PROJECT REPORT ON

HATIM VEHICLE IDENTIFICATION APP

SUBMITTED BY:

LALBIAKHLUZUALA SAMTE..... ROLL NO. 2023BCA003

NICHOLAS BEICHATLAISA .....ROLL NO. 2023BCA005

Under the guidance of:

Mr. H. Lalruatkima

Assistant Professor

Bachelor of Computer Application

(HATIM)

MIZORAM UNIVERSITY, TANHRIL: AIZAWL

2023

HIGHER AND TECHNICAL INSTITUTE, MIZORAM
KAWMZAWL, LUNGLEI – 796701

CERTIFICATE

This is to certify that LALBIAKHLUZUALA SAMTE and NICHOLAS BEICHATLAISA has fully completed the project entitled, "HATIM VEHICLE IDENTIFICATION APP" in order to meet the requirement of the Mizoram University for the VI Semester Bachelor of Computer Application in the year 2023 (January - May). It is to certify that all the corrections/suggestions indicated for internal assessment has been incorporated in the project. The project report has been approved as it satisfies the academic requirements in respects of the project work prescribed for the BCA Course.

(H. LALRUATKIMA)	(VUANSANGA VANCHHAWNG)	(K. LALMUANPUIA)
Project guide	Principal	Head of Department
Dept. of Computer Science	HATIM	Dept. of Computer Science
		H.O.D. Department of Computer Science Higher & Technical Institute,Mizoram Lunglei : Mizoram

Name of Examiner

LALREMPUIA SAIKO

Signature with date

ACKNOWLEDGEMENT

First, we would like to give thanks to God for guiding us throughout the process of our project, giving us with good health so that we are able to get this far.

We thank to our Principal Mr. Vuansanga Vanchhawng. We thank him for the support, encouragement and permission, which we enjoy freely in order to accomplish our project, and in this time of Pandemic situation.

Our heartfelt gratitude goes to Mr. H. Lalruatkima, Project Guide and Mr. K. Lalmuanpuia, Head of Department in Department of Computer Science, Higher and Technical Institute, Mizoram for their continual advice, backup, confidence, patience and their encouraging words which gives us courage and confidence without which we would be nowhere.

We thank all the people in the Institution for being always available and helpful over the semester.

Last but not the least; we thank our parents for their moral support and encouragement.

LALBIAKHLUZUALA SAMTE
NICHOLAS BEICHATLAISA

VI Semester, BCA
Higher and Technical Institute, Mizoram

ABSTRACT

The HATIM Vehicle Identification Application project is a Flutter application that utilizes the camera and Google ML Kit's text recognition capabilities to scan vehicle information from images. The application allows users to capture an image of a vehicle number plate and extract relevant details such as the vehicle number and owner's name. The extracted information is then used to query a Firestore database containing vehicle information.

The main features of the application include:

- Camera Integration: The application utilizes the device's camera to capture images for text recognition.
- Text Recognition: Google ML Kit's text recognition API is employed to extract text from the captured images.
- Indian State Vehicle Code Extraction: The application searches for a valid Indian state vehicle code within the recognized text to identify the state of the vehicle.
- Firestore Database Integration: The extracted vehicle number and owner's name are used to query a Firestore database for further details about the vehicle.
- User Interaction: Users can edit the recognized text if necessary, and the application dynamically updates the search results based on the edited text.
- UI Elements: The user interface includes a futuristic viewfinder, camera control buttons, and a scan button for capturing the vehicle number plate image.

The Vehicle Info project aims to provide a convenient way for users to retrieve information about vehicles by simply scanning their number plates. It can be utilized for various purposes such as verifying vehicle ownership, checking vehicle details, and facilitating quick access to relevant information.

DECLARATION

We do hereby that the project entitled “HATIM Vehicle Identification Application” is not submitted to any other university or institution for the award of any degree, diploma of fellowship or published any time before. This project is prepared under the guidance of our project guide Mr. H. Lalruatkima which forms our partial fulfilment of the requirements for the three years Bachelor Degree in Computer Applications of Mizoram University.

LIST OF FIGURES

NOMENCLATURE AND ABBREVIATIONS

1. **API** : Application Programming Interface
2. **SDK** : Software Development Kit
3. **UI** : User Interface
4. **JSON** : JavaScript Object Notation
5. **HTML** : Hypertext Markup Language
6. **HTTPS** : Hypertext Transfer Protocol Secure
7. **NoSQL** : A flexible, scalable database approach for handling unstructured data.
8. **Firebase** : A powerful, integrated platform for developing and managing web and mobile applications.
9. **Flutter** : A versatile framework for building cross-platform mobile applications.
10. **SafeArea** : A Flutter widget that automatically insets its child to avoid intrusions by the operating system's UI elements.
11. **Cloud Firestore** : A NoSQL document-oriented database service provided by Firebase, offering real-time syncing and offline support for mobile and web applications.
12. **Google ML Kit** : Machine Learning SDK by Google for integrating powerful ML capabilities into mobile and web applications.

CONTENTS

1. INTRODUCTION

CONTENT

- **Introduction**
- **Overview of the Project**
- **Objective of the Project**

INTRODUCTION

The HATIM Vehicle Identification Application is a cutting-edge Flutter app designed to simplify the process of retrieving information about vehicles. It is an app specifically designed for use by HATIM as the name suggests. Leveraging the power of Google ML Kit's text recognition technology and the integration of a camera, this app allows users to capture images of vehicle number plates and extract essential details such as the vehicle number and owner's name. By utilizing these extracted details, the app seamlessly queries a Firestore database to provide comprehensive information about the vehicle.

