Project Report

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

Kar-Rental: The Car Rental Management Application

Introduction

The Car Rental Management Application is a user-friendly mobile app designed to streamline the process of renting vehicles for both customers and administrators. Whether it's for a quick city drive or a long weekend getaway, users can browse, book, and manage car rentals with ease and efficiency. With real-time vehicle availability, secure booking, and integrated payment gateways, the app ensures a seamless experience from reservation to return. Customers can view car details, compare prices, track rental history, and receive timely reminders about pick-up and drop-off schedules. For admins, the platform provides robust tools for managing fleet inventory, monitoring bookings, and generating reports.

Problem Statement

In today's fast-paced world, individuals and businesses often face challenges when trying to rent vehicles efficiently and reliably. Traditional car rental systems can be time-consuming, lack transparency, and offer limited accessibility, especially in urgent or last-minute scenarios. Issues such as unclear vehicle availability, manual booking processes, and poor communication between service providers and customers contribute to user frustration and operational inefficiencies.

Solution

The Car Rental Management Application addresses the common challenges faced in vehicle rental, such as limited accessibility, poor tracking, and inefficient booking systems. It streamlines the entire rental process through a user-friendly mobile interface that allows customers to browse available cars, compare pricing, and book vehicles instantly. With real-time vehicle availability, GPS-based location services, and secure payment integration, the app ensures convenience, transparency, and reliability. For administrators, it offers powerful tools to manage bookings, monitor fleet status, and generate insightful reports.

Key Technologies:

- **Flutter**: Delivers a smooth and interactive user interface across platforms.

- **MongoDB**: Stores user profiles, emergency contact lists, Cars List, and video data efficiently.
- **Node.js**: Handles business logic, Searching, and real-time alert processing via APIs.

Features

The application includes the following key functionalities:

- **Emergency SOS Alert**: Allows users to notify pre-saved emergency contacts with their GPS location through a single button press.
- **Preparedness Quizzes**: Provides scenario-based questions that help assess a user's emergency preparedness level.
- **Secure Login**: Authenticates users through email and OTP, ensuring personal data and emergency contacts remain protected.

These features empower users to take control of emergency situations both reactively and proactively.

Design and Architecture

The application adopts the **Model-View-Controller (MVC)** architecture for clear separation of concerns:

- **Model**: Managed using MongoDB, storing users' personal data, quiz progress, emergency contacts, and video metadata.
- **View**: Created using Flutter, ensuring a responsive and visually engaging user experience.
- **Controller**: Handled via Node.js, managing user interactions, alert delivery, and data processing.

The app uses **RESTful APIs** for efficient communication between the Flutter front-end and the Node.js back-end.

Project Report

Implementation Details

SOS Alert System The emergency alert system captures the user's GPS location upon activation and sends an alert message along with a map link to all pre-configured emergency contacts via email or SMS. This ensures immediate visibility of the user's location during a crisis. **Quiz Module** Each quiz contains multiple-choice questions focused on various disaster scenarios. The backend evaluates the responses, stores user scores in MongoDB, and provides feedback to help improve disaster awareness. **Educational Content** A curated set of emergency-related videos is integrated into the app. These videos are hosted or embedded securely and provide step-by-step visual guides for handling injuries, CPR, or fire outbreaks. **Security Measures** Login functionality is secured through email-based OTP verification, ensuring only authorized users can access personal and sensitive information like emergency contacts.

Challenges

Key challenges faced during development included real-time location accuracy, timely delivery of SOS alerts, and maintaining video performance across platforms. These were addressed through GPS optimization, integration with reliable SMS/email APIs, and using efficient video loading techniques.

Results

The application has proven effective in emergency simulations and user testing. The SOS feature performs reliably, sending accurate location-based alerts, while the quizzes and video tutorials have been praised for their clarity and educational value. While the app is still in its early stage, it successfully achieves its primary goal of providing both emergency response capabilities and disaster preparedness education.

Future Scope

There is significant potential to enhance the Disaster Preparedness and Management Application with the following upgrades:

- **Real-Time Emergency Services Integration**: Direct alerting to nearby hospitals or fire stations.

- **Offline Support** : Storing critical guides and tutorials for access without an internet connection.
- **Multi-language Support**: Making the content accessible to a broader audience.

Conclusion

The Car Rental Management Application showcases how digital innovation can transform the traditional vehicle rental experience into a seamless, efficient, and user-centric process. With features such as real-time car availability, secure booking, GPS integration, and streamlined fleet management, the app enhances convenience for customers while providing robust tools for administrators. Built using Flutter, Node.js, and MongoDB, the application stands on a solid technical foundation that ensures scalability, performance, and cross-platform accessibility. As the platform continues to evolve, it holds significant potential to become a leading solution in the mobility sector, redefining how individuals and businesses access transportation.

Experiment - 1

Name: Shivpratik Hande	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM : Installation and Configuration of the Flutter Environment

Theory :

1. Install Flutter SDK:

- Visit flutter.dev.
- Download the Windows installation bundle.

2. Set Flutter Path:

- Go to **This PC > Properties > Advanced system settings > Environment Variables**.
- Under "System variables", edit Path.
- Add: C:\Flutter\bin.

3. Verify Installation:

- Run flutter doctor in CMD to check dependencies and configuration.

4. Install Android Studio (if Android SDK is missing):

- Download from developer.android.com/studio.
- Run the installer and complete the setup.

5. Set Android Emulator:

- Open Android Studio > Tools > AVD Manager > Create Virtual Device.
- Choose device > Select latest Android system image > Confirm config > Finish.
- Start the emulator by clicking the green play icon.

6. Install Flutter & Dart Plugins:

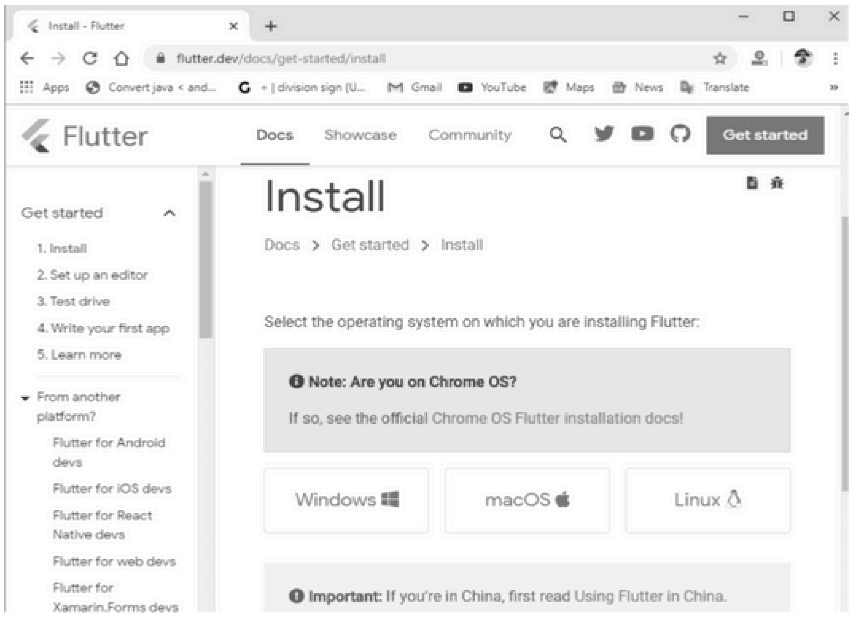
- In Android Studio: File > Settings > Plugins.
- Search & install Flutter (auto-prompts Dart plugin) and restart Android Studio.

