

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 1

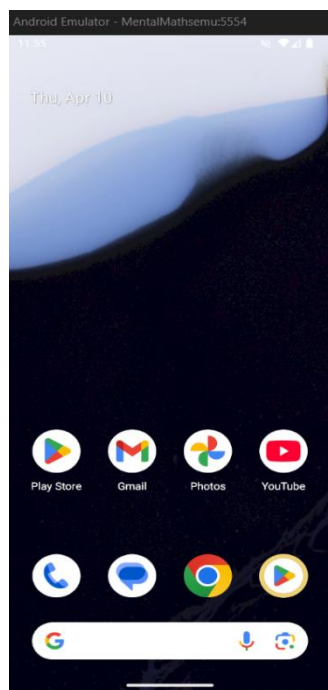
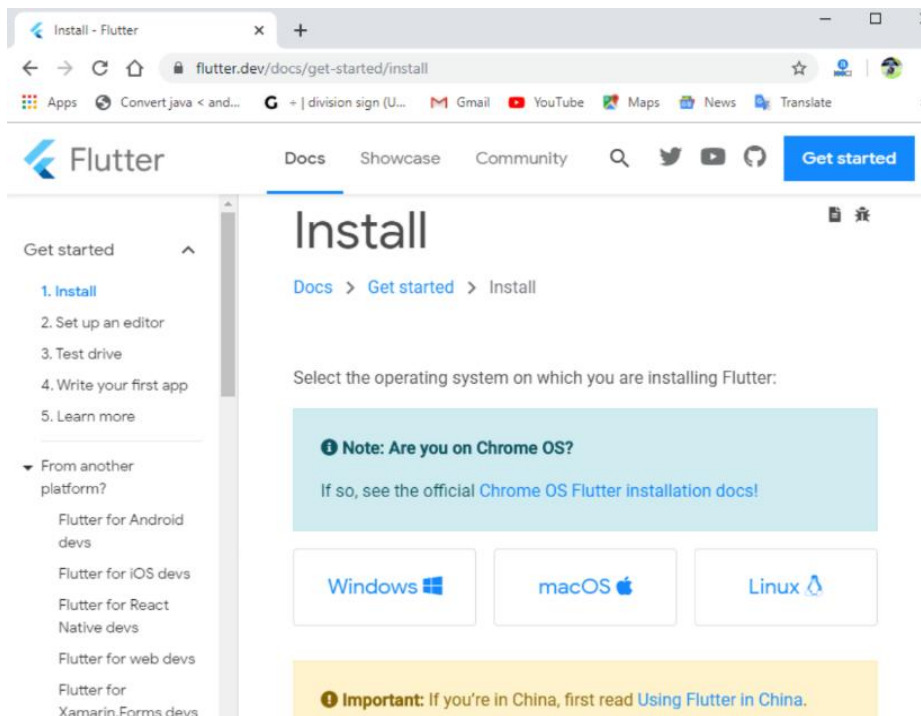
**Aim:** Installation and Configuration of Flutter environment.

### Theory:

- Flutter is an open-source UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase.
- To install and configure the Flutter development environment on a local system. This includes setting up Flutter SDK, Android Studio, and necessary plugins.
- It enables developers to build and run Flutter applications on emulators or physical devices.

### Output:

The screenshot shows the Flutter Docs website. The header includes the Flutter logo, 'Docs', and navigation links: Homepage, Community, Packages, API reference, and a search icon. A left sidebar lists navigation options under 'Get started' (Set up Flutter, Choose a platform, On Windows, On macOS, On Linux, On ChromeOS, Learn Flutter, Stay up to date, App solutions) and 'User interface' (Introduction, Widget catalog, Layout, Adaptive & responsive design, Design & theming, Interactivity, Assets & media). The main content area is titled 'Choose your development platform to get started' with a breadcrumb 'Get started > Install'. Below this are four platform cards: Windows (Current device), macOS, Linux, and ChromeOS. A blue box titled 'Developing in China' contains text about using Flutter in China. At the bottom, there is a feedback section 'Was this page's content helpful?' with thumbs up/down icons and a footer note: 'Unless stated otherwise, the documentation on this site reflects the latest stable version of Flutter. Page last updated on 2025-03-12. View source or report an issue.'



### Conclusion:

The Flutter environment setup, including the SDK, Android Studio, and plugins, enables developers to build and run apps smoothly on emulators or devices, forming the foundation for efficient Flutter development.

Github link: [https://github.com/w4lyf/MPL\\_LAB\\_01](https://github.com/w4lyf/MPL_LAB_01)

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 2

**Aim:** To design Flutter UI by including common widgets.

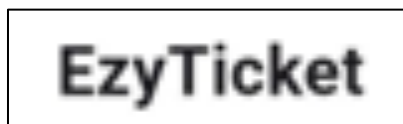
### Theory:

In Flutter, everything is a widget — from layout structures to UI elements. In this experiment, we created a simple user interface using common Flutter widgets like Text, Image, Container, Row, and Column.

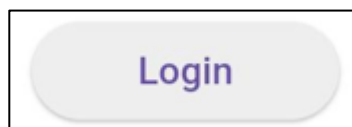
It helped us understand how widgets are the building blocks of Flutter apps and how they can be combined to design responsive UIs.