With its user-friendly interface and efficient functionality, Vehicle Info aims to revolutionize the way individuals access and verify vehicle information, offering convenience and reliability in a single, intuitive application.

Overview of the Project

- Image-Based Vehicle Info Retrieval: Develop an app using image recognition to extract vehicle details from number plates.
- Google ML Kit Integration: Utilize Google ML Kit for accurate text recognition from captured images.
- Firestore Database Integration: Seamlessly store and retrieve comprehensive vehicle info from Firestore.
- User-Friendly Interface: Provide an intuitive interface with a futuristic viewfinder and scan functionality.
- Versatile Applications: Serve law enforcement, individuals, and organizations for quick and reliable vehicle info retrieval.

Objective of the Project

The objective of the Vehicle Info project is to create a robust and user-friendly Flutter application that utilizes image recognition and text extraction techniques to provide seamless access to vehicle information.

The project aims to simplify the process of retrieving details from vehicle number plates by leveraging Google ML Kit's text recognition capabilities and integrating them with a Firestore database.

The primary goal is to offer users a convenient and efficient way to obtain comprehensive information about vehicles by simply capturing images, thereby enhancing convenience, accuracy, and accessibility in accessing and verifying vehicle details.

2. SYSTEM ANALYSIS

CONTENT

- **Need for System**
- **Feasibility Study**
- **Hardware Requirement**
- **Software Requirement**
- **Entity-Relationship Diagram**
- **Data Flow Diagram**

SYSTEM ANALYSIS

Need for System

The Vehicle Info system meets the demand for a reliable and convenient method of accessing vehicle information. With the growing number of vehicles, there is a need for quick and accurate retrieval of details such as vehicle numbers and ownership information. Manual entry and paper-based documentation are time-consuming and prone to errors. The Vehicle Info system provides an automated solution using image recognition and text extraction technologies, enabling users to efficiently retrieve and verify vehicle information. It caters to individuals, law enforcement agencies, and organizations seeking a streamlined approach to access comprehensive vehicle details in a digitized world.

Feasibility Study

1. Technical Feasibility: The project utilizes proven technologies like Flutter, Google ML Kit, and Firestore, ensuring a stable and reliable foundation for development.
2. Economic Feasibility: The project leverages cost-effective open-source tools and scalable database solutions, minimizing development expenses and accommodating future growth.
3. Operational Feasibility: The user-friendly interface requires minimal training, allowing users to easily capture images, extract text, and query the database.
4. Time Feasibility: The project timeline is achievable with efficient planning and coordination, leveraging well-documented frameworks and libraries.
5. Legal and Ethical Feasibility: The project complies with privacy regulations, safeguarding user data and preventing unauthorized access to sensitive information.

Hardware Requirement

1. Mobile Device : A smartphone or tablet running Android.
2. Camera : Minimum resolution for clear image capture.
3. Storage : Adequate internal storage or expandable option. (atleast 200MB of space available)
4. Processing Power : Capable processor for smooth app performance. (256MB RAM or above)
5. Network Connectivity : Reliable internet connection for database access.

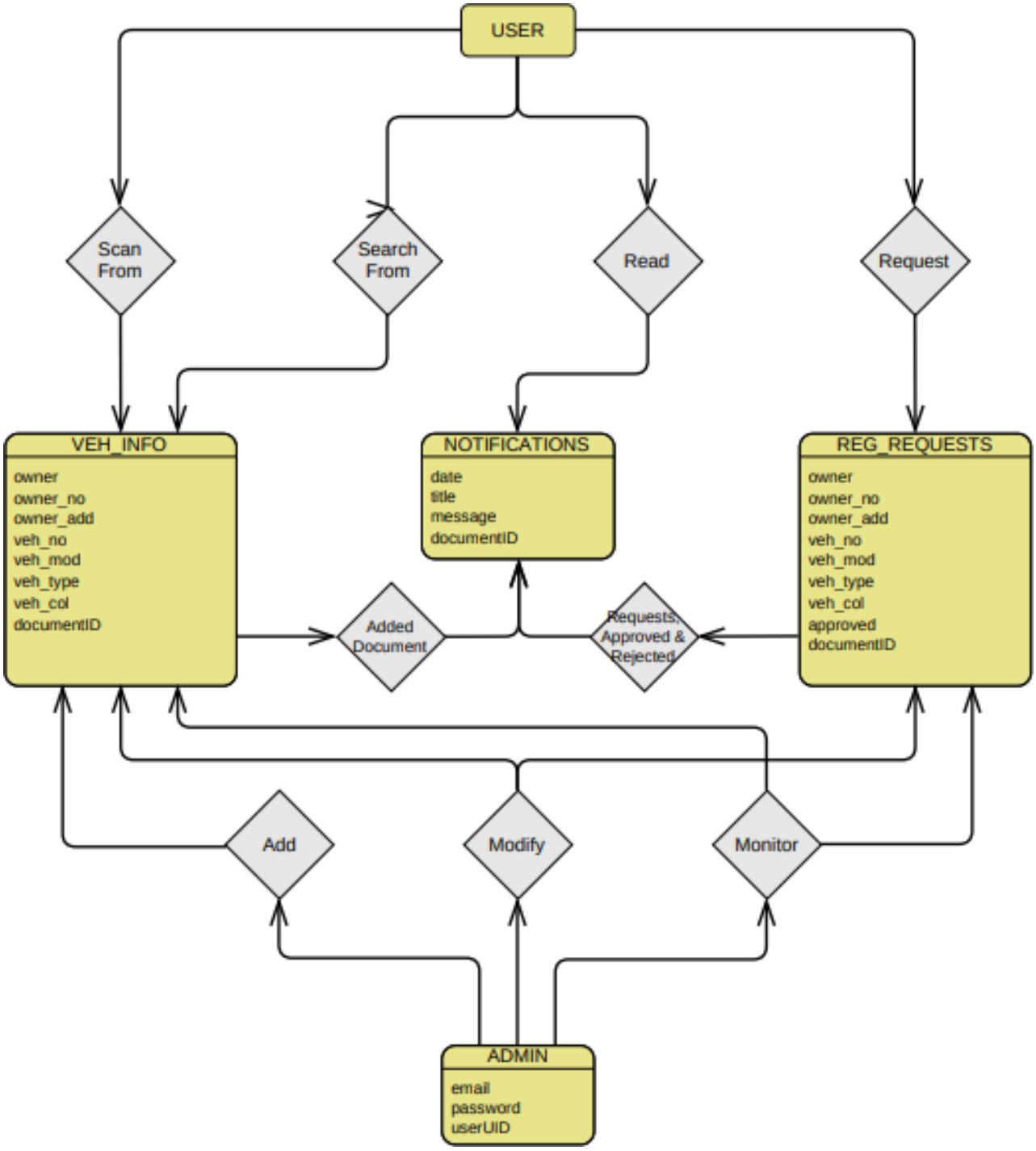
Note: Specific requirements may vary based on platform and features.

