

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

AIM: Installation and Configuration of Flutter Environment.

Description: Flutter is an open-source UI software development kit (SDK) created by Google. It's used to build natively compiled applications for mobile (iOS, Android), web, and desktop from a single codebase.

Key Features:

Cross-platform: Write once, run on multiple platforms.

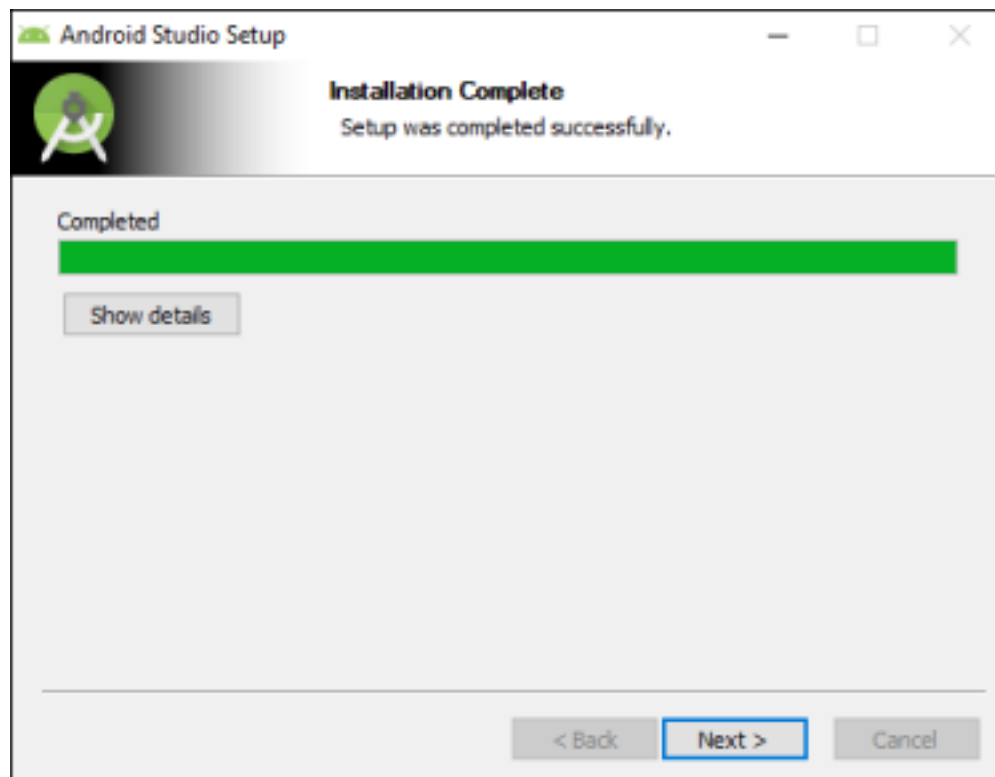
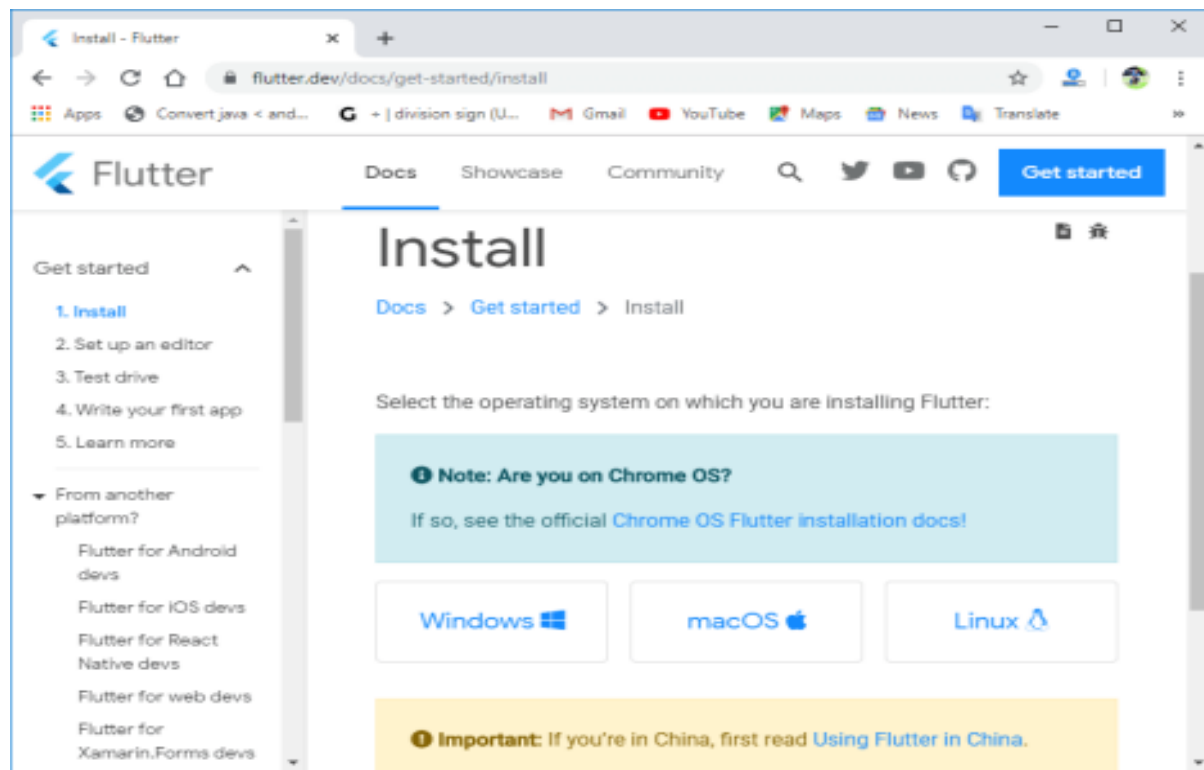
Dart Language: Flutter apps are written in Dart, a modern object-oriented language.

Hot Reload: Instantly see changes in the code without restarting the app.

Rich Widgets: Flutter offers a wide range of pre-designed, customizable widgets.

High Performance: Compiles to native ARM code for mobile, providing fast performance.

Screenshots:



Conclusion:

In this experiment, we successfully installed and set up Flutter for cross-platform app development. The process included installing the Flutter SDK, setting up Dart, and configuring an editor like VS Code or Android Studio. This setup allows developers to build high-performance apps for Android, iOS, web, and desktop using a single codebase. The installation also enabled features like hot reload, making development faster and more efficient.

