

Source [sw.js](#)

Received 4/9/2025, 9:46:13 AM

Status ● #383 activated and is running Stop

Clients [http://localhost:8000/](#)

Push Push

Sync Sync

Periodic sync Periodic sync

Update Cycle

Version	Update Activity	Timeline
▶ #383	Install	
▶ #383	Wait	
▶ #383	Activate	<div></div>

Service workers from other origins

3. Conclusion:

This experiment demonstrates the process of coding, registering, installing, and activating a service worker. Service workers are crucial for PWAs, enabling offline functionality, caching, and enhanced performance.

4. GitHub Link:

<https://github.com/atharvnikam38/techkart-pwa>

EXPERIMENT NO : 09**MAD and PWA Lab****1. Aim:**

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

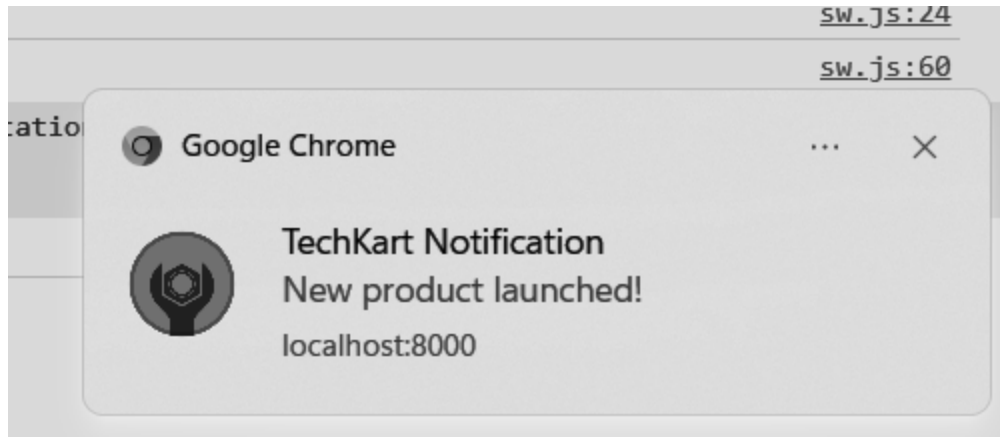
2. Description:

This experiment focuses on implementing key Service Worker events that enhance PWA functionality: fetch, sync, and push. These events enable PWAs to manage network requests, handle background data synchronization, and deliver push notifications.

- Key Service Worker Events:
 - Fetch Event: Allows the Service Worker to intercept and handle network requests. This is crucial for implementing caching strategies (e.g., "cache first," "network first") and providing offline functionality.
 - Sync Event: Enables background data synchronization when the network connection is available. This allows PWAs to defer actions until a stable connection is present.
 - Push Event: Handles push notifications received from a server, allowing PWAs to deliver timely updates and engage users.

The screenshot displays a web application interface for managing service worker events. At the top, the status is indicated as '#385 activated and is running' with a 'Stop' button. Below this, the 'Clients' section shows 'http://localhost:8000/' with a refresh icon. The 'Push' section contains a text input with the JSON object '{ "message": "New product launched!" }' and a 'Push' button. The 'Sync' section has a text input with 'test-tag-from-devtools' and a 'Sync' button. The 'Periodic sync' section also has a text input with 'test-tag-from-devtools' and a 'Periodic sync' button.

Status	● #385 activated and is running	Stop
Clients	http://localhost:8000/	
Push	<input type="text" value='{ "message": "New product launched!" }'/>	Push
Sync	<input type="text" value="test-tag-from-devtools"/>	Sync
Periodic sync	<input type="text" value="test-tag-from-devtools"/>	Periodic sync



```
> navigator.serviceWorker.ready.then(sw => {  
  sw.sync.register('sync-message');  
});  
◀ ▼ Promise {<fulfilled>: undefined} i  
  ▶ [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
    [[PromiseResult]]: undefined  
[SW] Background sync triggered!
```

3. Conclusion:

This experiment demonstrates the implementation of fetch, sync, and push events in a Service Worker. These events are essential for creating robust, offline-capable, and engaging PWAs.