Software Requirement

1. Operating System : Android 8.0 Oreo or later.
2. Flutter Framework : Flutter SDK.
3. Camera API : Support for Flutter Camera package.
4. Google ML Kit : ML Kit Text Recognition API support.
5. Firebase : Firebase SDK for cloud-based data storage.

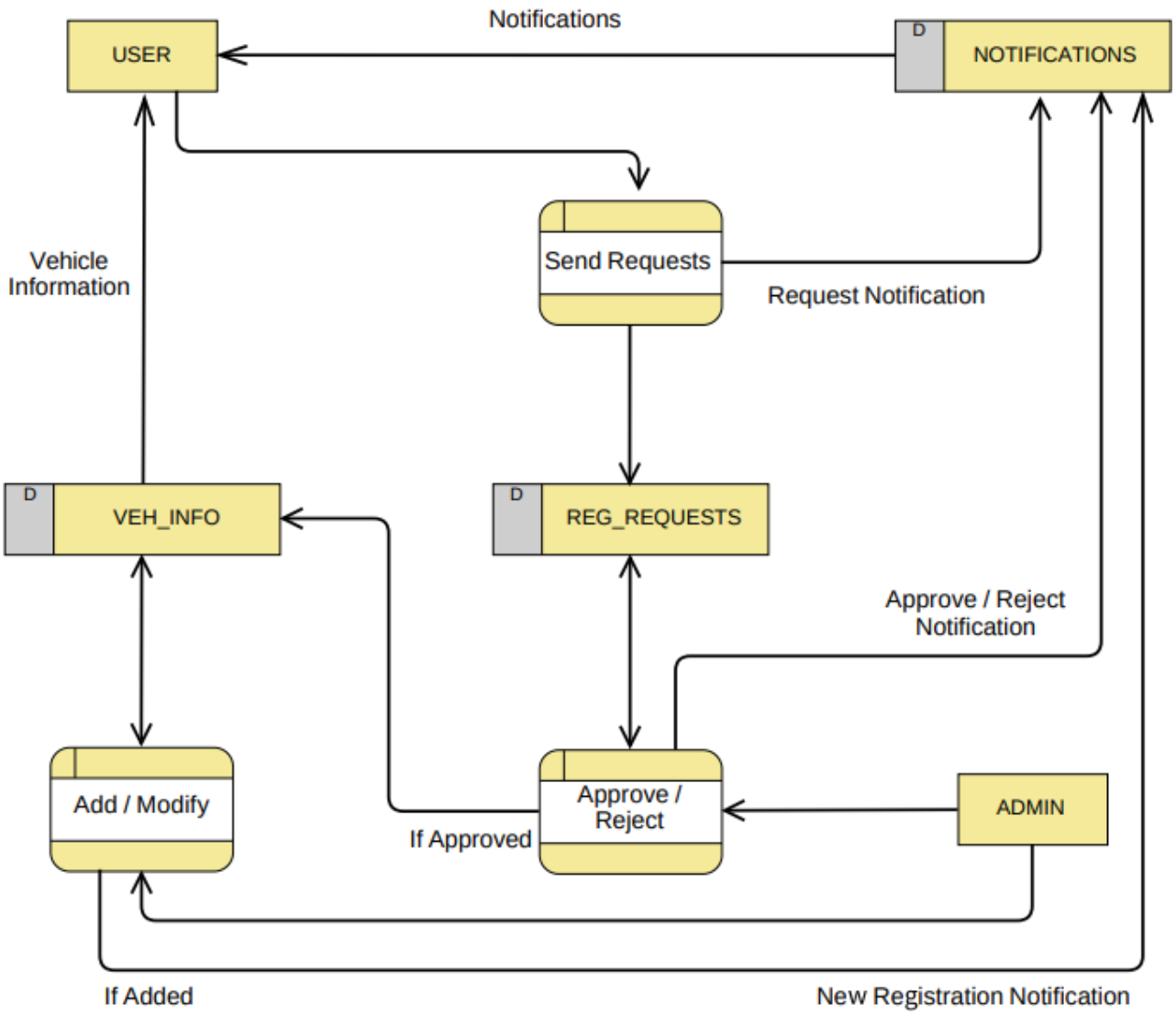
Note: Requirements may vary based on specific versions and dependencies.