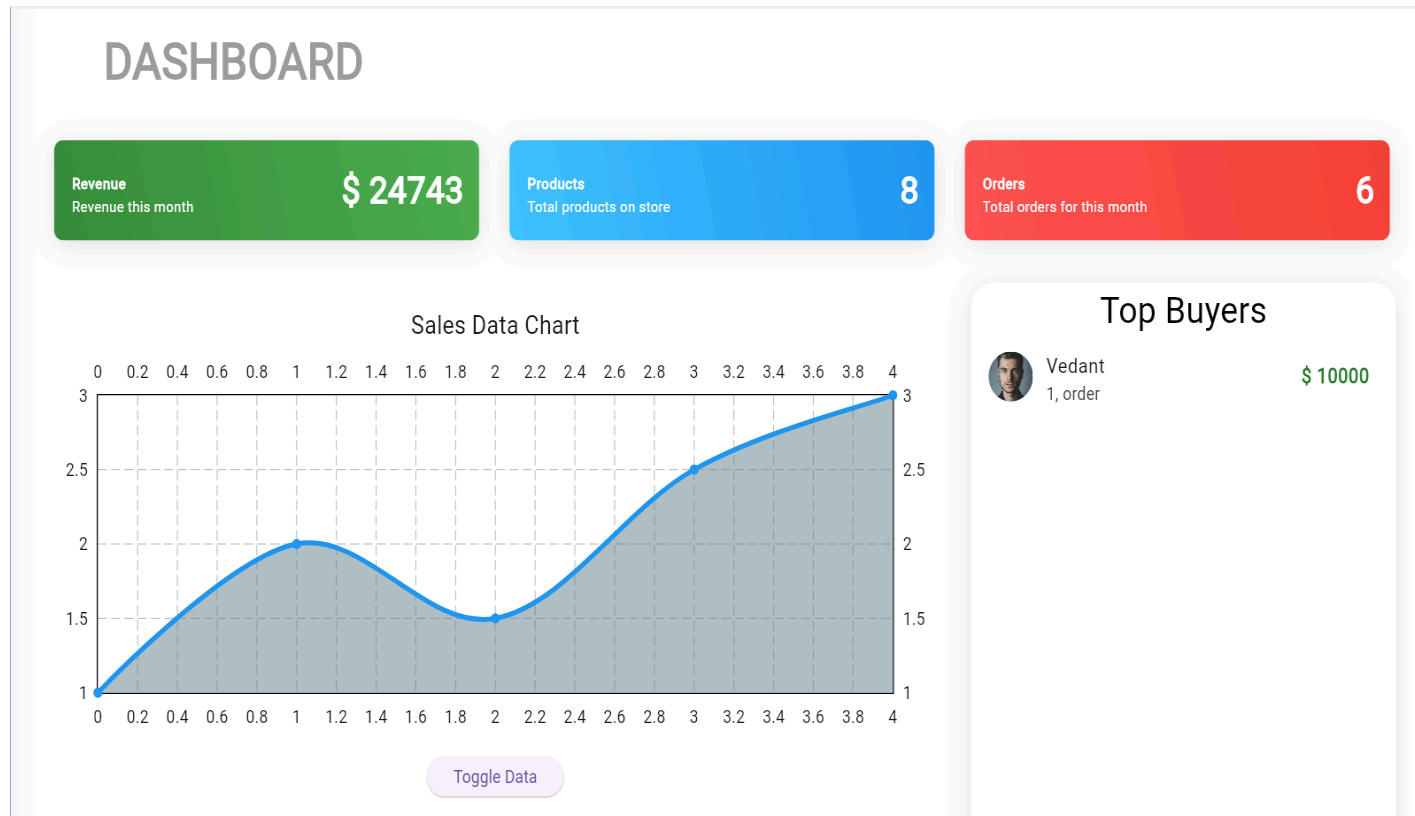
Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 2

AIM: To design Flutter UI by including common widgets.

Description: This code defines a HomePageDesktop widget using StatelessWidget since no state management is required. It imports Flutter's Material package along with custom widgets like CardsList, CustomText, PageHeader, SalesChart, and TopBuyerWidget. The UI is structured within a vertically scrollable ListView, starting with a "DASHBOARD" header. Below, CardsList displays key metrics, followed by a Row containing the SalesChart and TopBuyerWidget, spaced evenly. The chart is placed in a SizedBox with fixed height and dynamic width using MediaQuery. A custom-styled "Top Buyers" heading is displayed above the top buyers list.

Screenshot:



Conclusion: In this experiment, we designed a Flutter UI using common and custom widgets to build a responsive dashboard interface. By combining layout widgets like `ListView`, `Row`, and `SizedBox` with custom components such as `CardsList`, `SalesChart`, and `TopBuyerWidget`, we achieved a clean and structured design. This demonstrates Flutter's flexibility and efficiency in creating dynamic, cross-platform user interfaces.

Github Link:

https://github.com/VedantDhoke/flutter_store/blob/main/lib/pages/home/desktop.dart

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

Exp 3

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

Description:

1. Icons (FloatingActionButton)

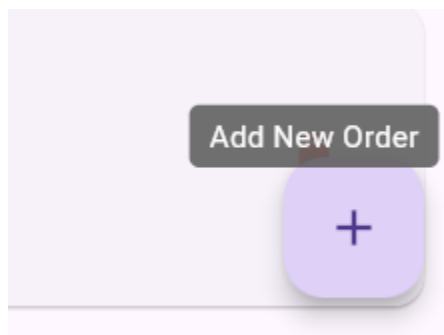
A FloatingActionButton creates a circular button at the screen's bottom-right, commonly used for primary actions. It triggers `_showAddOrderDialog(context)` on press to open a form or dialog. The Icons.add displays a plus (+) symbol, indicating "Add Order", and a tooltip "Add New Order" provides helpful context on hover or long press.

2. Images

`Image.asset("images/logo.png")` loads a local image from the app's assets folder. It's preferred over `Image.network()` when using locally stored images, as it doesn't rely on internet access and improves performance.

Screenshots:

1.



2.



Conclusion: In this experiment, we successfully included icons, images, and custom fonts in a Flutter app. We used `FloatingActionButton` with `Icons.add` to perform an action and display a tooltip, `Image.asset` to load local images efficiently, and integrated fonts for enhanced text styling. This demonstrates how Flutter simplifies UI customization and improves user experience through rich media and design elements.