Experiment - 1

7. Accept Licenses:

- Run flutter doctor --android-licenses in CMD.

The screenshots below are the installation and configuration steps for Flutter & Android Studio.



Install

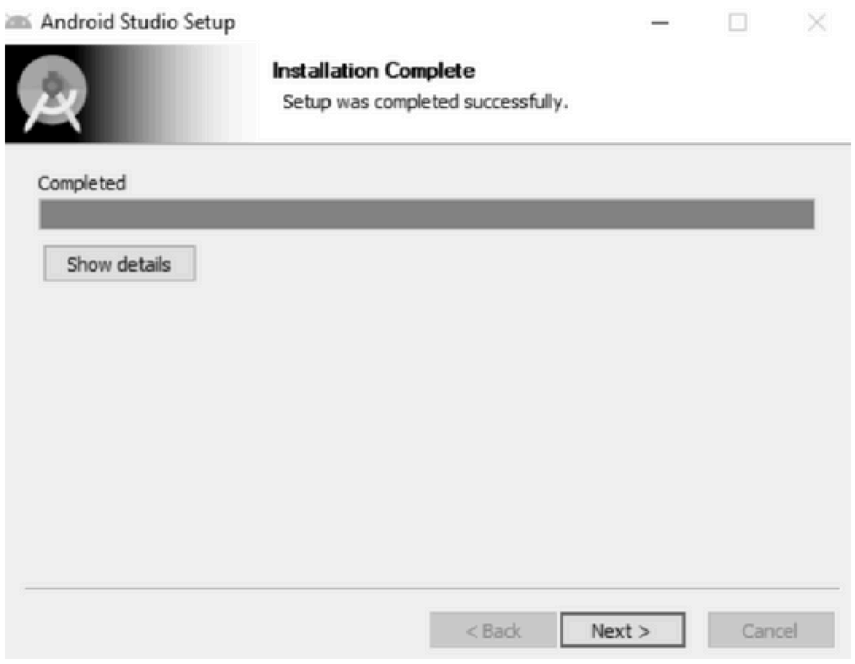
Docs > Get started > Install

Select the operating system on which you are installing Flutter:

Note: Are you on Chrome OS?

If so, see the official Chrome OS Flutter installation docs!

Important: If you're in China, first read Using Flutter in China.



Installation Complete

Setup was completed successfully.

Completed

Conclusion:

These are the series of steps involved in setting up and configuring Flutter for working on Windows along with Android Emulator.

Experiment - 2

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM: To design Flutter UI by including common widgets.

Theory :

Flutter is a UI toolkit developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. One of Flutter's core strengths is its rich set of **predefined widgets**, which can be used to create responsive, beautiful, and functional user interfaces.

1. Understanding Widgets in Flutter: Everything in Flutter is a widget, whether it's structural

elements (like buttons or text), stylistic
elements (like padding or alignment), or layout models (like rows and columns). Widgets are either:

- **StatelessWidget:** A widget that does not require mutable state.
- **StatefulWidget:** A widget that maintains mutable state and can rebuild when the state changes.

2. Commonly Used Flutter Widgets:

Here are some of the most frequently used widgets in Flutter UI development:3. UI Design Structure:
Text - Displays a string of text.

Container - A box model widget used for layout and styling.

Row / Column - Aligns children horizontally (Row) or vertically (Column).

Image - Displays images from assets or networks.

Icon - Displays material design icons.

ElevatedButton - A material design button with elevation.

TextField - Accepts user input in text form.

Experiment - 2

ListView - A scrollable list of widgets.

Card - A material design card with rounded corners and elevation.

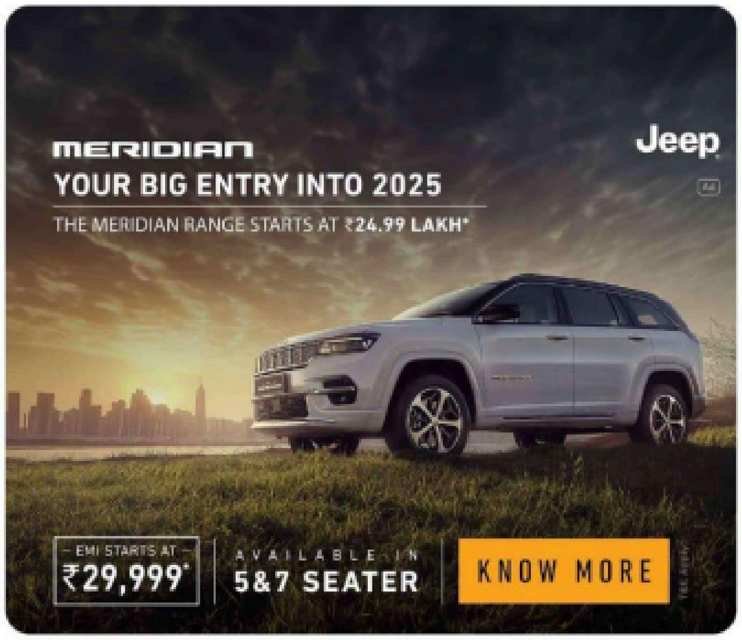
Flutter UI is built by combining widgets in a tree structure, often referred to as the **widget tree**. The layout is achieved using parent-child relationships where each widget nests within another.

4. Customization and Styling:

Widgets in Flutter are highly customizable using properties like:

- `padding, margin, alignment for layout`
- `color, fontSize, borderRadius for styling`
- BoxDecoration, TextStyle, etc., for advanced UI effects

An example of the use of widgets inside of my application,each page is navigated to using a separate widget.



Conclusion:

Designing a Flutter UI using common widgets provides a modular and reusable approach to app development. Mastering these core widgets is essential for creating interactive, responsive, and visually appealing user interfaces.

Experiment - 3

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:
To include icons, fonts, and images in your Flutter app.