Entity-Relationship Diagram



(Fig 1.1 – Entity Relationship Diagram)

Data Flow Diagram



(Fig 1.2 – Data Flow Diagram)

3. SYSTEM DESIGN

CONTENT

- **Program Structure**
- **Modularization Details**
- **User Interface Design**
  - **Splash Screen**
  - **Home**
  - **Scanner**
  - **Edit Screen**
  - **Result Screen**
  - **Manual Search Screen**
  - **Detailed Information Page**
  - **Official Sarathi Parivihan Website**
  - **New Registration Request**
  - **Rules and Regulations**
  - **About Page**
  - **Admin Login Page**
  - **Admin Panel**
  - **Registered Documents Page**
  - **New Document Registration Page**
  - **Registration Requests Page**
  - **Registration Request Details Page**

SYSTEM DESIGN

Program Structure

The program structure in the provided code follows a typical Flutter application structure. Here is an overview of the program structure:

1. Main Entry Point: The **main()** function initializes the Flutter app, ensures Firebase initialization, and runs the **App** widget.
2. App Widget: The **App** widget is the root widget of the application. It configures the app's theme, title, and initial route.
3. SplashScreen: The **SplashScreen** widget is displayed as the initial screen while the app is loading. It could be used for showing a logo or any other introductory content.
4. HomeScreen: The **HomeScreen** widget represents the main screen of the application. It handles the camera functionality, text recognition, and user interactions related to capturing vehicle information.
5. Scanner: The **Scanner** scans and extract the detected texts from the image captured and sent the text to the ResultScreen where information about the scanned document are displayed.
6. ResultScreen: The **ResultScreen** widget displays the results of the text recognition process. It presents the retrieved data from Firestore based on the scanned vehicle information.
7. ManualSearch: The **ManualSearch** widget displays the documents stored in the Firebase in a list, which navigates to the Detailed Information page when pressing the document.
8. DetailedInformation: The **DetailedInformation** widget displays the details of the selected document displayed in the ManualSearch screen by fetching the data stored in the Firebase.

9. NewRegistrationUser: The **NewRegsitrationUser** widget displays a list of textfields in which the user can request for registration to the Admin.

10. AdminPanel: The **AdminPanel** widget is the page in which only Admins can have access to it. The page navigates to three pages i.e., RegisteredDocuments, NewRegistrationAdmin and RegistrationRequests pages.

11. RegisteredDocuments: The **RegisteredDocuments** widget displays the list of documents stored in the Firebase, allowing Admin to edit or delete existing document from the database.

12. NewRegistrationAdmin: The **NewRegistrationAdmin** widget is located inside the AdminPanel. This page allows Admins to register document directly to the Firebase.

13. RegistrationRequests: The **RegistrationRequests** widget displays the registration requests sent by the user from the User end. It allows the Admin to Approve or Reject requests.

14. Firebase Firestore: The code interacts with the Firebase Firestore database to retrieve and store vehicle information. It includes fetching data based on the scanned vehicle details and updating data if needed.

15. Other Dependencies: The code uses various dependencies like **camera**, **cloud_firestore**, **google_mlkit_text_recognition**, and **permission_handler** to enable camera access, text recognition, and database functionalities.

Modularization Details

The modularization details in the provided code can be summarized as follows:

1. File Organization: The code is organized into separate files for different screens and components, promoting code separation and manageability.
2. Widget Composition: Flutter's widget composition is used to create reusable and self-contained components within the application.
3. Separation of Concerns: Related functionalities are separated into different classes or methods, enhancing code organization and maintainability.
4. Dependency Injection: External libraries and services are integrated using dependency injection, allowing for modular integration and code reusability.
5. Encapsulation of Firebase Functionality: Firebase Firestore database functionalities are encapsulated in a method, providing a modular interface for data retrieval.

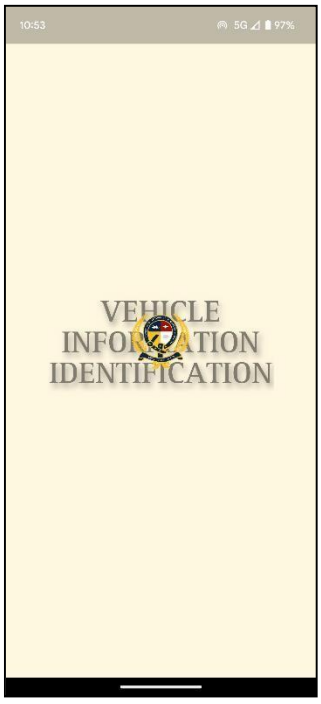
These modularization techniques improve code reusability, separation of concerns, and overall maintainability of the application.

User Interface Design

The user interface (UI) of the Vehicle Info app is designed with a user-centric approach, aiming to provide a seamless and intuitive experience. The UI features a clean and visually appealing design, making it easy for users to navigate and interact with the app. The futuristic viewfinder enhances the scanning process, allowing users to capture accurate images of vehicle number plates. The interface also includes user-friendly camera control buttons, enabling effortless switching between flash modes and cameras. With a focus on simplicity and efficiency, the UI ensures that users can quickly access and retrieve vehicle information with just a few taps, enhancing the overall usability of the app.