Github Link:

https://github.com/VedantDhoke/flutter_store/blob/main/lib/widgets/navbar/navbar_logo.dart

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 4

AIM: To create an interactive Form using form widget

Description:

The form in this RegistrationPage consists of three input fields for user information:

Username Field -> Takes user input for the username. Uses a TextEditingController (authProvider.name) to manage the input.

Email Field -> Collects the user's email address. Managed using authProvider.email.

Password Field -> Uses a custom password input field (PasswordField). Connected to authProvider.password. It includes password visibility toggle for user convenience.

Form Behavior & Submission -> Users enter their details in the input fields. On clicking the "REGISTER" button. It calls authProvider.signUp(), which likely sends the data for authentication. If registration fails, a SnackBar (Registration failed!) is shown. On success, it clears the fields and navigates to the login page.

Screenshot:

The image shows a registration form with a blue border. The form is titled "REGISTRATION" in bold black text. Below the title are three input fields: "Username" with a person icon, "Email" with an envelope icon, and "Password" with a lock icon and a toggle eye icon. Below these fields is a blue button with the text "REGISTER" in white. At the bottom, there is a link that says "Already have an account? Sign in here..".

Conclusion: In this experiment, we created an interactive form using Flutter's Form widget. By combining input fields like TextFormField with validation logic and submission handling, we built a user-friendly and responsive form. This demonstrates Flutter's ability to manage form state effectively, making it easy to collect and validate user input.

Github Link:

https://github.com/VedantDhoke/flutter_store/blob/main/lib/pages/registration/registration.dart

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 5

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

Description :

1.GestureDetector -> Detects taps on the text "Sign up here."On tap, it navigates to the RegistrationPage using a global navigation service (locator<NavigationService>()).The text is styled using CustomText.

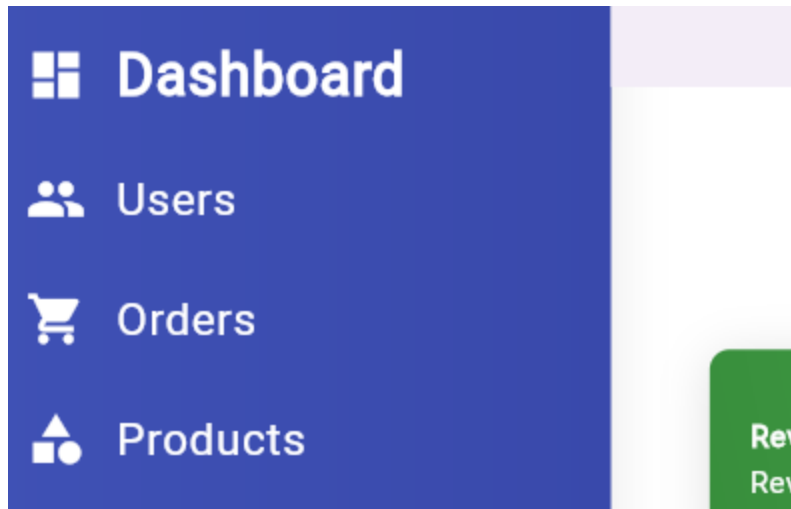
2.Route Management (generateRoute function) -> This function dynamically routes pages based on a given route name.

Checks the requested route (settings.name).Returns the corresponding page using `_getPageRoute`.

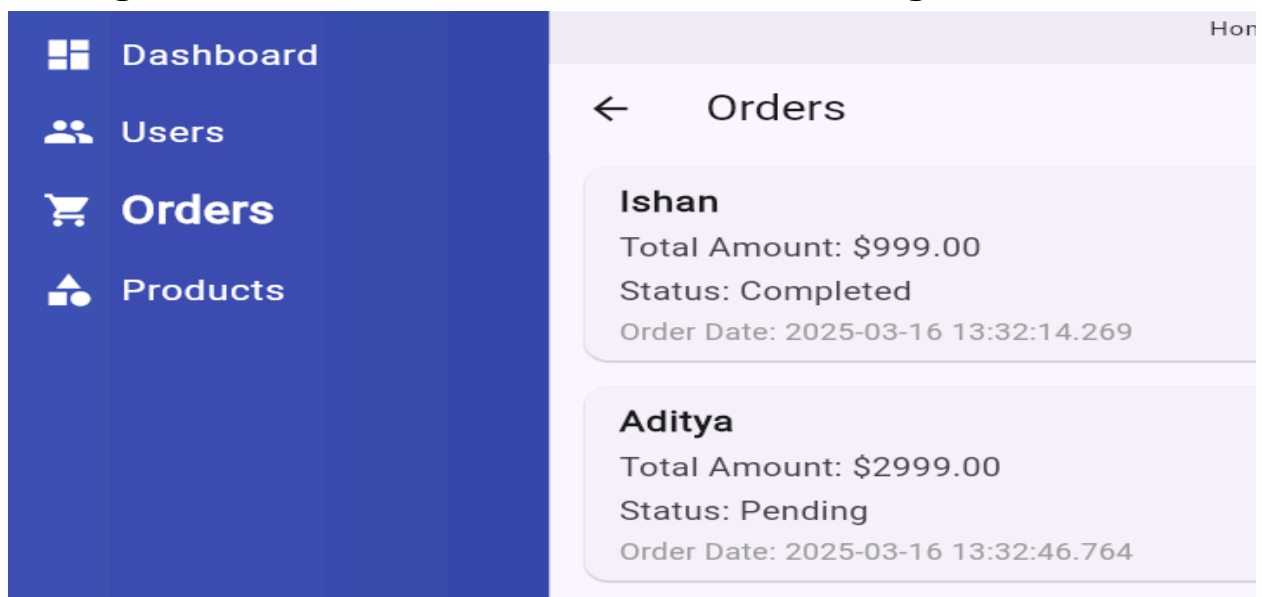
If the route is not found, it defaults to LoginPage.

Wraps a widget (page) inside a MaterialPageRoute. Ensures smooth navigation between pages.

Screenshots:



Clicking on the Orders button takes user to the Orders Page



Conclusion:

In this experiment, we implemented navigation, routing, and gesture detection in a Flutter app. Using Flutter's Navigator and route system, we enabled smooth transitions between screens. Gesture detectors were used to respond to user interactions like taps and swipes. This showcases how Flutter supports intuitive app navigation and interactive user experiences.