Theory: Flutter supports the use of icons, custom fonts, and images to enhance the visual appeal and user experience of an application. These elements are essential for creating visually engaging and brand-consistent user interfaces. **1. Icons in Flutter** Flutter provides built-in support for Material Design icons through the Icons class. Developers can customize these icons in terms of size and color. For extended icon sets, third-party packages like font_awesome_flutter can be integrated to access a wider range of icons. **2. Custom Fonts** To maintain brand consistency and improve design aesthetics, Flutter allows the use of custom fonts. Font files are placed in the assets directory and declared in the pubspec.yaml file. Once added, they can be applied throughout the app using TextStyle. This helps in maintaining a consistent and unique typography style. **3. Images in Flutter**

Flutter supports three types of images: asset images, network images, and file images.

- **Asset Images** are bundled with the app and are commonly used for logos or static content.

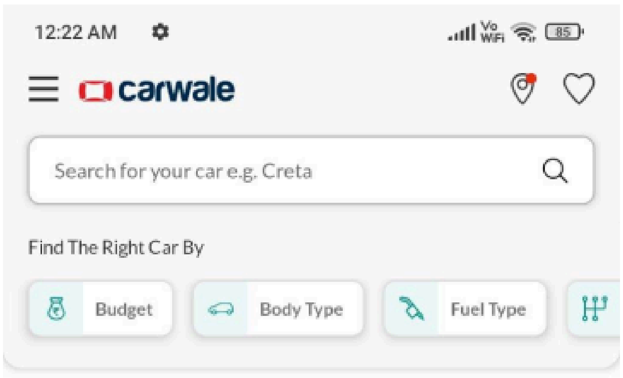
- **Network Images** are fetched from the internet and are useful for dynamic content.

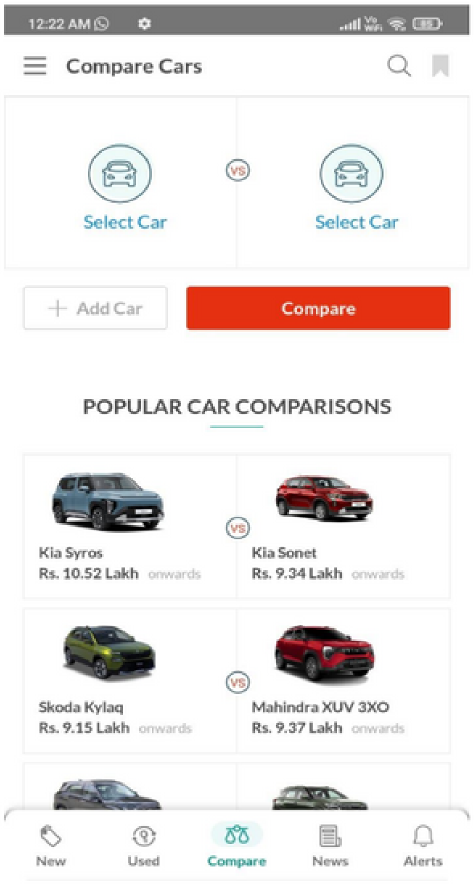
- **File Images** are used when accessing images stored locally on a device.

Experiment - 3

To use asset images or fonts, developers must declare them in the pubspec.yaml file. Proper asset management ensures that resources load efficiently and appear correctly across different screen sizes.

Example usage inside of my Application :





The first page makes use of icons for the functionalities of showing the user account,the home page along with the icons for the various pages that the application has.
The application also contains safety and tutorial demonstration videos which are also a part of the image and video assets that the application has.

Conclusion:

Adding icons, custom fonts, and images significantly improves the user interface and experience in Flutter apps. With proper asset organization and usage, developers can build attractive, accessible, and responsive applications.

Experiment - 4

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

To create an interactive Form using Form widget.

Theory:

Forms are essential components in mobile applications for collecting user input such as login credentials, contact details, or feedback. In Flutter, the Form widget is used to group and manage multiple form fields efficiently.

1. Form Widget Overview

The Form widget acts as a container for input fields like TextFormField, DropdownButtonFormField, etc. It provides a mechanism to validate and save the input data. Each form can be associated with a GlobalKey to manage its state.

2. Common Input Widgets

- TextFormField: Used for text input with built-in validation support.
- Checkbox, Radio, Switch: For selecting options.
- DropdownButtonFormField: For dropdown selections.
  These fields are interactive and respond to user actions in real-time.

3. Validation and State Management

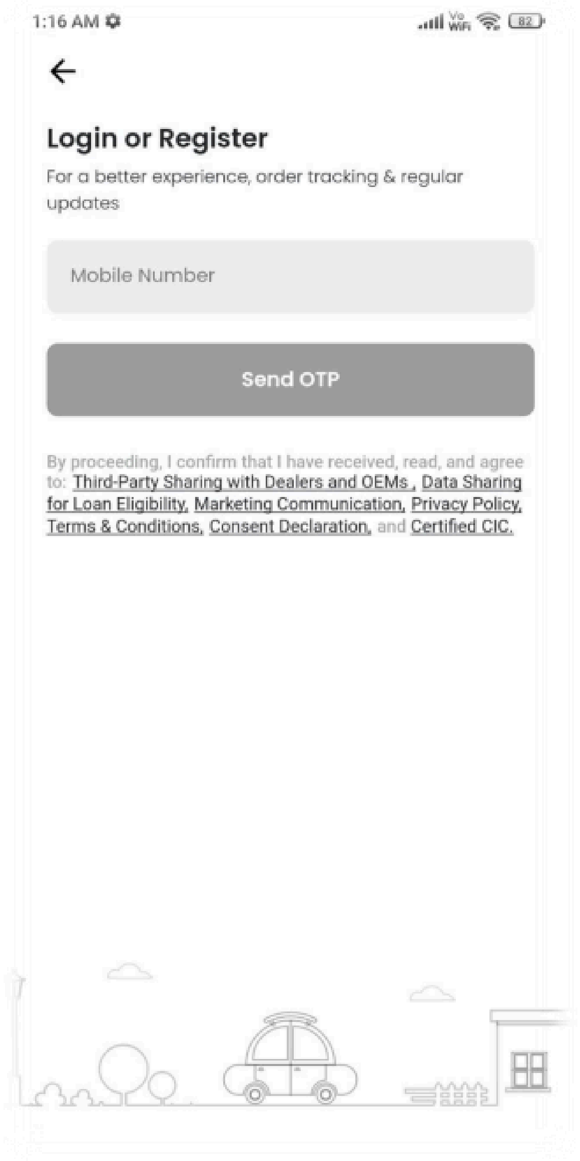
Form validation ensures the correctness of user input before submission. Flutter supports built-in and custom validation logic. The FormState object helps in validating, resetting, and saving form data. This enhances reliability and user experience.

4. User Interaction

Flutter forms support dynamic updates and real-time feedback using validation messages and input listeners. Focus and keyboard control make forms more user-friendly and interactive.

Experiment - 4

Using a simple Login/Sign Up form inside of my application.