Splash Screen –

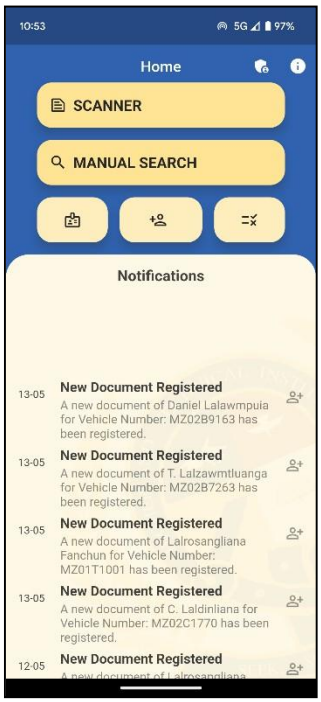
The Splash Screen is the first screen that will appear when you open the application. It displays a simple logo with minimal design and a simple background colourway.



(Fig. 2.1 – Splash Screen)

Home –

The **HomeScreen** widget consists of an app bar, background image, and several buttons for different functionalities. There is also a section for displaying notifications.



(Fig. 2.2 – Home)

The app bar is displayed at the top and includes a title "Home" and two icons: one for accessing the AdminPanel settings and another for displaying information about the app.

The background image adds a visual appeal to the screen and enhances the overall aesthetics.

There are three main buttons:

1. "SCANNER" Button: This button, represented by an icon and text, allows the user to navigate to the **Scanner** when pressed.
2. "MANUAL SEARCH" Button: Similar to the first button, this button enables the user to navigate to the **Manual Search Screen** when pressed.
3. Three Smaller Buttons: These buttons are displayed in a row and provide quick access to external resources. The first button directs the user to the official Parivahan website for applying **Learner's Driving License** and other relevant activities, the second button navigates to the **New Registration Page**, and the third button navigates to the **Rules and Regulations** screen.

Lastly, there is a section dedicated to displaying notifications. It includes a title "Notifications" and a list that fetches and displays a list of notifications. Each notification item consists of a date, title, message, and a trailing icon based on the type of notification.

Scanner –

This is the Scanner that allows users to scan for the vehicle number. The scanned characters will be copied to the next page i.e., Edit Screen where the scanned text will be able to be edited in case of scanning/spelling mistake.

(Fig. 2.3 – Scanner)

After the Edit Screen is the Result Screen; this page will display the actual result of the scanned number by displaying detailed information of the vehicle being scanned.

The camera preview is displayed as the background of the screen. This allows the user to see what the camera is capturing. A futuristic viewfinder is placed at the centre of the screen. This provides a visual guide for the user to align the camera with the target.

Buttons and controls are placed at the top of the screen for easy access. These include a close button, a flash toggle button, and a camera rotation button. The scan button is located at the bottom of the screen for easy access. This button triggers the scanning functionality.

If the camera permission is not granted, a message is displayed at the centre of the screen informing the user of the denied permission.

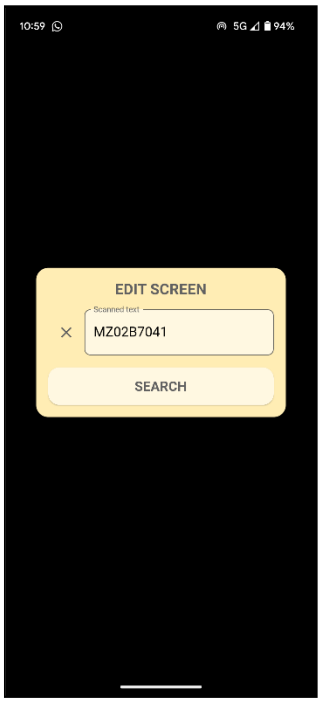
Overall, the Scanner is simple, intuitive, and easy to use. It provides all the necessary controls for the user to scan an image with ease.

Edit Screen –

This is the Edit Screen that opens after scanning a text from the Scanner. It allows the user to edit the scanned text in case the scanned text is being copied incorrectly.

It displays a dialog with a title, a close button, a text input field, and a "SEARCH" button. The text input field applies formatting and validation rules to allow alphanumeric characters only. When the dialog is closed, the edited text is returned to the caller.

By pressing the “SEARCH” button, it will navigate to the Result Screen that will display the detailed information of the scanned vehicle number if it is found.



(Fig. 2.4 – Edit Screen)

Result Screen –

This page will display the actual result of the scanned texts.

(Fig. 2.5 – Result Screen)

The body of the screen is contained with a light amber background color and rounded borders at the top-left and top-right corners. This gives a visually appealing look to the screen.

Inside the body, there is a Stream Builder that listens to a Firestore stream and displays different UI states based on the snapshot's data availability and errors.

When data is available, it is displayed in a list. Each item in the list represents a document from the Firestore collection.

Each document is displayed as a card with a light amber background color. The card contains various details related to the scanned text, such as vehicle number, owner information, contact details, address, vehicle model, type, and color.