### Output:

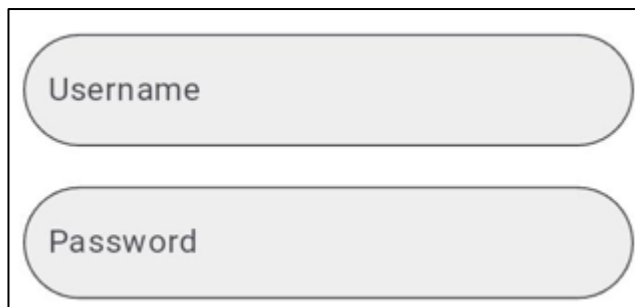
#### Text:

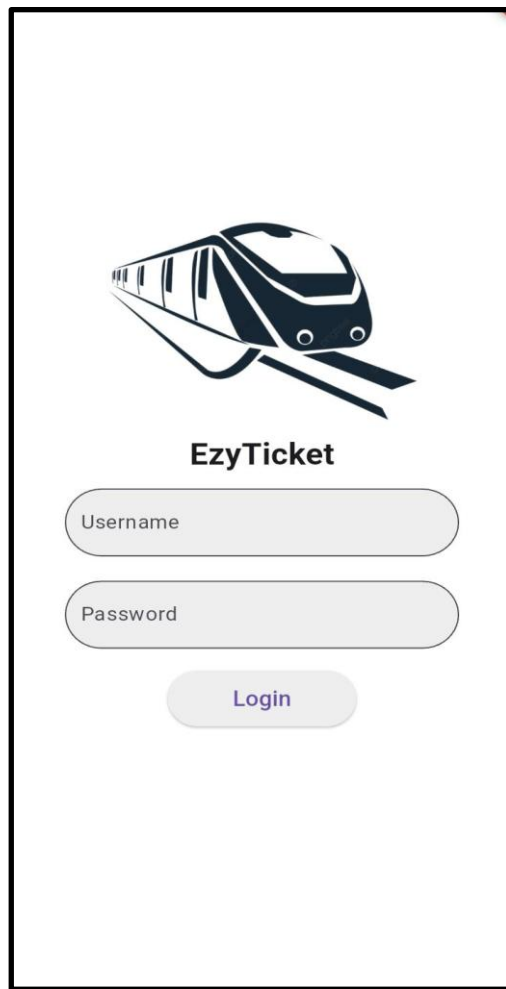
A screenshot of a text widget displaying the word "EzyTicket" in a bold, black, sans-serif font, centered within a white rectangular box with a thin black border.

#### Elevated Button:

A screenshot of an ElevatedButton widget with the text "Login" in a purple font, centered within a light gray rounded rectangle with a thin black border.

#### TextField:

A screenshot of two TextField widgets stacked vertically. The top widget contains the text "Username" and the bottom widget contains the text "Password". Both are light gray rounded rectangles with thin black borders.



### **Conclusion:**

In this experiment, we successfully designed a Flutter user interface by incorporating commonly used widgets such as Text, Image, Row, Column, Container, ElevatedButton, and others.

This hands-on experience provided a deeper understanding of how Flutter widgets work together to build responsive and visually appealing UIs. Mastering these basic widgets is essential for creating complex and interactive mobile applications using Flutter.

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

### Experiment 3

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

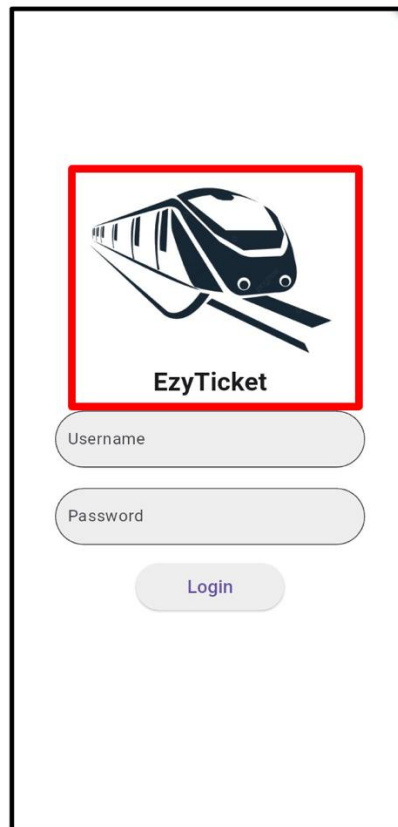
#### Theory:

Flutter allows easy customization of UI using various assets such as icons, images, and fonts. In this experiment, we learned how to enhance the visual appeal of a Flutter app by adding custom icons, images, and fonts. This helps in creating a more engaging and personalized user interface that aligns with the app's branding and design goals.

#### Output:

```
assets:
  - assets/INDIAN RAILWAY STATION LIST.csv
  - assets/stations.json
  - assets/railway_logo_1.png
  - assets/railway_map.png
  - assets/station_coordinates.json
  - assets/train_info.csv
```

```
fonts:
  - family: Schyler
    fonts:
      - asset: fonts/Schyler-Regular.ttf
      - asset: fonts/Schyler-Italic.ttf
        style: italic
  - family: Trajan Pro
    fonts:
      - asset: fonts/TrajanPro.ttf
      - asset: fonts/TrajanPro_Bold.ttf
        weight: 700
```



### **Conclusion:**

By successfully including icons, images, and custom fonts in the Flutter app, we gained practical knowledge of enhancing the app's visual design and user experience. This experiment highlighted the importance of media and typography in making apps more attractive and user-friendly.