1:16 AM

Login or Register

For a better experience, order tracking & regular updates

Mobile Number

Send OTP

By proceeding, I confirm that I have received, read, and agree to: Third-Party Sharing with Dealers and OEMs , Data Sharing for Loan Eligibility, Marketing Communication, Privacy Policy, Terms & Conditions, Consent Declaration, and Certified CIC.

Conclusion:
Creating forms using Flutter’s Form widget provides a structured and efficient way to collect and validate user input. Proper use of form fields and validation improves interactivity and ensures a seamless user experience.

Experiment - 5

Name: Shivpratik Hande	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

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

Theory :
Flutter is a powerful framework for building cross-platform mobile applications, and it provides efficient mechanisms for:

Navigation and Routing :
Navigation allows moving between different screens (also called routes or pages) in a Flutter app. Flutter provides multiple methods to implement navigation:

a) Basic Navigation (Navigator.push and Navigator.pop)
- Navigator.push(context, MaterialPageRoute(builder: (_) => SecondPage())); Pushes a new route onto the stack.
- Navigator.pop(context); Pops the top-most route from the stack and returns to the previous screen

b) Named Routing
- Define routes in MaterialApp's routes property:

c) Navigation Stack
Flutter uses a stack-based navigation model where each new screen is "pushed" onto a stack and can be "popped" to return to the previous screen.

2. Gestures in Flutter
Gestures are used to detect user interaction like taps, swipes, drags, etc.
Flutter uses the GestureDetector widget to handle gestures:

Gesture Type	Widget/Callback Used
Tap	onTap
Double Tap	onDoubleTap
Long Press	onLongPress
Vertical Drag on	VerticalDragUpdate
Horizontal Drag on	HorizontalDragUpdate

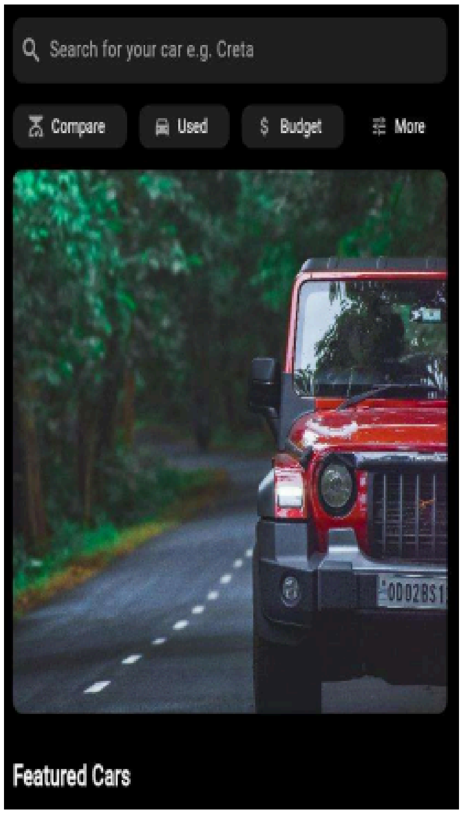
Experiment - 5

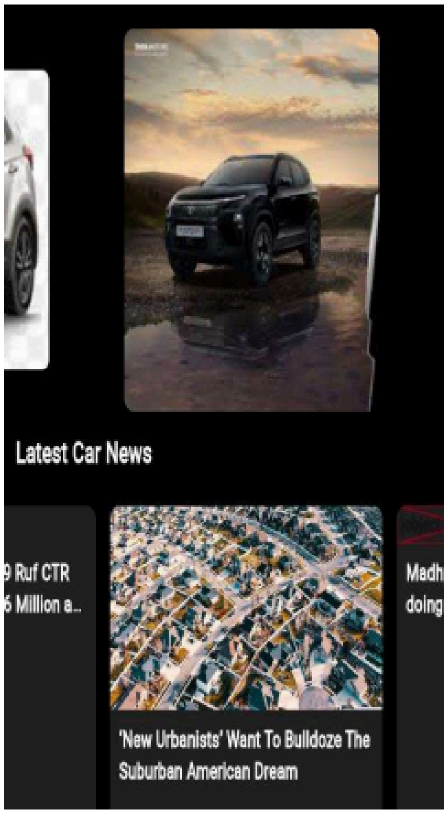
GestureDetector is a powerful tool for creating interactive UIs and responding to user inputs like swipe-to-dismiss, tap-to-select, or drag-to-move elements.