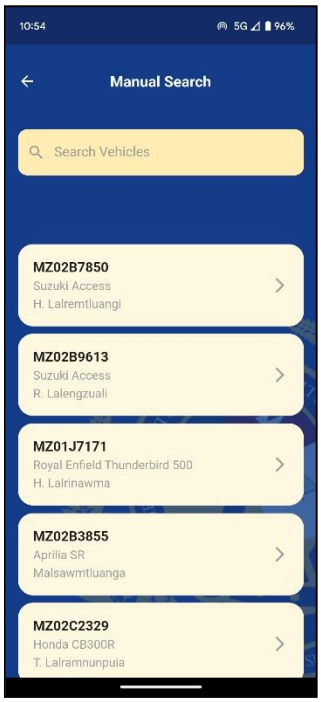
The UI design uses different font sizes and styles to distinguish between different types of information. The important details are displayed with a bold font weight.

The UI layout is designed to be vertically scrollable, allowing the user to view all the search results.

Overall, the **Result Screen** provides a visually appealing and organized UI for presenting the search result based on the scanned text.

Manual Search Page –

This is the page where users will be able to search for registered vehicle documents and look upon their details. The screen has a visually pleasing background image that adds an aesthetic touch to the interface.



(Fig. 2.6 – Manual Search Page)

The top of the screen features an AppBar with a centered title "Manual Search" in white text. The transparent background of the AppBar allows the background image to be visible.

Below the AppBar, there is a search bar where users can enter their search queries. The search bar has a filled background with rounded corners, a search icon, and a hint text "Search Vehicles". It allows users to filter the list of vehicles based on their search input.

The main content area contains an Expanded widget, which dynamically adjusts its height based on the available space. It hosts a list view that displays the filtered list of vehicles.

Each vehicle in the list is represented by a container with a light amber background color and rounded corners. Inside the container, the vehicle's information, such as its number, model, and owner, is displayed in a clear and readable format.

Users can tap on a vehicle to view detailed information on a separate screen. Tapping on a vehicle item triggers the navigation to the **Detailed Information Page**. If there are no search results, a message "No Results Found" is displayed at the centre of the screen, providing feedback to the user.

The Manual Search Screen combines aesthetics and functionality to create an intuitive and visually appealing interface for manual vehicle search.

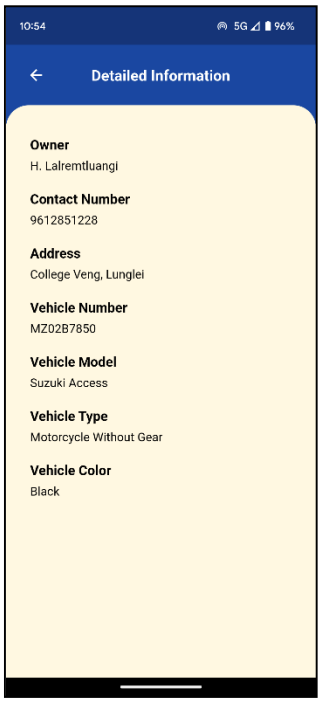
Detailed Information Page –

This is the page that displays all the detailed information of the selected vehicle data from the Manual Search Screen.

The content is displayed within a **SafeArea** widget to ensure it is visible and not obscured by system UI elements.

At the top of the page, there is a row containing an arrow back button and the title "Detailed Information." The back button allows the user to easily navigate back to the previous screen. The title is displayed in white text with a large and bold font, making it prominent and easy to read.

The main content area is represented by an Expanded widget, taking up the remaining vertical space. It is wrapped in a container with a light amber background color, giving it a card-like appearance.



(Fig. 2.7 – Detailed Information Page)

Inside the content area, there is a ListView widget that displays the detailed information about the vehicle. Each piece of information is presented in an organized manner, consisting a label and its corresponding value. The labels are displayed in a bold font, while the values are displayed in a regular font. There is appropriate spacing between each label-value pair, making it easy to scan and read the information.

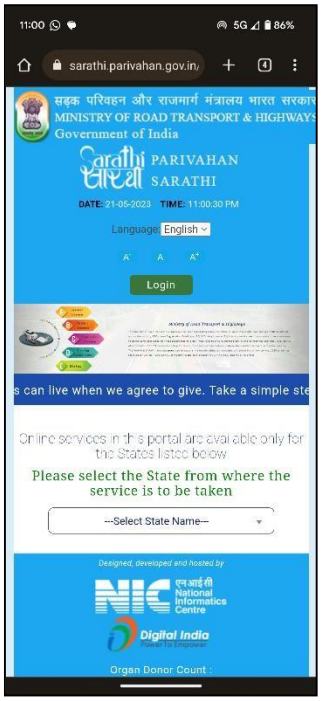
The layout of the detailed information follows a vertical arrangement, allowing the user to scroll through the content if it exceeds the available space.

The Detailed Information Page focuses on clarity and readability. The use of contrasting colors, consistent styling, and organized presentation of information ensures that users can easily access and understand the detailed information about the vehicle.

Official Sarathi Parivahan Website

This is the official Sarathi Parivahan website attached with the application. It will allow users to apply for Driving Licence and all other relevant services available online in the webpage.

This page will be available for all users and will be redirected by pressing the left-most button among the three small buttons below the “Manual Search” button.