Github Link:

https://github.com/VedantDhoke/flutter_store/blob/main/lib/pages/login/login.dart

https://github.com/VedantDhoke/flutter_store/blob/main/lib/routing/router.dart

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 6

AIM: To connect Flutter UI with Firebase Database

Description:

Firebase is a Backend-as-a-Service (BaaS) platform by Google that provides tools like real-time databases, authentication, cloud storage, and more for building modern apps. It helps developers build secure, scalable apps quickly without managing backend infrastructure.

In my project Firebase is used to manage three collections (tables):

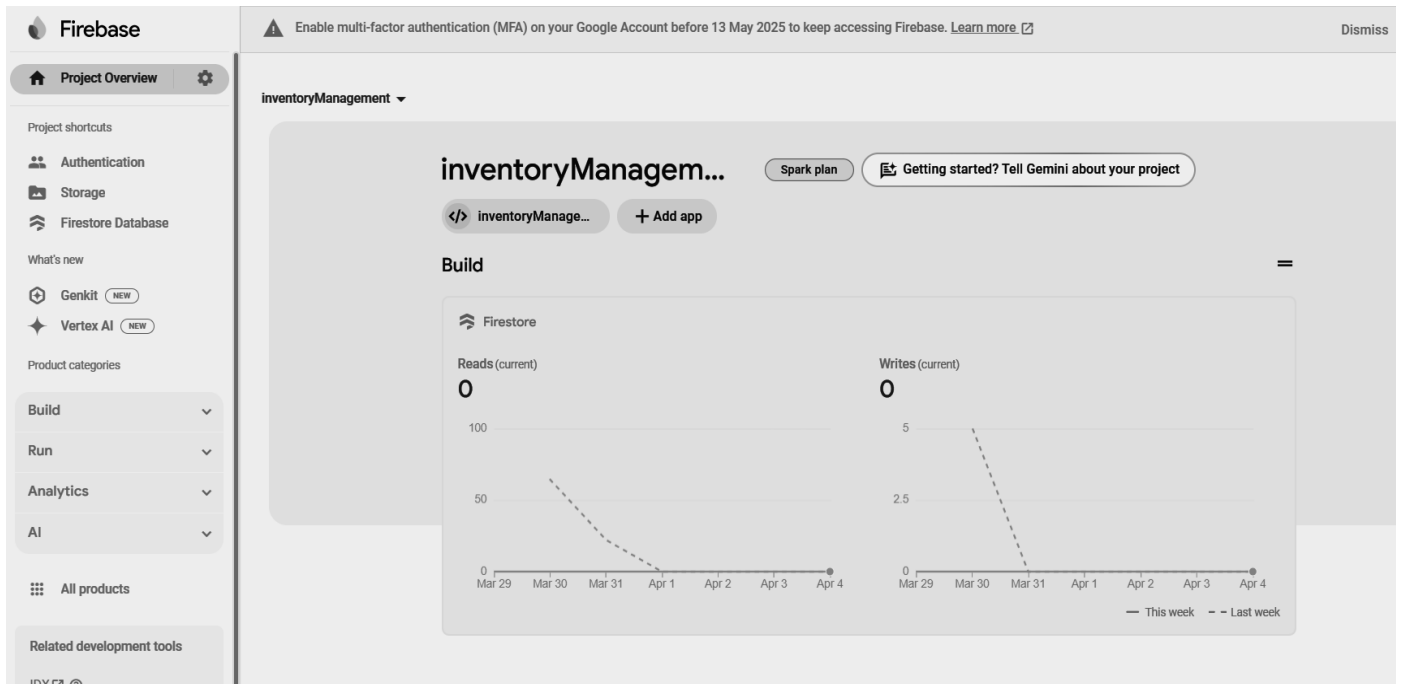
Orders – Stores order details placed by users.

Products – Contains product information such as name, price, and stock.

Users – Holds user profiles, authentication data, and other related info.

Firebase ensures real-time updates, secure data handling, and seamless integration with your Flutter app.

Screenshots:



Panel view Query builder

products > 1mmfpeAAYKtb.

More in Google Cloud

(default)	products	1mmfpeAAYKtb1lo4cocA
+ Start collection	+ Add document	+ Start collection
orders	1mmfpeAAYKtb1lo4cocA	+ Add field
products	90vwcARaRut11L0RU9DQ	name: "Sony Bravia 43" 4K LED"
users	ALONMxZIXbLBHWFRRaCg	price: 2199
	MNV0Dif15wMkx2v0jqQF	timestamp: 16 March 2025 at 13:36:10 UTC+5:30
	QjrL0pD0sr4qC0UejTZW	
	XjmAfLRNCnQ7ctUobyce	
	eWFnzVhVmHkDMjbcBr7f	
	jQSuzqL506QkSPn0tIN2	

Conclusion:

In this experiment, we successfully connected the Flutter UI with the Firebase Database. We integrated Firebase to store and manage data for users, products, and orders. This connection enabled real-time data updates and seamless backend support, demonstrating how Flutter and Firebase together provide a powerful solution for building dynamic and data-driven applications.