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 4

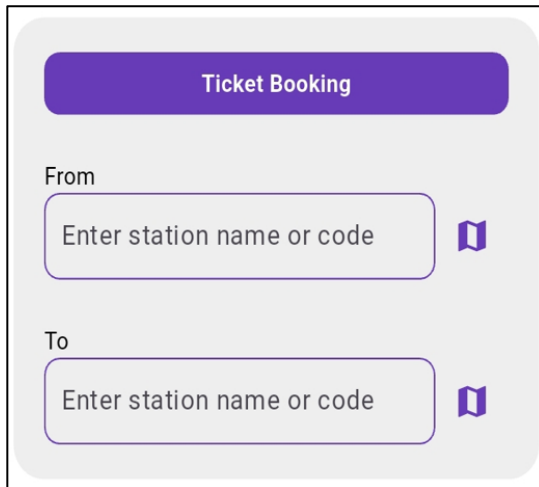
**Aim:** To create an interactive Form using form widget

### Theory:

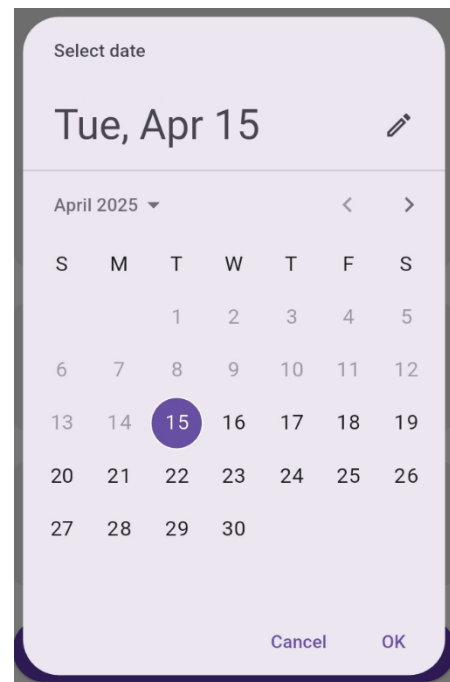
Flutter provides the **Form** widget to group and manage multiple form fields. It helps in validating and saving user input efficiently. In this experiment, we created an interactive form in Flutter using Form, TextFormField, and ElevatedButton widgets. It helped us understand how to collect, validate, and manage user input effectively within a Flutter app.

Forms are commonly used in login screens, registration, or any data-entry interface.

### Output:



The screenshot shows a mobile app interface for "Ticket Booking". At the top, there is a purple header bar with the text "Ticket Booking". Below the header, there are two input fields. The first field is labeled "From" and contains the placeholder text "Enter station name or code". The second field is labeled "To" and also contains the placeholder text "Enter station name or code". To the right of each input field is a small purple icon resembling a book or a document.



The screenshot shows a date selection interface. At the top, it says "Select date". Below this, the selected date is displayed as "Tue, Apr 15" with a small edit icon to its right. Underneath, there is a calendar grid for "April 2025". The days of the week are listed as S, M, T, W, T, F, S. The date "15" is highlighted with a purple circle. At the bottom of the calendar, there are two buttons: "Cancel" and "OK".

The screenshot shows a mobile application interface for ticket booking. At the top, the status bar displays the time 8:27, signal strength, LTE connectivity, and 42% battery. The app's header includes a welcome message "WELCOME BACK, ABHINAVS" and a user profile icon. A red banner in the top right corner reads "New Version 1.0.0". The main content area features a "Ticket Booking" section with two input fields for "From" and "To", each with a placeholder "Enter station name or code" and a magnifying glass icon. Below these are two dropdown menus for "Class" (set to 2A) and "Quota" (set to GN). A "Select Date" section shows a date picker with "2025-04-06" selected. A large purple "Search" button is positioned below the date selector. At the bottom of the form is a "Calibrate Map" button. The bottom navigation bar contains three icons: "Train" (a train icon), "Map" (a globe icon), and "Settings" (a gear icon).

## Conclusion:

By building an interactive form, we learned how to handle user input, perform validations, and manage form state in Flutter. This experiment is essential for developing apps that require user data input, such as login, registration, or feedback forms.



Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 5

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

### Theory:

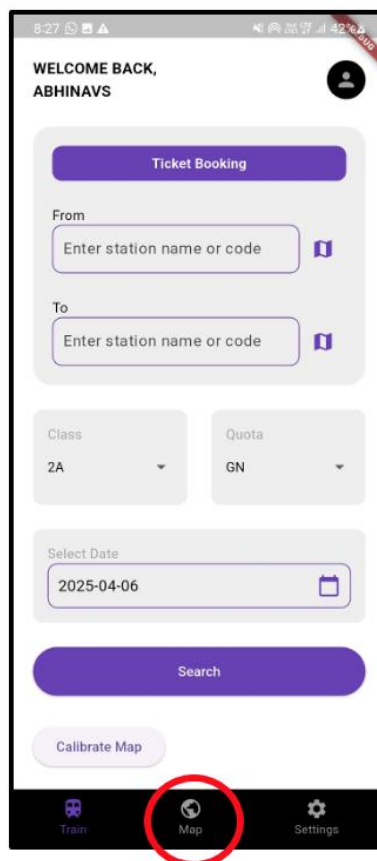
Navigation and gestures make Flutter apps dynamic and user-friendly. Flutter provides tools to easily move between pages (routes) and respond to user actions like taps and swipes.

In this experiment, we implemented navigation and routing to switch between different screens in a Flutter app. We also added gesture detection to make the app more interactive and responsive to user actions.

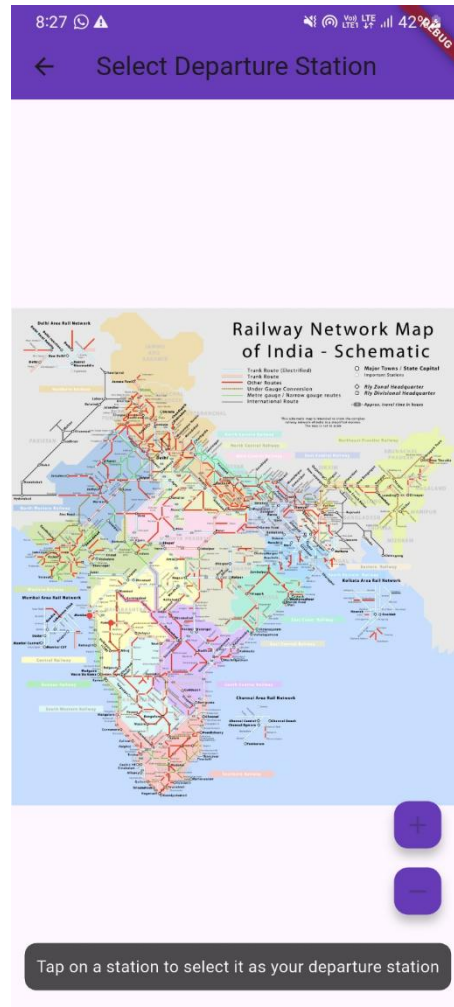
This experiment ties together user flow and responsiveness in the app.