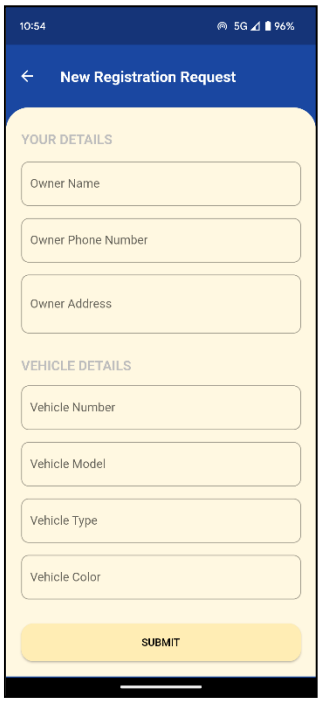
(Fig. 2.8 – Official Sarathi Parivahan Page)

New Registration Request Page –

The **New Registration Request Page** represents a user interface for submitting a new registration request. The main content area is contained within a rounded rectangular container with a light amber background color. This container stands out against the background, creating a visual separation and focus for the form.

The form is organized in a vertical layout, allowing users to easily navigate through the input fields. The spacing between the form sections and input fields enhances readability and clarity.

The form starts with a section titled "YOUR DETAILS," displayed in a grayish color, setting it apart from the rest of the form. This helps users quickly identify the section relevant to their personal information.



(Fig. 2.9 – New Registration Page)

Each input field within the form is accompanied by a descriptive label, such as "Owner Name," "Owner Phone Number," etc., providing clear guidance on the required information. The input fields are designed with rounded corners and a white background, making them visually distinct and ensuring user focus on the fields. The text entered by users is displayed in black, maintaining good contrast against the background.

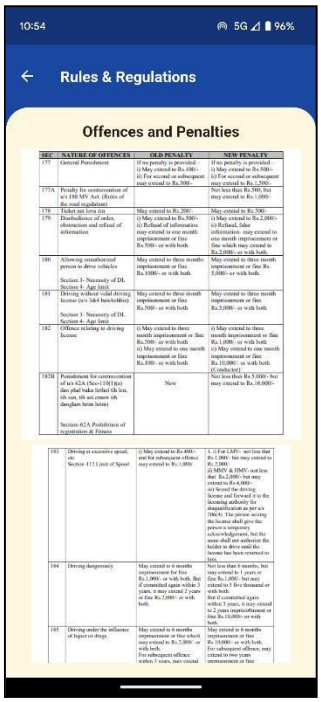
Input validation is implemented to ensure that the required fields are not left empty or contain invalid data. Error messages are displayed beneath the corresponding input fields if validation fails, guiding users in correcting their input.

The "SUBMIT" button is prominently displayed in the center of the screen, styled with a light amber background color. The button has rounded corners and a moderate size, making it easy for users to tap on both mobile and desktop devices.

Rules and Regulations/Offenses and Penalties Page –

The "Rules and Regulations" page features a blue background and a white app bar with a clear title. The content area has a rounded rectangular container with a light amber background. The page displays sections of offences and penalties using images loaded from local assets. Vertical scrolling is enabled to ensure all the information is accessible.

Overall, the UI presents the rules and regulations in a visually appealing and organized manner for easy comprehension.



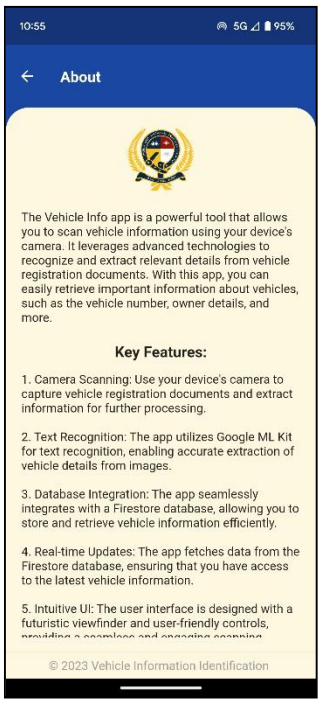
(Fig. 2.10 – Rules and Regulations Page)

About Page –

The “About” page includes an app logo, followed by a description of the "Vehicle Info" app, its key features, permissions required, and a disclaimer.

Additionally, the page provides information about the developers, including their names, roll numbers, registration numbers, and semester details. It also includes a "Contact Us" section with icons for phone, email, and address.

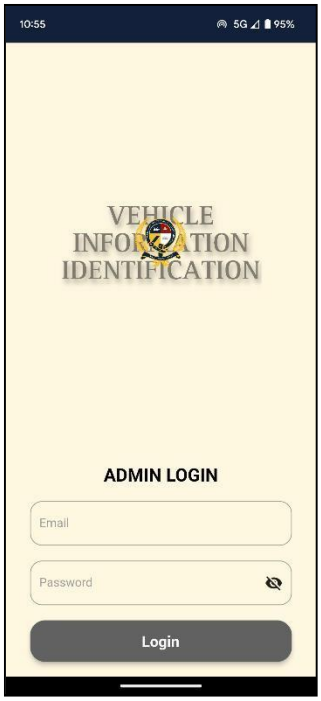
Overall, the UI of the "About" page presents information about the app, its features, developers, and contact details in a visually appealing and organized manner.



(Fig. 2.11 – About Page)

Admin Login Page –

The login screen allows users to enter their email and password for authentication.



(Fig. 2.12 – Admin Login Page)

The login screen UI consists of an image, a title ("ADMIN LOGIN"), and two input fields for email and password. The password field has a visibility toggle button to show/hide the entered password. There is also a "Login" button that triggers the login process.