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

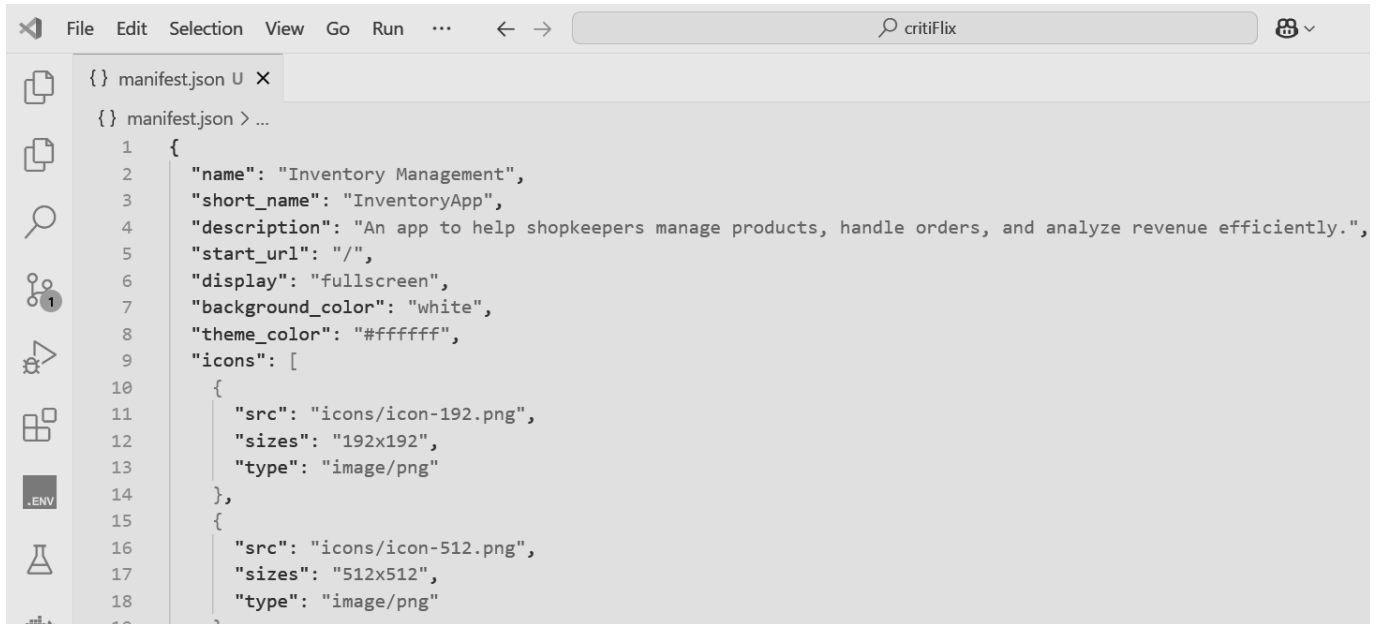
EXP 7

AIM: To write meta data of your Ecommerce PWA in a Web app manifest file

Description:

In this experiment, we created a Web App Manifest file to define the metadata for our E-commerce PWA. The manifest.json includes essential details such as the app's name, description, start URL, display mode, theme color, and icons. This setup allows the app to behave like a native application when installed, enhancing the user experience with features like fullscreen display and home screen icons.

Screenshots:



```
{  
  "name": "Inventory Management",  
  "short_name": "InventoryApp",  
  "description": "An app to help shopkeepers manage products, handle orders, and analyze revenue efficiently.",  
  "start_url": "/",  
  "display": "fullscreen",  
  "background_color": "white",  
  "theme_color": "#ffffff",  
  "icons": [  
    {  
      "src": "icons/icon-192.png",  
      "sizes": "192x192",  
      "type": "image/png"  
    },  
    {  
      "src": "icons/icon-512.png",  
      "sizes": "512x512",  
      "type": "image/png"  
    }  
  ]  
}
```

Conclusion:

In this experiment, we created a Web App Manifest file to define the metadata for our E-commerce PWA. The manifest.json includes essential details such as the app's name, description, start URL, display mode, theme color, and icons. This setup allows the app to behave like a native application when installed, enhancing the user experience with features like fullscreen display and home screen icons.

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

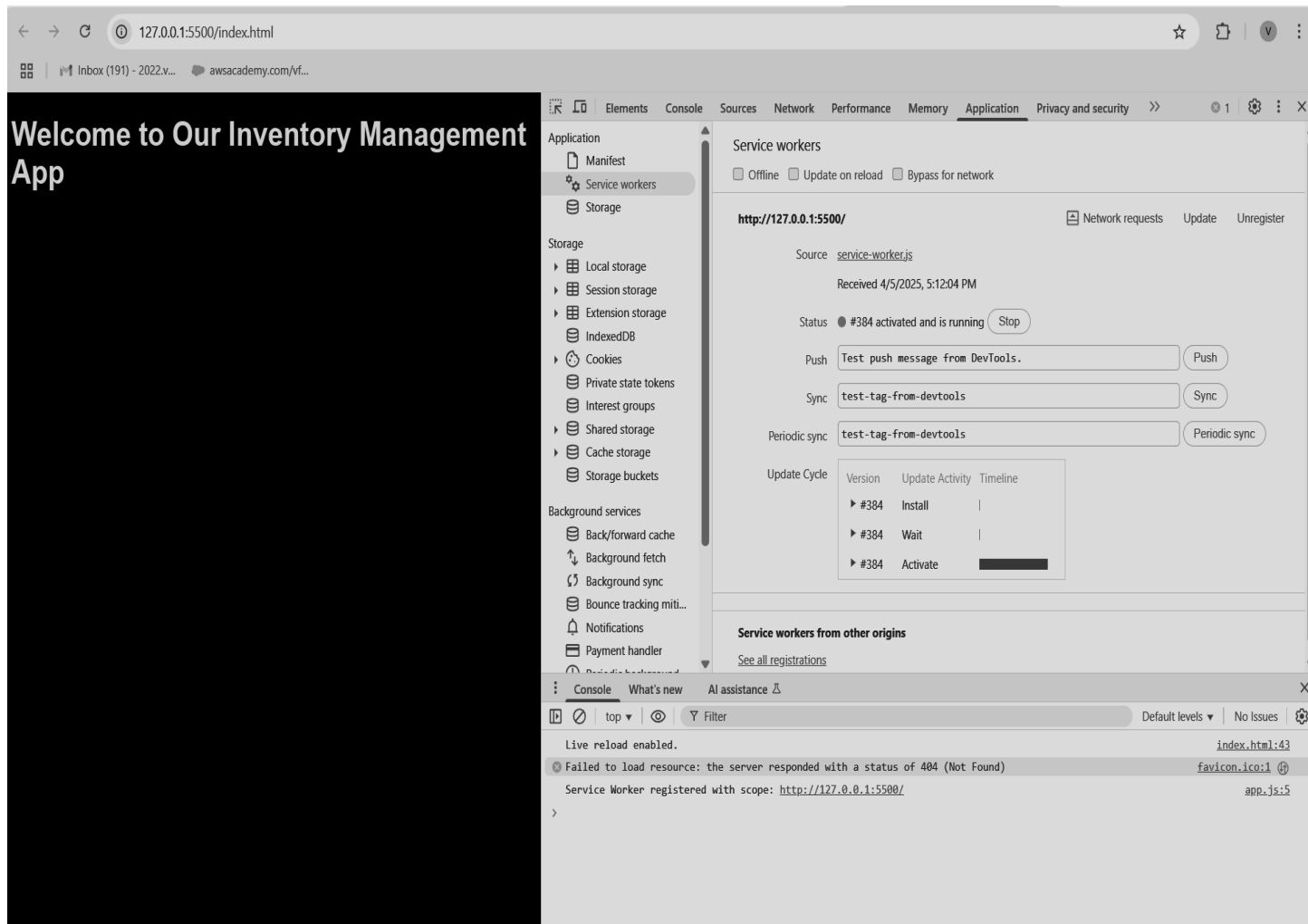
EXP 8

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.