### Output:

Route is **/dashboard**



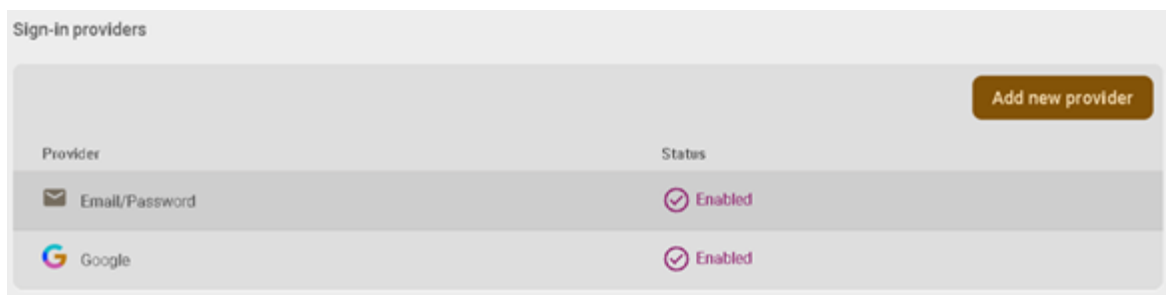
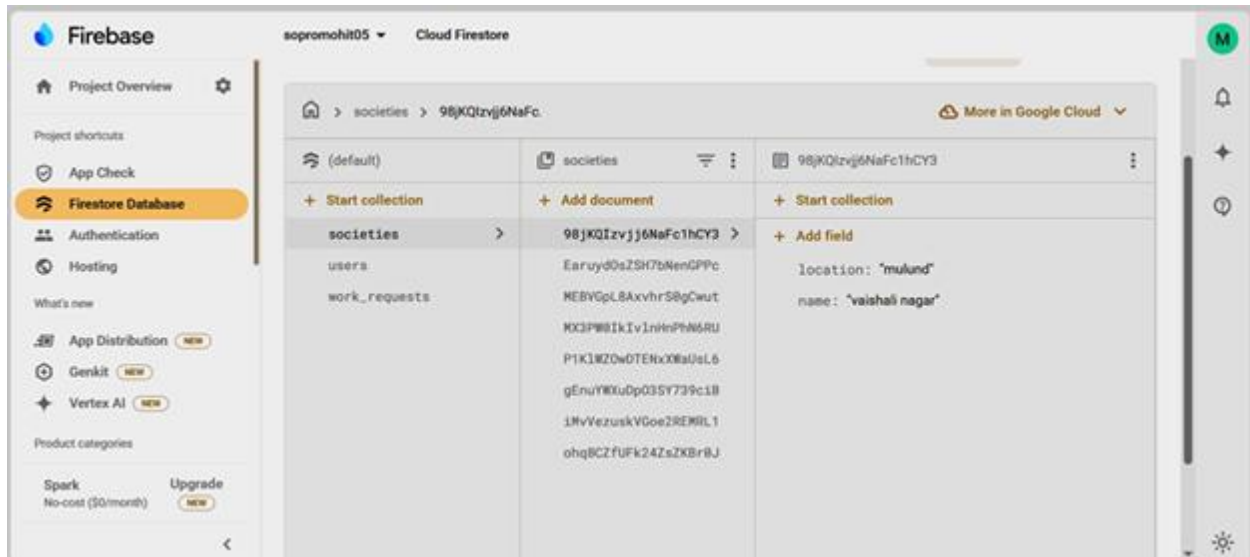
Route is /map



## Conclusion:

This experiment helped us understand the concepts of navigation and routing in Flutter, allowing smooth transitions between screens. Additionally, by using gesture detectors, we enhanced user interaction, making the app more dynamic and user-friendly.





## Conclusion:

This experiment provided hands-on experience in integrating Firebase with a Flutter app for both Android and iOS. It enables the use of powerful backend features like authentication, Firestore, and real-time updates, which are essential for modern app development.

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 7

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

### Theory:

Progressive Web Apps (PWAs) are web applications enhanced with modern web capabilities to deliver an app-like experience. One essential part of a PWA is the **Web App Manifest** — a JSON file that contains **metadata** about the app.