4. GitHub Link:

<https://github.com/atharvnikam38/techkart-pwa>

EXPERIMENT NO. 10

MAD and PWA Lab

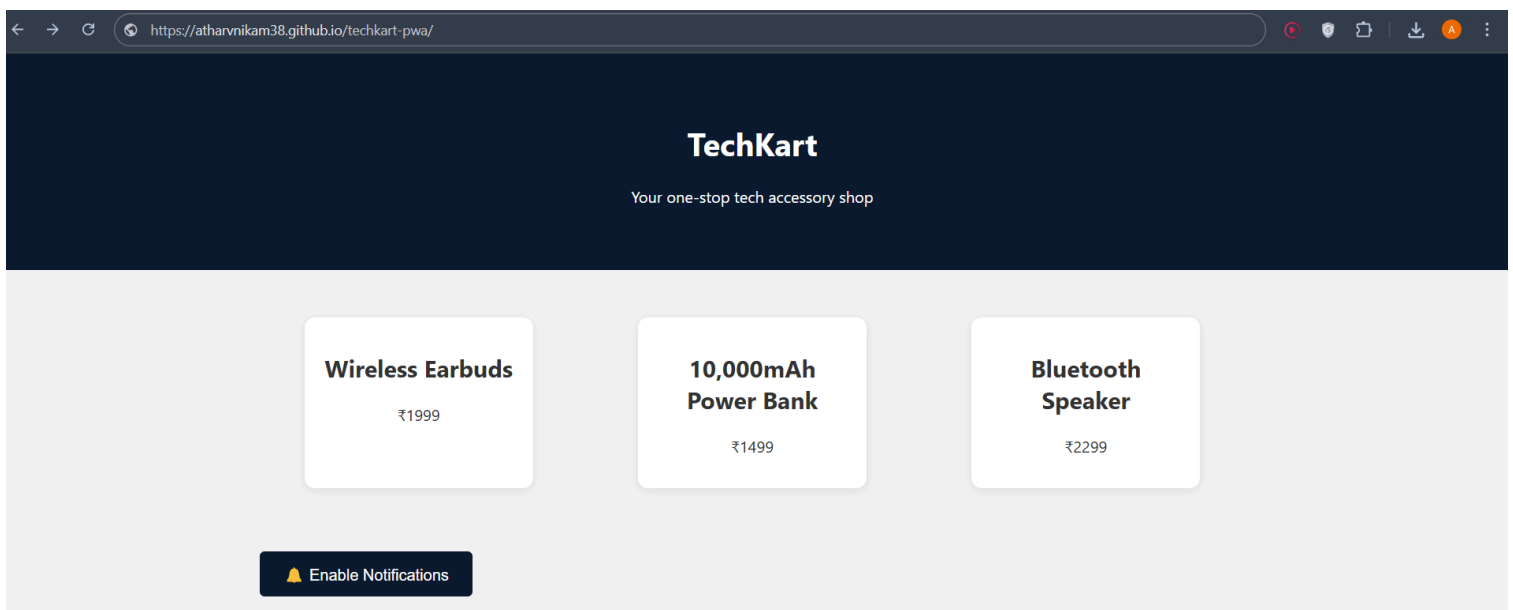
1. Aim:

To study and implement deployment of an E-commerce PWA to GitHub Pages.

2. Description:

This experiment focuses on deploying an E-commerce Progressive Web App (PWA) to GitHub Pages. GitHub Pages is a service that hosts web pages directly from a GitHub repository, making it a convenient option for deploying static websites and PWAs.