Description:

A Service Worker is a background script that runs independently from the web page, enabling powerful features like offline support, background sync, and push notifications. It acts as a proxy between the web app and the network, allowing you to intercept and handle network requests, cache resources, and improve performance. Service workers only run over HTTPS (or localhost for development) and follow a lifecycle of registration, installation, and activation.

Screenshot:



Conclusion:

In this experiment, we successfully coded and registered a service worker for our Inventory Management PWA. We completed the service worker lifecycle—registration, installation, and activation—to enable features like offline access and caching. This enhances app performance, reliability, and user experience by allowing the app to function even without an internet connection.

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 9

AIM: : To implement Service worker events like fetch, sync and push for Inventory Management PWA.

Description:

A Service Worker is a script that runs in the background of a browser and allows developers to intercept and manage network requests, enable offline functionality, handle background sync, and display push notifications. Service workers play a critical role in developing Progressive Web Apps (PWAs) by enhancing performance and reliability, especially in unreliable network conditions.

Key Events:

Fetch Event:

Allows the service worker to intercept network requests. It enables caching strategies like cache-first (check cache before going to network) and network-first (try network first, fallback to cache). This improves loading times and supports offline access.

Sync Event:

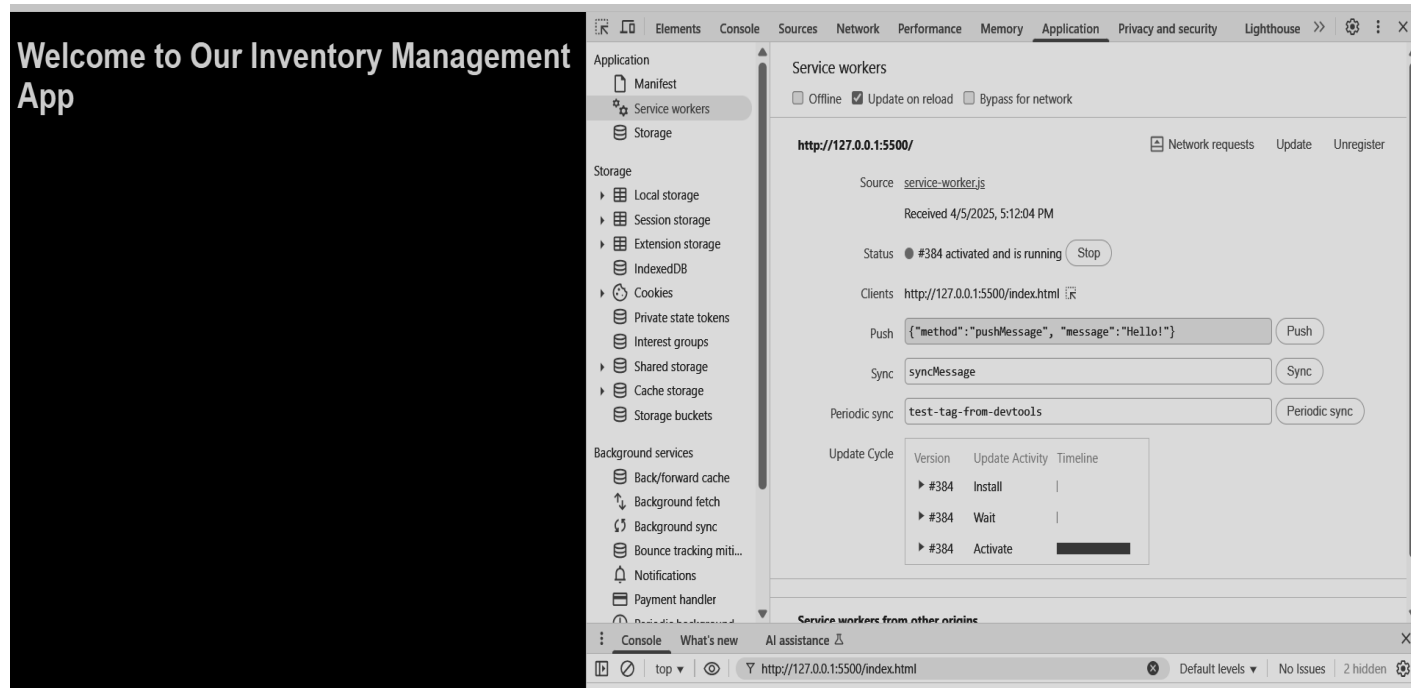
Background Sync allows processes (like sending data) to be deferred until the user has a stable internet connection. For example, if the user updates inventory data offline, the service worker can sync it automatically once back online.

Push Event:

Enables the application to receive push notifications even when it is not open. This keeps users informed with real-time updates, like low stock alerts or new order confirmations.

These service worker events significantly improve user experience by providing offline functionality, ensuring data consistency, and increasing engagement through timely notifications.

Screenshot:



Conclusion:

In this experiment, we successfully implemented Service Worker events like fetch, sync, and push in an Inventory Management PWA. The fetch event helped manage network requests using cache-first and network-first strategies, enabling offline access to cached resources. The sync event allowed background synchronization when the internet was unavailable, ensuring reliable data updates. Finally, the push event enabled real-time notifications to users. This implementation enhanced the PWA's offline capabilities, responsiveness, and user engagement.