Example usage of navigation and routing in Flutter

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomePage(),
    '/second': (context) => SecondPage(),
  },
);
// Navigate using named route
Navigator.pushNamed(context, '/second');
```

Pages and Navigation inside of my Disaster Preparedness Application




Latest Car News
9 Ruf CTR
6 Million a...
'New Urbanists' Want To Bulldoze The Suburban American Dream
Madh
doing

Conclusion : These are the various ways in which navigation, gestures and routing were implemented inside the application.

Experiment - 6

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

To connect Flutter to Firebase.

Theory:

Firebase is a Backend-as-a-Service (BaaS) platform by Google that provides a suite of cloud-based tools to help developers build and scale mobile applications. Flutter integrates seamlessly with Firebase to support real-time data storage, user authentication, cloud functions, push notifications, and more. **1. Firebase Integration with Flutter**

To connect Flutter with Firebase, the project must first be registered on the Firebase Console. The FlutterFire plugins (such as firebase_core, firebase_auth, cloud_firestore, etc.) are used to enable Firebase services in the app. These packages are added via pubspec.yaml. **2. Initialization** The firebase_core plugin is essential to initialize Firebase in a Flutter app. Initialization is typically done in the main() method using WidgetsFlutterBinding.ensureInitialized() followed by Firebase.initializeApp().

3. Common Firebase Services in Flutter

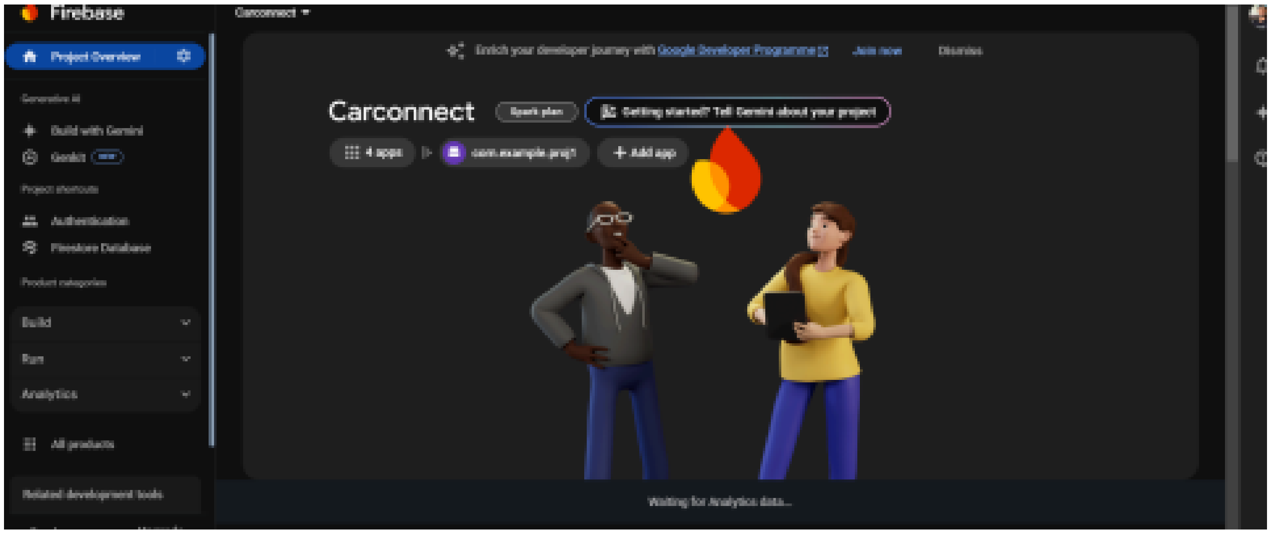
- **Firebase Authentication**: Supports user sign-in with email/password, Google, phone number, etc.
- **Cloud Firestore**: A NoSQL cloud database used for storing and syncing real-time data.
- **Firebase Storage**: For storing user-generated files such as images or videos.
- **Firebase Realtime Database**: Another real-time data syncing option with a simpler structure.

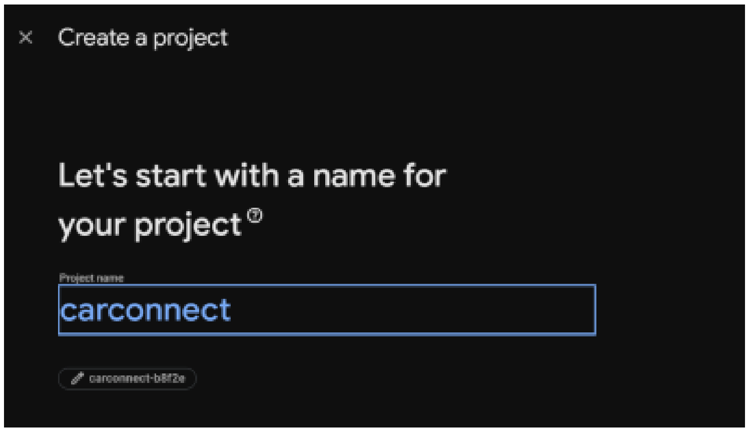
4. Advantages of Using Firebase

- Easy integration with minimal backend setup.

Experiment - 6

- Real-time database syncing.
- Scalable cloud infrastructure.
- Built-in security and analytics tools.

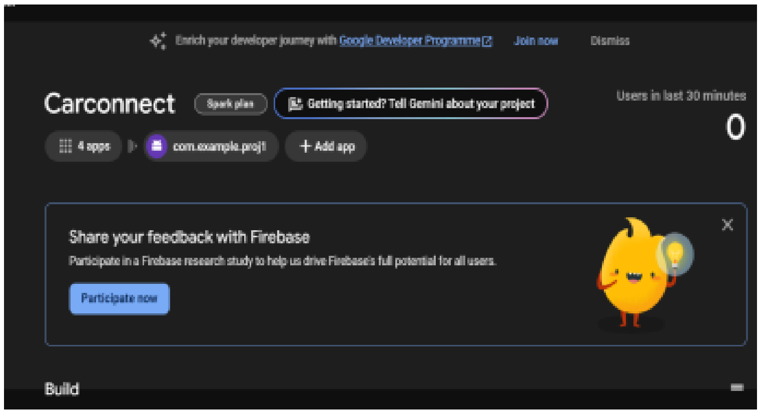



Create a project

Let's start with a name for your project

Project name: carconnect

carconnect-b8f2e



Share your feedback with Firebase

Participate in a Firebase research study to help us drive Firebase's full potential for all users.

Conclusion:
Connecting Flutter to Firebase allows developers to build full-featured apps with cloud-based backend services. It simplifies authentication, database management, and data storage, enabling faster development and scalable solutions.

Experiment - 7

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

To write metadata of your E-commerce PWA in a Web App Manifest file to enable “Add to Homescreen” feature.

Theory:

A Progressive Web App (PWA) combines the best of web and mobile apps. One key feature of PWAs is the ability to be installed on a user’s device directly from the browser using the **“Add to Homescreen”** feature. This functionality is enabled by defining a **Web App Manifest** file.

1. What is a Web App Manifest?
A Web App Manifest is a simple JSON file that contains metadata about the app, such as its name, icons, start URL, background color, display mode, and theme. It tells the browser how the app should appear when installed and launched from the home screen. **2. Enabling Add to Homescreen**

For the “Add to Homescreen” feature to work:

- A valid manifest file must be linked in the HTML <head>.
- The app must be served over HTTPS.
- A registered service worker must handle caching and offline support.

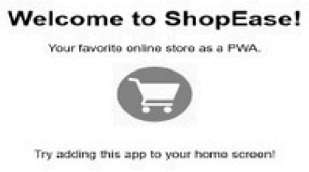
3. Benefits for E-commerce PWAs

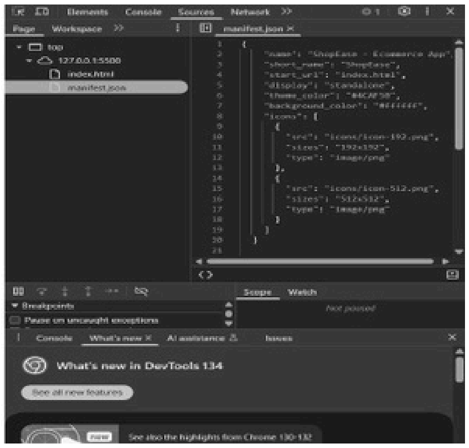
- Provides app-like experience.
- Increases user engagement and return visits.
- Works offline or in low-network conditions.
- Enhances brand visibility with custom icons and splash screens.

```
{
```

Experiment - 7

```
"name": "ShopEase - Your Ecommerce Store",
"short_name": "ShopEase",
"start_url": "/index.html",
"display": "standalone",
"background_color": "#ffffff",
"theme_color": "#4CAF50",
"icons": [
{
"src": "icons/icon-192x192.png",
"sizes": "192x192",
"type": "image/png"
},
{
"src": "icons/icon-512x512.png",
"sizes": "512x512",
"type": "image/png
```


Welcome to ShopEase!
Your favorite online store as a PWA.
Try adding this app to your home screen!