This metadata includes:

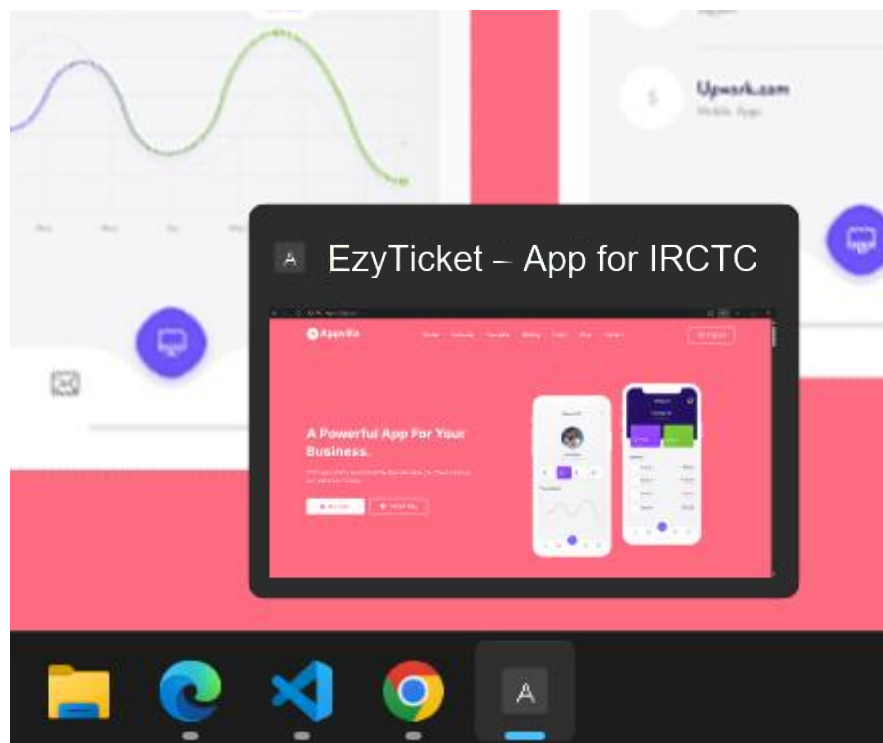
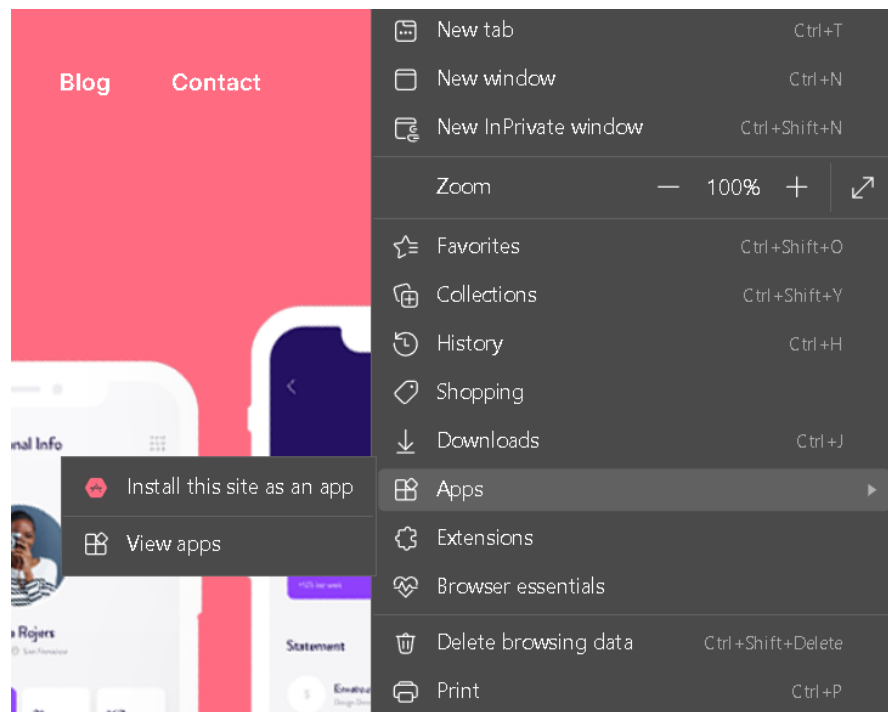
- App name and short name
- Start URL and scope
- Icons for different screen sizes
- Theme and background colors
- Display mode (e.g., standalone, fullscreen)

By linking this `manifest.json` file in the HTML, the app becomes **installable** on user devices and can appear on the home screen like a native app. This enhances user engagement, accessibility, and branding for the Ecommerce platform.

In this experiment, metadata was defined in a manifest file and connected to the main HTML file, enabling the “**Add to Home Screen**” functionality for the Ecommerce PWA.

### Output:

Name	Date modified	Type	Size
images	06-02-2025 10:05 PM	File folder	
app	06-02-2025 10:05 PM	JavaScript Source ...	4 KB
blog	07-02-2025 09:35 AM	Microsoft Edge HT...	17 KB
contact	07-02-2025 09:35 AM	Microsoft Edge HT...	10 KB
index	20-02-2025 08:33 PM	Microsoft Edge HT...	31 KB
style.css	06-02-2025 10:05 PM	CSSfile	56 KB



## Conclusion:

In this experiment, we successfully create and integrated a **Web App Manifest file** for our Progressive Web App. By defining essential metadata such as the app's name, icons, theme color, and display mode, we enabled the **“Add to Home Screen”** functionality.