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

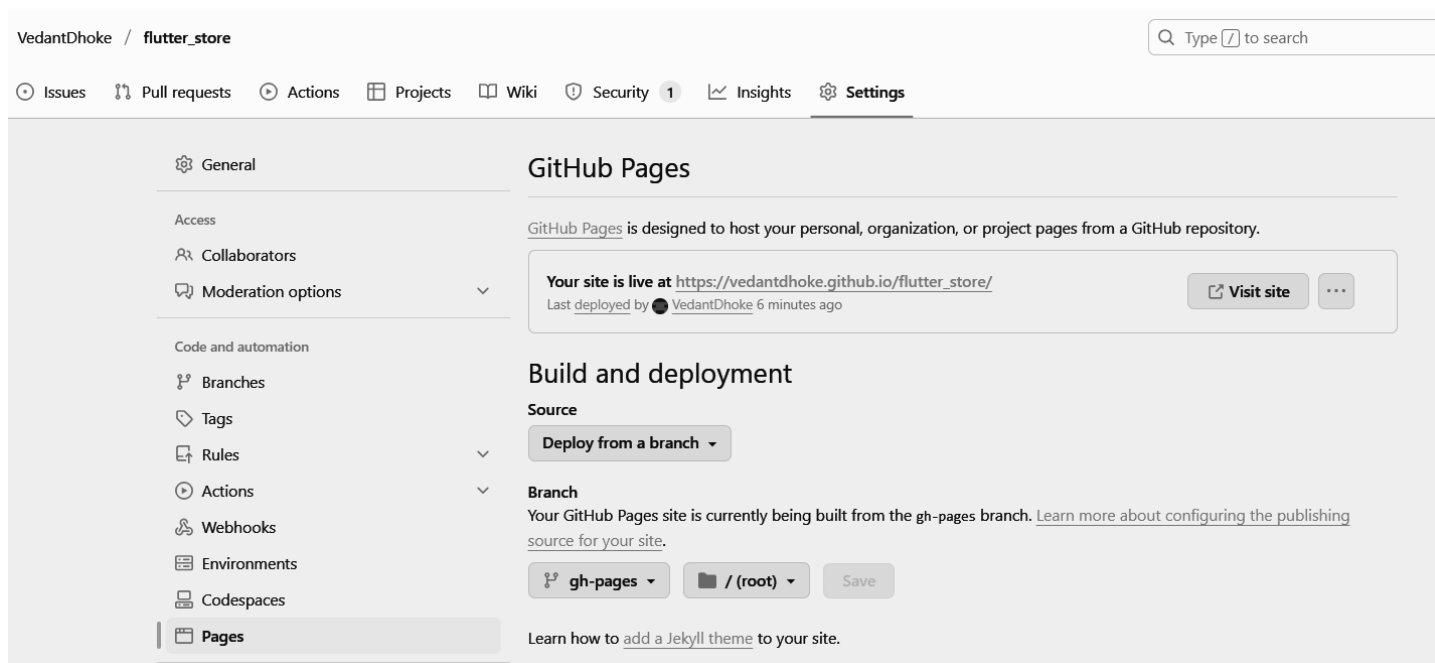
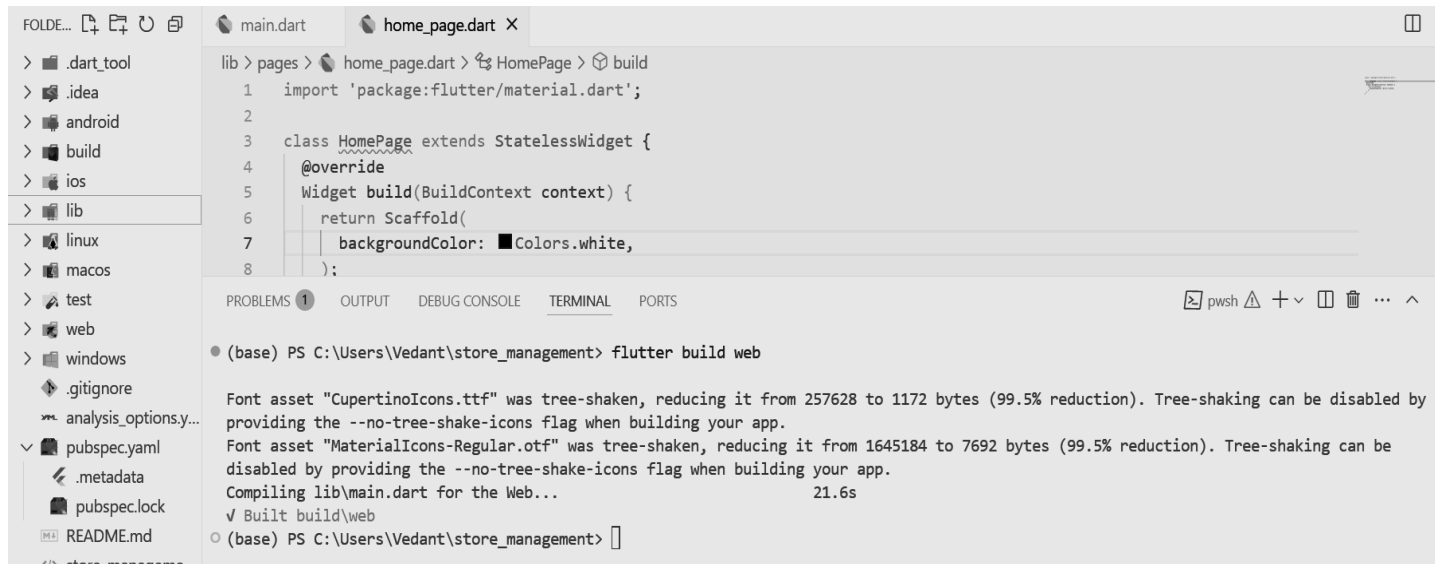
EXP 10

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

Description:

GitHub Pages is a free hosting service that allows developers to deploy static web pages directly from their GitHub repositories. It simplifies the deployment process by enabling instant publishing—just push code to the gh-pages branch, and the site goes live. It supports Jekyll for static site generation, offers custom domain integration through a simple CNAME file, and is ideal for hosting portfolios, documentation, or project-based websites. Major companies like Lyft and HubSpot utilize GitHub Pages for its simplicity and fast setup.

Screenshot:



Conclusion:

This experiment helped us understand how to deploy a Flutter-based E-commerce PWA to GitHub Pages. GitHub Pages is an efficient and cost-effective option for hosting static sites, especially for projects already managed on GitHub. Compared to Firebase, GitHub Pages is simpler for static deployments but lacks real-time backend support. Our deployment was successful, and we verified that the application is live and accessible via the provided GitHub Pages link.