- **GitHub Pages:**
 - Public web pages hosted directly from a GitHub repository.
 - Easy and free to use for static content.
 - Supports custom domains.
- **Deployment Process:**
 - The process involves pushing the PWA's static files (HTML, CSS, JavaScript) to a specific branch (usually `gh-pages`) in the GitHub repository.
 - GitHub Pages then serves these files as a website.



3. Conclusion:

This experiment demonstrates the process of deploying an E-commerce PWA to GitHub Pages. GitHub Pages provides a simple and efficient way to host static websites and PWAs directly from a GitHub repository.

4. Github Link :

<https://atharvnikam38.github.io/techkart-pwa/>

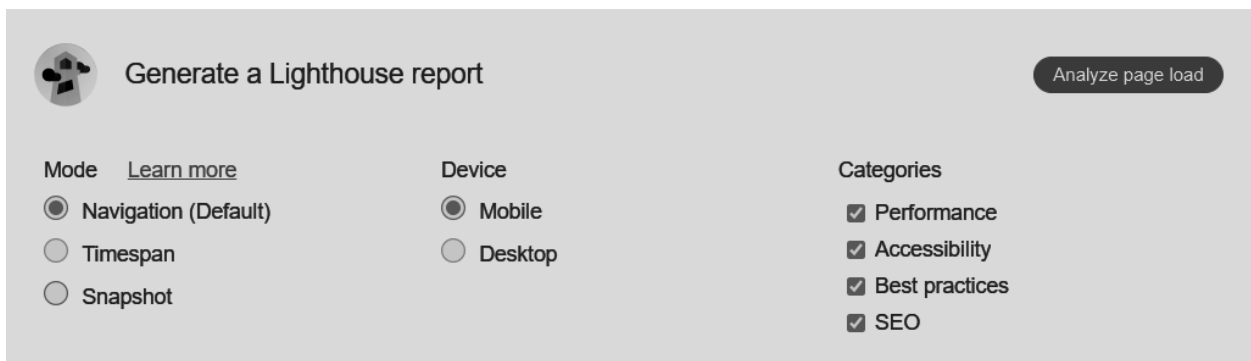
EXPERIMENT NO. 11**MAD and PWA Lab****1. Aim:**

To use Google Lighthouse PWA Analysis Tool to test the PWA functioning.