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 8

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

### Theory:

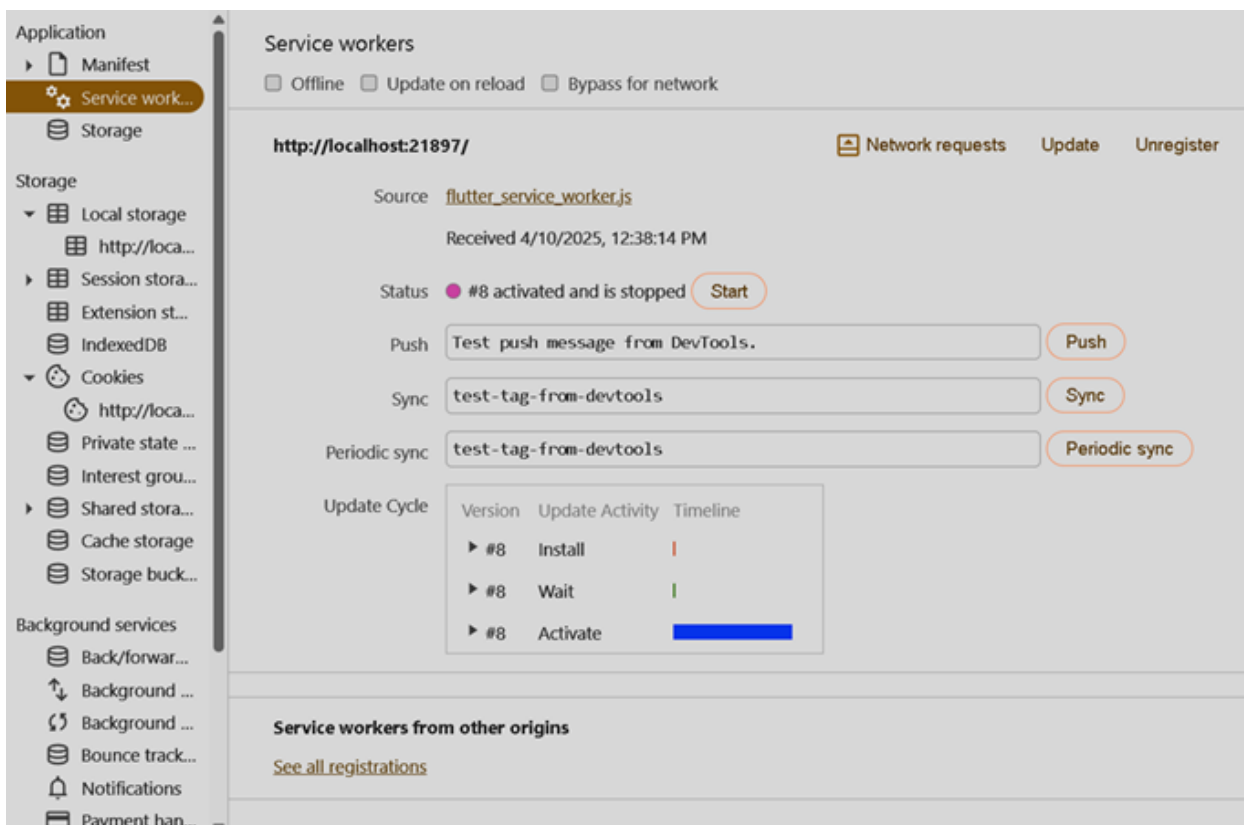
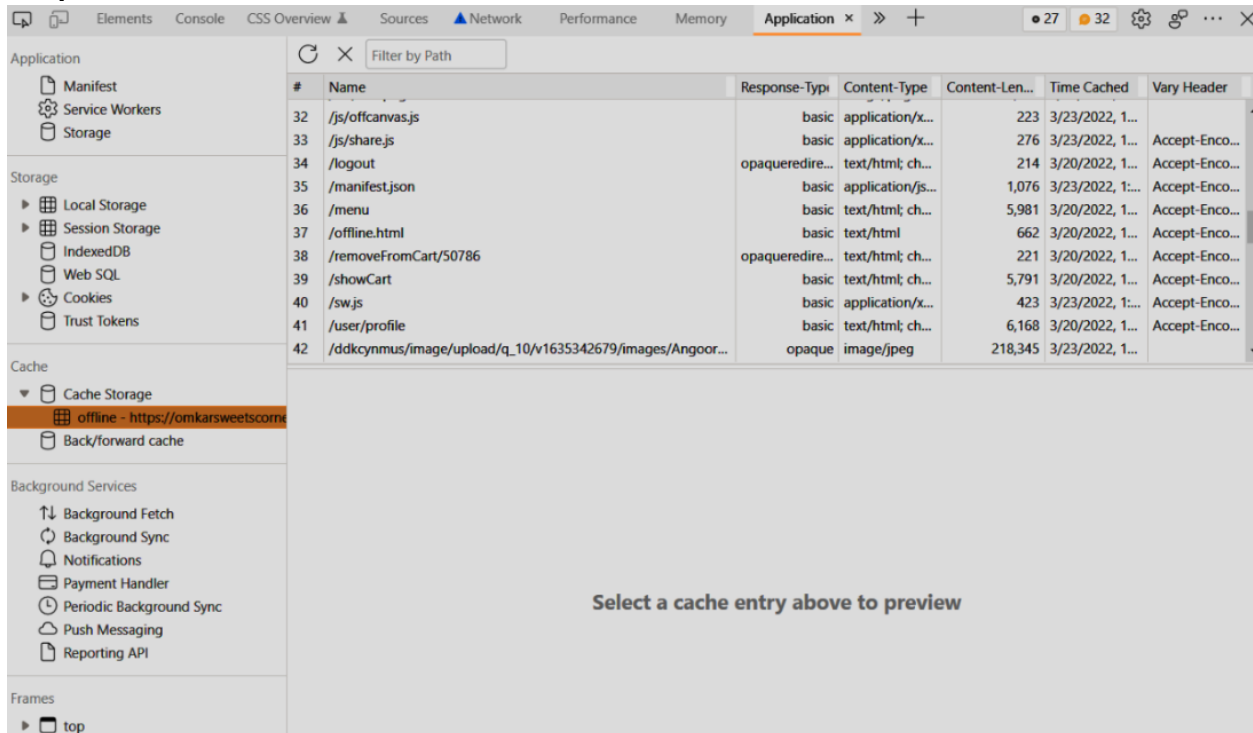
A **Service Worker** is a type of web worker script that runs in the background, separate from the main browser thread. It plays a key role in enabling **Progressive Web App (PWA)** features like offline support, background sync, and push notifications.