Name	Vedant Dhoke
Class/Roll No	D15C / 9
Subject	MAD and PWA Lab
DOP	
DOS	
Sign	

EXP 11

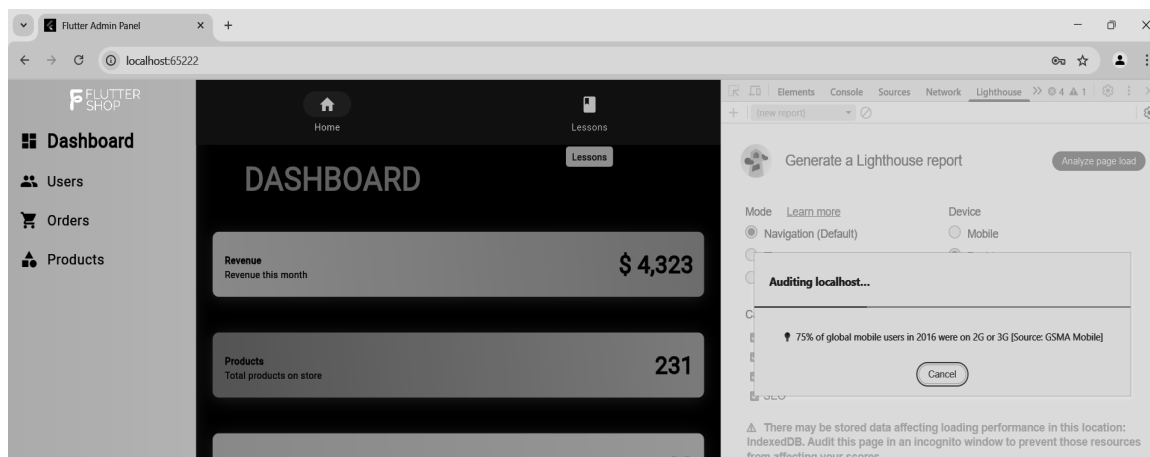
AIM: To use google Lighthouse PWA Analysis Tool to test the PWA functioning.

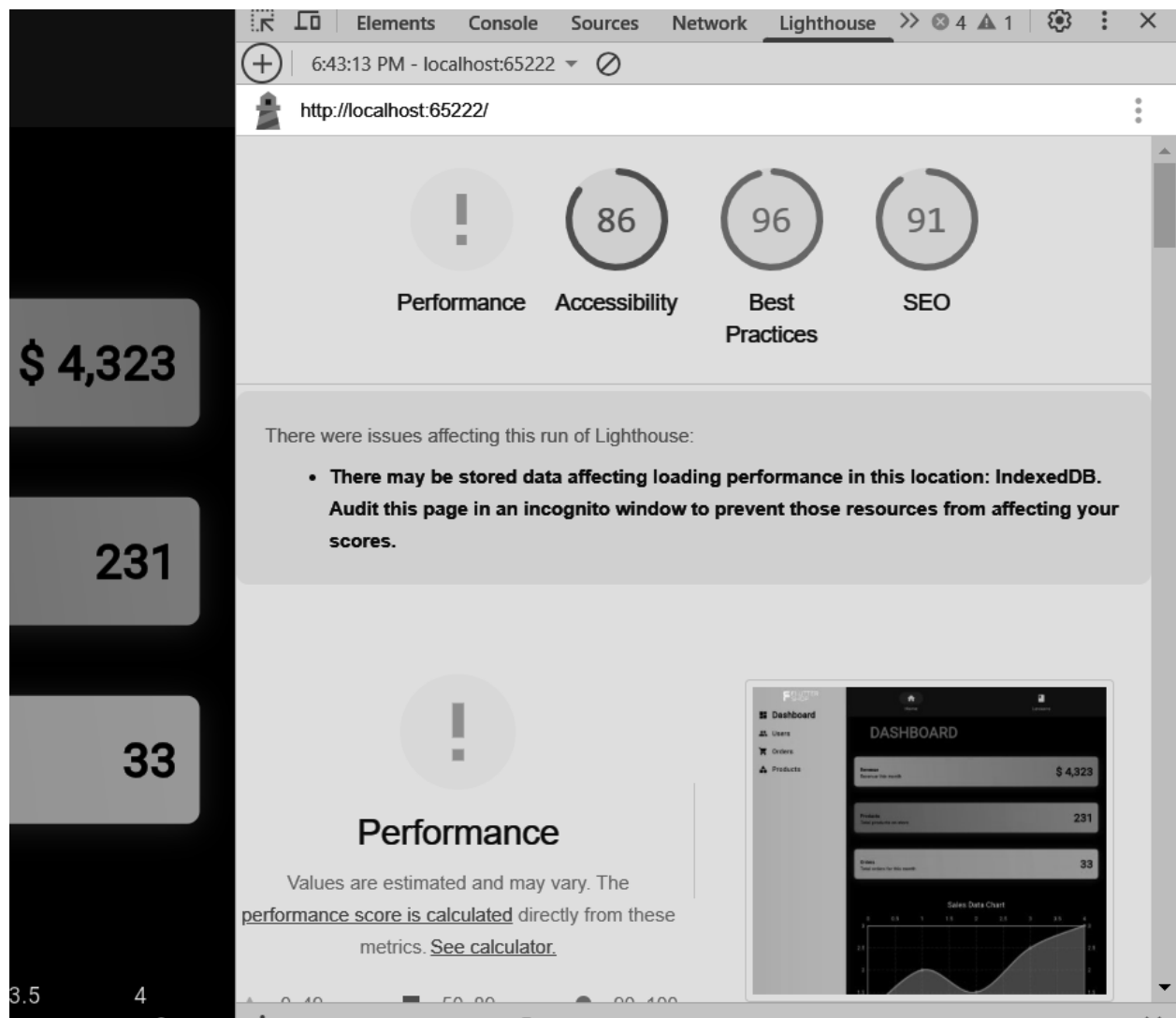
Description:

Google Lighthouse : Google Lighthouse is a tool that lets you audit your web application based on a number of parameters including (but not limited to) performance, based on a number of metrics, mobile compatibility, Progressive Web App (PWA) implementations, etc. All you have to do is run it on a page or pass it a URL, sit back for a couple of minutes and get a very elaborate report, not much short of one that a professional auditor would have compiled in about a week.

The best part is that you have to set up almost nothing to get started. Let's begin by looking at some of the top features and audit criteria used by Lighthouse.

Screenshot:





Conclusion: In this experiment, we effectively used Google Lighthouse to evaluate the PWA functionality of our ecommerce application. The tool provided a detailed audit covering key aspects such as Performance, Accessibility, Best Practices, and Progressive Web App compliance.

The Lighthouse analysis helped us understand how well our app meets modern web standards and provided insights to enhance its speed, reliability, and overall user experience.