```
{
  "name": "ShopEase - Ecommerce App",
  "short_name": "ShopEase",
  "start_url": "index.html",
  "display": "standalone",
  "theme_color": "#4CAF50",
  "background_color": "#ffffff",
  "icons": [
    {
      "src": "icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icons/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Conclusion:
Adding a Web App Manifest file with proper metadata is essential for enabling the “Add to Home Screen” feature in E-commerce PWAs. It improves usability, promotes user retention, and delivers a seamless mobile-like experience.

Experiment - 8

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

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:

A **Service Worker** is a script that the browser runs in the background, separate from a web page, enabling features like **offline access**, **background sync**, and **push notifications**. It is essential for building Progressive Web Apps (PWAs) that work reliably regardless of network conditions. **1. What is a Service Worker?**

A service worker acts as a network proxy between the web app and the internet. It intercepts network requests, allowing developers to control caching strategies and serve content from cache when offline. **2. Service Worker Lifecycle**

The service worker follows three main phases:

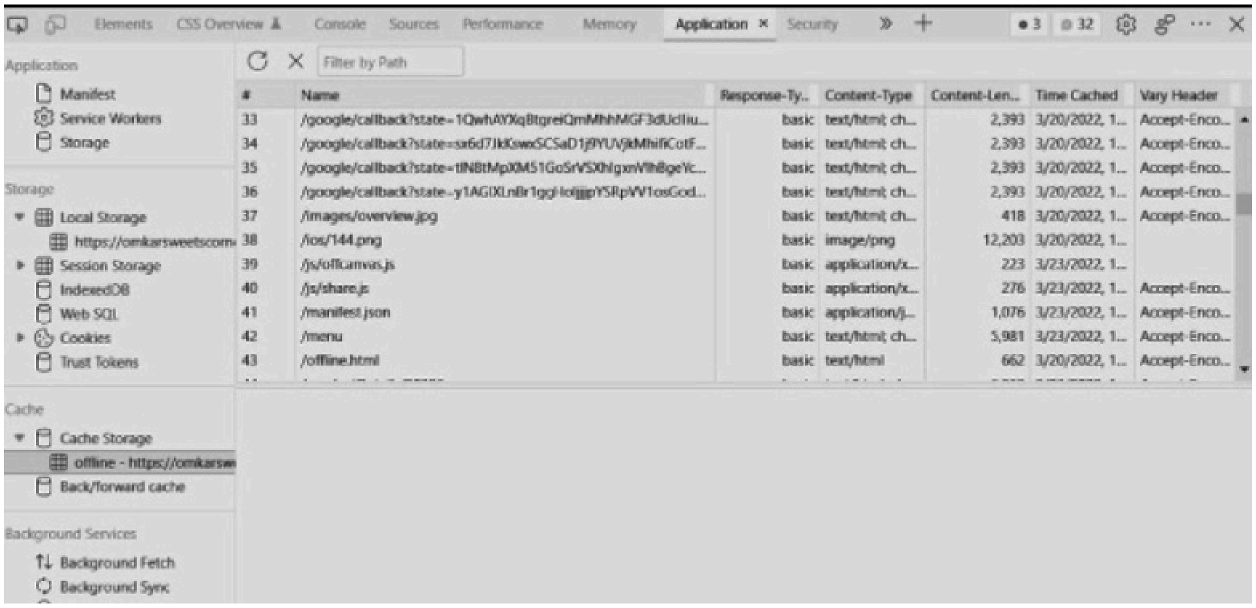
- **Registration**: Linking the service worker file with the browser.
- **Installation**: Caching required static assets.
- **Activation**: Taking control of all pages and clearing old caches if needed.

3. Registration Process
The service worker is registered in the main JavaScript file. Once registered, the browser handles the lifecycle events automatically and activates the worker if all steps succeed.
4. Importance for E-commerce PWAs

- Ensures smooth browsing even during connectivity issues.
- Speeds up loading by serving cached content.
- Increases reliability and user engagement.

Experiment - 8

- Enables "Add to Homescreen" and background features.



#	Name	Response-Ty..	Content-Type	Content-Len...	Time Cached	Vary Header
33	/google/callback?state=1QwhAYXqBtgreiQmMhhMGF3dUclIu...	basic	text/html; ch...	2,393	3/20/2022, 1...	Accept-Enco...
34	/google/callback?state=sx6d7JkKswxSCSaD1j9YUVjkMhifiCotF...	basic	text/html; ch...	2,393	3/20/2022, 1...	Accept-Enco...
35	/google/callback?state=tlN8tMpXM51GoSrVSXhlgxnVlhBgeYc...	basic	text/html; ch...	2,393	3/20/2022, 1...	Accept-Enco...
36	/google/callback?state=y1AGIXLnBr1ggHoljjjpYSRpVV1osGod...	basic	text/html; ch...	2,393	3/20/2022, 1...	Accept-Enco...
37	/images/overview.jpg	basic	text/html; ch...	418	3/20/2022, 1...	Accept-Enco...
38	/ios/144.png	basic	image/png	12,203	3/20/2022, 1...	
39	/js/offcanvas.js	basic	application/x...	223	3/23/2022, 1...	
40	/js/share.js	basic	application/x...	276	3/23/2022, 1...	Accept-Enco...
41	/manifest.json	basic	application/j...	1,076	3/23/2022, 1...	Accept-Enco...
42	/menu	basic	text/html; ch...	5,981	3/23/2022, 1...	Accept-Enco...
43	/offline.html	basic	text/html	662	3/20/2022, 1...	Accept-Enco...

Conclusion:
Coding and registering a service worker is crucial in transforming a traditional web app into a fully functional PWA. It enhances user experience by providing offline capabilities, faster performance, and better engagement for E-commerce applications.

Experiment - 9

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

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

Theory:

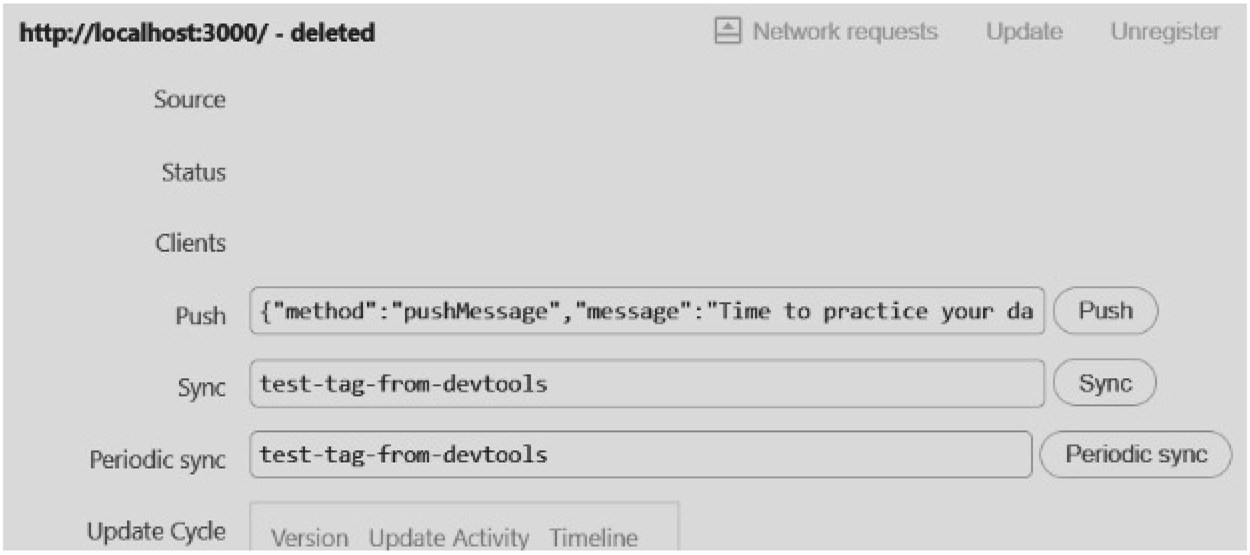
Service Workers enhance Progressive Web Apps (PWAs) by enabling advanced features like offline access, background synchronization, and push notifications. These capabilities rely on specific service worker events: **fetch**, **sync**, and **push**. **1. Fetch Event** This event is triggered when the app makes a network request. The service worker can intercept these requests and respond with cached data or fetch from the network. This allows the app to work offline or load faster using stored resources. **2. Sync Event** The background sync event is triggered when the device regains connectivity. It allows the service worker to retry previously failed network operations like form submissions or cart updates, ensuring data consistency in poor network conditions. **3. Push Event**

This event is triggered when the server sends a push notification to the service worker. It allows the app to receive real-time updates or alerts even when it is not actively open in the browser.

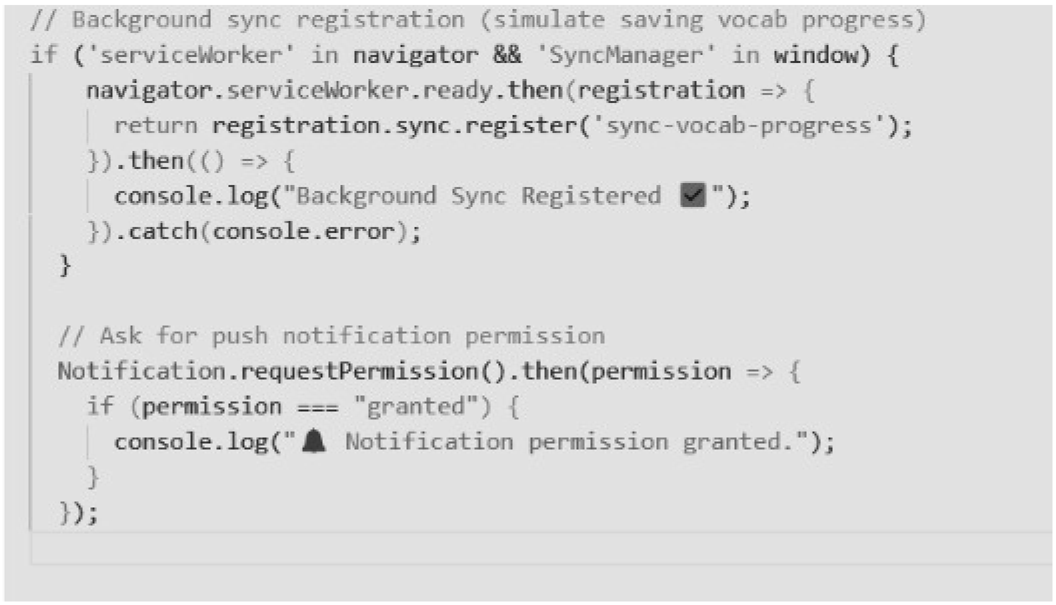
4. Benefits for E-commerce PWAs

- **Fetch** ensures essential product pages and assets are available offline.
- **Sync** helps maintain accurate orders and cart data.
- **Push** keeps users engaged with alerts on offers, orders, or stock updates.

Experiment - 9



```
{"method":"pushMessage","message":"Time to practice your da
test-tag-from-devtools
test-tag-from-devtools
```



```
// Background sync registration (simulate saving vocab progress)
if ('serviceWorker' in navigator && 'SyncManager' in window) {
    navigator.serviceWorker.ready.then(registration => {
      return registration.sync.register('sync-vocab-progress');
    }).then(() => {
      console.log("Background Sync Registered ✅");
    }).catch(console.error);
  }

  // Ask for push notification permission
  Notification.requestPermission().then(permission => {
    if (permission === "granted") {
      console.log("🔔 Notification permission granted.");
    }
  });
```

Conclusion:
Implementing fetch, sync, and push events in a service worker greatly improves the functionality and reliability of an E-commerce PWA. It ensures better performance, user engagement, and seamless experience across network conditions.

Experiment - 10

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

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

Theory:

GitHub Pages is a free hosting service that allows developers to publish web apps directly from a GitHub repository. It is commonly used to deploy static websites, including Progressive Web Apps (PWAs), with a simple workflow. **1. Overview of Deployment**

To deploy an E-commerce PWA on GitHub Pages:

- The project must be pushed to a GitHub repository.
- It should be built into static files (e.g., using npm run build).
- The build output must be served from the appropriate branch (usually gh-pages).