When a service worker is registered, it goes through three phases:

1. **Install** – Triggered once when the service worker is installed for the first time. Used to cache necessary assets.
2. **Activate** – Triggered when the service worker takes control of the page. Typically used for clearing out old caches.
3. **Fetch** – Intercepts network requests and serves cached responses if available, enabling offline access.

These features help improve **performance**, **reliability**, and **user experience**, especially in unstable or no network conditions.

## 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: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 9

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

### Theory:

A **Service Worker** is a JavaScript file that runs in the background of a Progressive Web App (PWA). It acts as a proxy between the web app and the network, enabling features like:

- **Caching content for offline use** (**fetch** event)
- **Syncing data in the background** (**sync** event)
- **Receiving and displaying push notifications** (**push** event)

These service worker events significantly improve user experience by ensuring fast loading, real-time updates, and engagement, even in low or no internet connectivity.

### Output:

1. Fetch Event

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

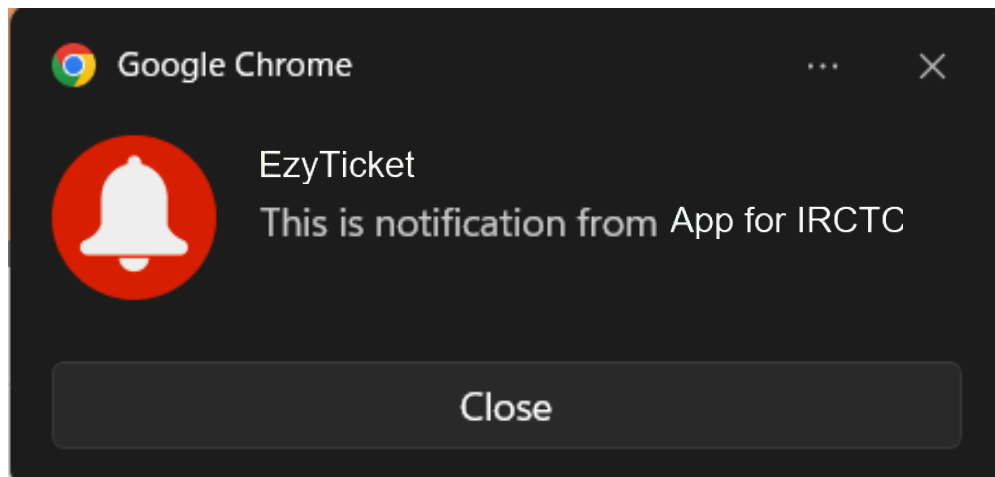
## 2. Sync Event

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

## 3. Push Event

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

Notification:



## Conclusion:

In this experiment, we successfully implemented the core **Service Worker** events (**fetch**, **sync**, and **push**) in the **railway PWA**. 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: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
DOP:	DOS:
Sign:	Grade:

## Experiment 10

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

### Theory:

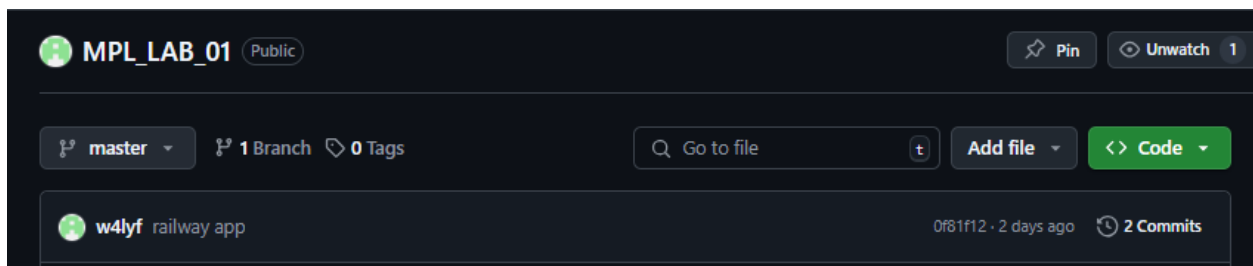
GitHub Pages is a free hosting platform by GitHub for static websites. It can serve HTML, CSS, JavaScript, and service worker files, making it a suitable option for deploying Progressive Web Apps (PWAs).