2. Description:

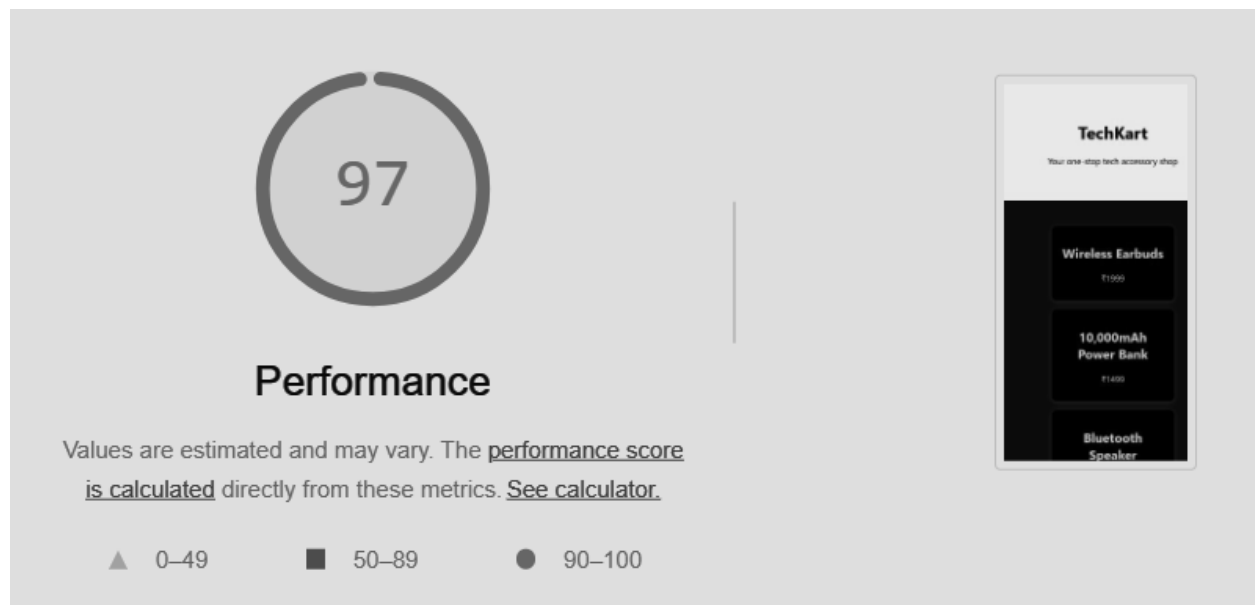
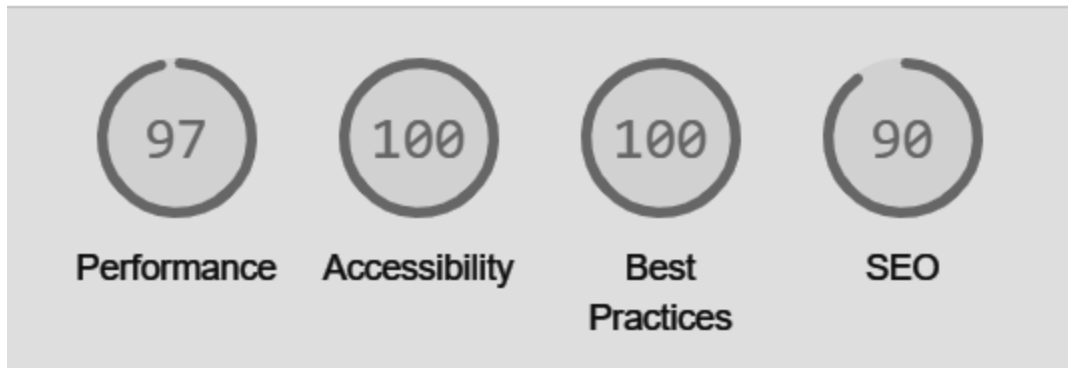
This experiment utilizes Google Lighthouse, an open-source, automated tool, to evaluate and improve the quality of web pages, including Progressive Web Apps (PWAs). Lighthouse audits various aspects of a web page, providing reports on performance, accessibility, PWA compliance, SEO, and more. It can be run within Chrome DevTools, from the command line, or as a Node module. Lighthouse provides a comprehensive analysis of a site's quality and offers recommendations for improvement.

- Key Features and Metrics:
 - Performance: Evaluates page load speed and provides metrics to optimize site performance.
 - Accessibility: Identifies opportunities to improve the usability of the site for all users.
 - Best Practices: Offers recommendations to enhance both performance and user experience.
 - SEO: Analyzes the page's optimization for search engine rankings.
 - Progressive Web App: Audits the page for PWA compliance, ensuring it meets the criteria for installability and native app-like functionality.



The screenshot shows the Google Lighthouse interface. At the top, there is a button labeled "Generate a Lighthouse report" with a Lighthouse icon to its left. To the right of this button is a button labeled "Analyze page load". Below these buttons, there are three sections of configuration options:

- Mode:** Includes a link "Learn more" and three radio button options: "Navigation (Default)" (selected), "Timespan", and "Snapshot".
- Device:** Includes two radio button options: "Mobile" (selected) and "Desktop".
- Categories:** Includes four checked checkboxes: "Performance", "Accessibility", "Best practices", and "SEO".



3. Conclusion:

This experiment demonstrates the use of Google Lighthouse as a valuable tool for testing and improving PWA functioning. Lighthouse provides detailed reports and actionable insights, enabling developers to optimize their PWAs for performance, accessibility, and overall quality.

4. GitHub Link:

<https://github.com/atharvnikam38/techkart-pwa>