2. Key Steps

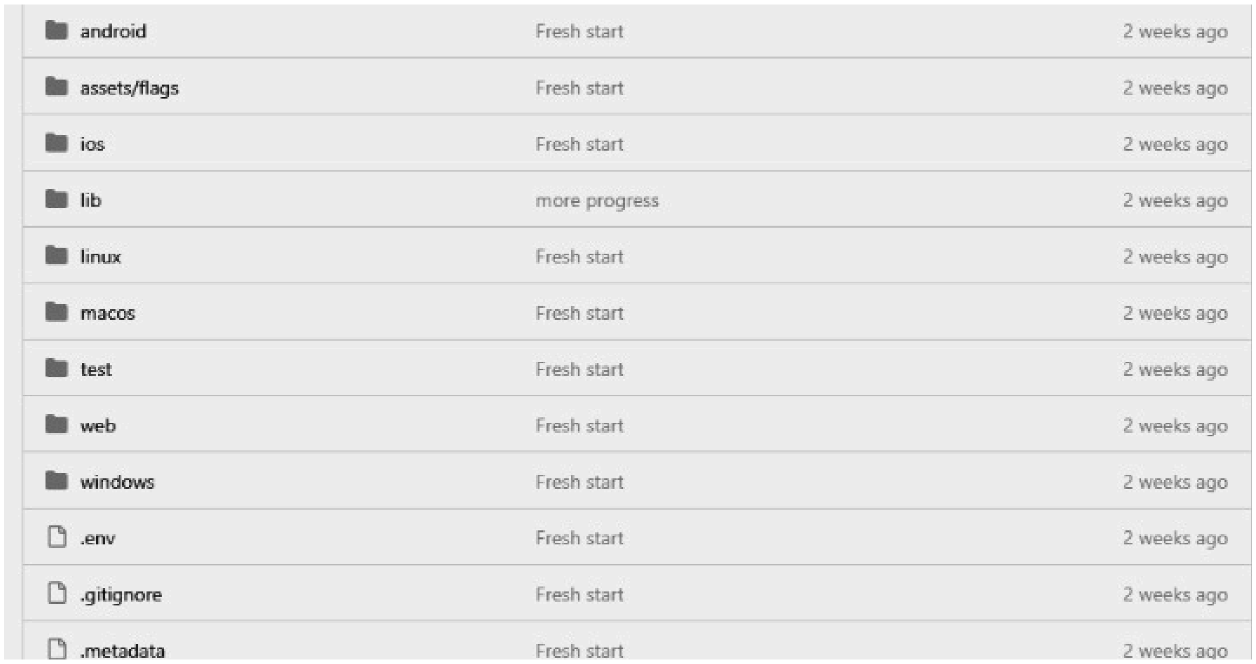
- **Repository Setup**: Push the PWA code to GitHub.
- **Build Process**: Generate optimized static files suitable for deployment.
- **Deployment Configuration** : Use tools like gh-pages npm package or GitHub Actions for automating deployment.
- **Set Homepage URL**: In package.json, define the homepage URL to ensure assets load correctly.

3. Benefits of Using GitHub Pages

- Free and easy to use.

- No server configuration required.
- Fast global content delivery via GitHub's CDN.
- Ideal for hosting PWAs and showcasing front-end projects.

4. Importance for E-commerce PWAs
Deploying to GitHub Pages makes the PWA publicly accessible, allowing users to install it, browse products, and interact with features in a real-world environment.



android	Fresh start	2 weeks ago
assets/flags	Fresh start	2 weeks ago
ios	Fresh start	2 weeks ago
lib	more progress	2 weeks ago
linux	Fresh start	2 weeks ago
macos	Fresh start	2 weeks ago
test	Fresh start	2 weeks ago
web	Fresh start	2 weeks ago
windows	Fresh start	2 weeks ago
.env	Fresh start	2 weeks ago
.gitignore	Fresh start	2 weeks ago
.metadata	Fresh start	2 weeks ago

Conclusion:
Deploying an E-commerce PWA to GitHub Pages provides a quick and reliable way to make the app available online. It simplifies distribution, testing, and user access without the need for traditional web hosting.

Experiment - 11

Name: **Shivpratik Hande**	Div-Roll no: **D15C-14**
DOP:	DOS:
Sign:	Grade:

AIM:

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

Theory:

Google Lighthouse is an open-source, automated tool used to audit web applications. It helps developers evaluate and improve the quality of Progressive Web Apps (PWAs) by generating detailed reports based on performance, accessibility, best practices, SEO, and PWA standards.
1. What is the Lighthouse PWA Audit?

Lighthouse checks whether a web app qualifies as a PWA by testing features like offline capability, HTTPS usage, responsive design, and “Add to Homescreen” readiness. The audit provides a score and suggestions for improvement. **2. How to Use Lighthouse**

Lighthouse can be accessed through:

- **Chrome DevTools (Audits tab)**
- **Command line (lighthouse <url>)**
- **Lighthouse CI or GitHub Actions**

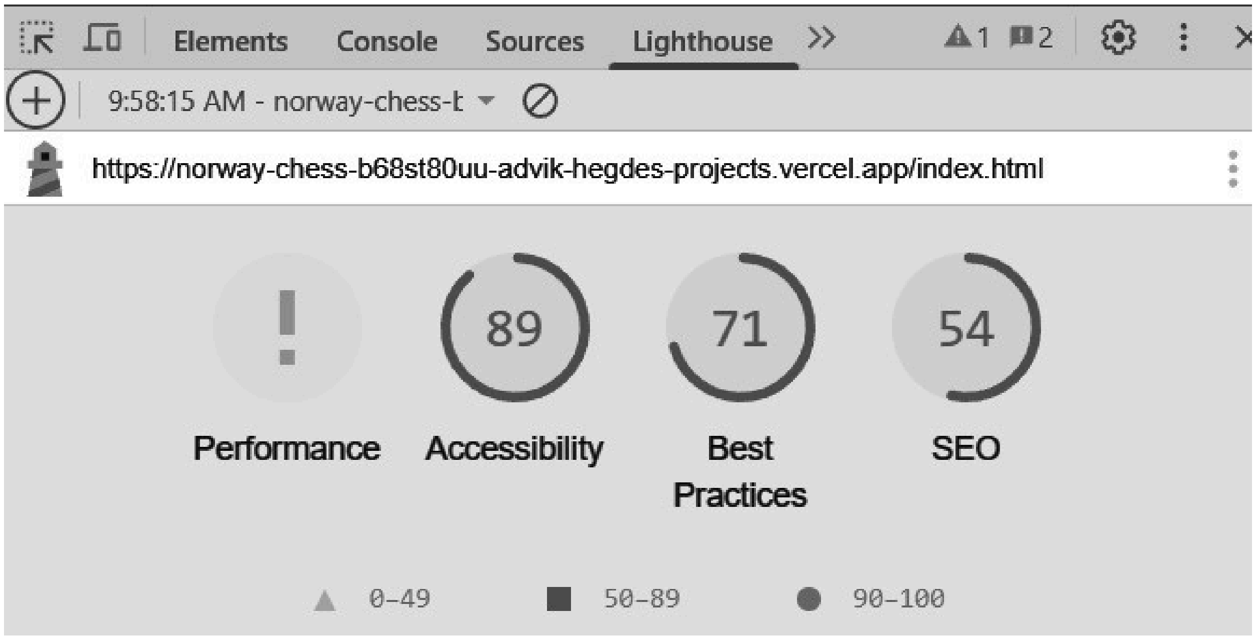
Users can run audits directly on their live or local PWA by generating a report within seconds.
3. Key Checks for PWA Audit

- App is served over HTTPS
- Service worker is registered and active
- Web App Manifest is valid
- Responsive and works offline

- Provides custom splash screen and install prompt

4. Benefits for E-commerce PWAs

- Ensures the app delivers a fast, reliable, and engaging experience
- Identifies and fixes performance and accessibility issues
- Helps in meeting PWA requirements for better user retention



Conclusion:
Using Google Lighthouse for PWA analysis ensures the E-commerce app meets modern web standards. It improves reliability, usability, and performance, leading to a better user experience and increased engagement.