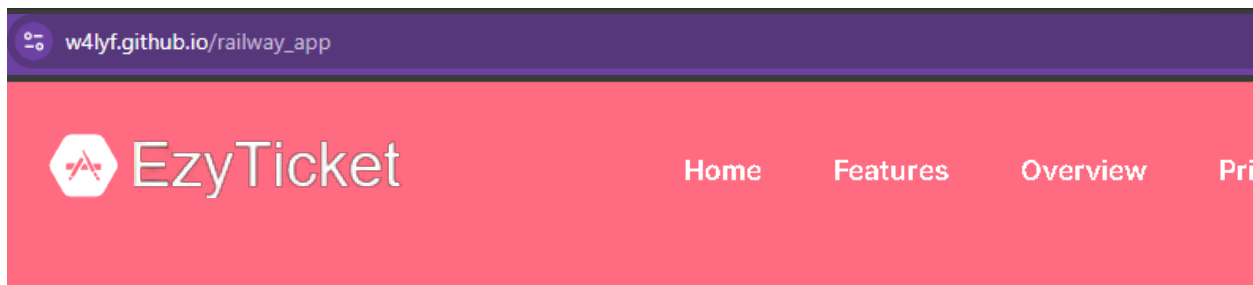
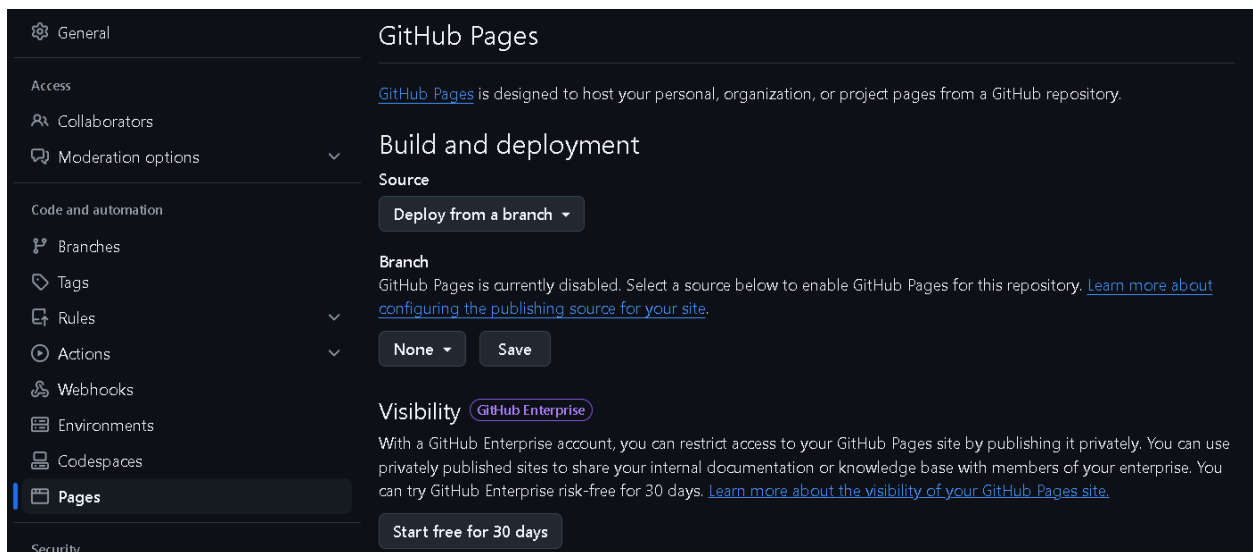
Deploying your PWA like railway app to GitHub Pages allows:

- Easy and quick sharing via a live URL.
- Hosting without any additional setup or cost.
- Support for offline features and "Add to Home Screen" via service workers and manifest file.
- Continuous updates through Git version control.

This helps you showcase and distribute your web app publicly with minimal effort.

### Output:





## Conclusion:

By deploying AppVilla as a Progressive Web App on GitHub Pages, we achieved a seamless and cost-effective way to make the application publicly accessible. This deployment not only ensures easy updates and version control through GitHub, but also enables core PWA features like offline access, faster loading, and the “Add to Home Screen” experience. Hosting on GitHub Pages simplifies the process of sharing and testing while providing a reliable platform to showcase the app's capabilities in real-world scenarios.

Name: <b>Abhinav Swaminathan</b>	Div-Roll no: <b>D15C-01</b>
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, automated tool developed by Google to improve the quality of web pages. It provides audits for performance, accessibility, SEO, best practices, and Progressive Web App (PWA) standards.

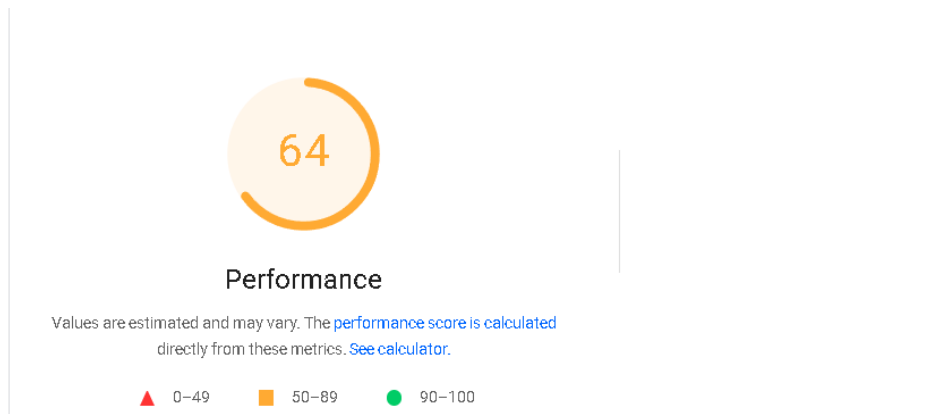
When testing a PWA, Lighthouse checks for critical requirements like:

- Valid web app manifest
- Presence and correct behavior of service workers
- HTTPS usage
- Responsive design
- Offline functionality
- “Add to Home Screen” capability

Lighthouse gives a score out of 100 based on how well the app performs as a PWA and offers suggestions for improvement. It is accessible directly in Chrome DevTools or as a browser extension.

This tool is crucial for ensuring the app meets modern web standards and provides a high-quality user experience on mobile and desktop.

### Output:



#### METRICS

[Expand view](#)

▲ First Contentful Paint

4.5 s

● Total Blocking Time

90 ms

■ Speed Index

5.7 s

▲ Largest Contentful Paint

6.2 s

● Cumulative Layout Shift

0

#### DIAGNOSTICS

- ▲ Largest Contentful Paint element — 6,230 ms
- ▲ Eliminate render-blocking resources — Potential savings of 1,890 ms
- ▲ Serve images in next-gen formats — Potential savings of 456 KiB
- ▲ Properly size images — Potential savings of 120 KiB
- ▲ Defer offscreen images — Potential savings of 89 KiB
- ▲ Reduce unused JavaScript — Potential savings of 72 KiB
- ▲ Reduce unused CSS — Potential savings of 18 KiB
- Image elements do not have explicit `width` and `height`
- Minify CSS — Potential savings of 2 KiB
- Serve static assets with an efficient cache policy — 27 resources found
- Efficiently encode images — Potential savings of 170 KiB

## Conclusion:

Using Google Lighthouse, we successfully analyzed the PWA capabilities of railway 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.