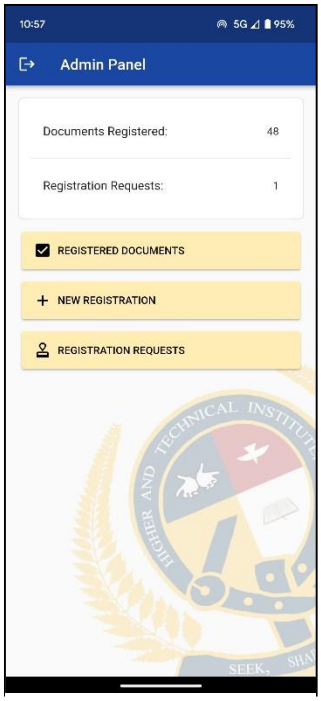
When the "Login" button is pressed, it attempts to sign in with the provided email and password using Firebase Authentication, and shows a loading indicator during the login process. If the login is successful, it navigates to the AdminPage and displays a success dialog. If an error occurs during login, it shows an error dialog.

The UI is designed with a white background and uses a gradient color scheme for the login button. It also incorporates visual feedback by disabling the button and showing a loading indicator while the login process is ongoing.

Overall, the **LoginPage** provides a basic login screen with email and password input fields, allowing users to authenticate and access the admin functionality of the app.

Admin Panel –

This is the page specifically created for the Admin so that they could monitor and look after the app's database. The entire body of the page is wrapped in a Stack, and a background image is positioned to fill the available space. This adds visual appeal to the page.



(Fig. 2.13 – Admin Panel)

Inside the Stack, there is a container that displays the counts for two categories: "Documents Registered" and "Registration Requests.". While the data is being fetched, a circular progress indicator is shown. Once the data is available, it is displayed as ListTile widgets.

Below the count information, there are three buttons arranged in a column. Each button represents a specific action:

1. "REGISTERED DOCUMENTS": This button, when pressed, navigates to the Registered Document Page. It displays an icon and text alongside.

2. "NEW REGISTRATION": This button, when pressed, navigates to the New Registration Page. It also displays an icon and text.

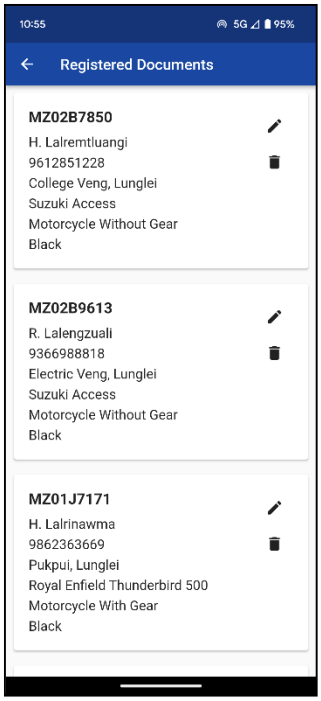
3. "REGISTRATION REQUESTS": This button, when pressed, navigates to the Registration Approval Page. It displays an icon and text as well.

The Admin Page class also includes a helper method getDocumentCount to fetch the count of documents in a specific collection using Firebase Firestore.

Overall, the UI of the **Admin Page** presents a clean and organized layout with relevant information and intuitive buttons for admins to perform actions related to registered documents and registration requests.

[Admin] Registered Documents Page –

The Registered Documents page is designed to display a list of registered documents in a visually appealing and organized manner.



(Fig. 2.14 – [Admin] Registered Documents Page)

The body of the page consists of a scrollable list of registered documents. Each document is displayed within a card that has a slight elevation, giving it a 3D effect. Each document card displays important information about the registered document. This includes the vehicle number, owner's name, owner's contact, owner's address, vehicle model, vehicle type, and vehicle color. The information is presented in a clear and readable format, with appropriate font sizes and spacing.

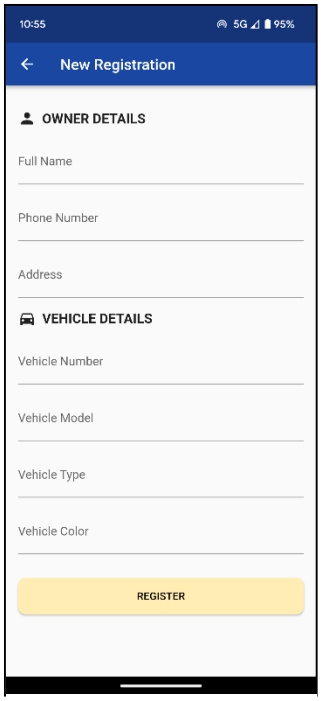
For each document, there are two icon buttons on the right side of the card. The first button, represented by an edit icon, allows the user to edit the document's details. When clicked, it opens an alert dialog with text input fields where the user can modify the information. The second button, represented by a delete icon, allows the user to delete the document from the list. Clicking this button triggers a confirmation dialog before the deletion is performed.

To handle the initial loading state and any potential errors, the page includes appropriate widgets. While the documents are being fetched from the database, a circular progress indicator is displayed at the centre of the screen. If an error occurs during the data retrieval process, an error message is shown to the user.

Overall, the UI of the Registered Documents page provides a user-friendly experience by presenting the registered documents in a visually pleasing manner and offering intuitive options to edit or delete them.

[Admin] New Document Registration Page –

The Admin's New Registration page is designed to provide a user-friendly and intuitive form for registering new documents.



(Fig. 2.15 – [Admin] New Registration Page)

The body of the page is wrapped in a SingleChildScrollView, allowing the form to scroll if it extends beyond the visible area. The form fields are arranged in a column layout, ensuring a logical and organized display of information.

The form is divided into two sections: "OWNER DETAILS" and "VEHICLE DETAILS." Each section is visually separated and labelled with an icon and a text heading. The headings are styled with a larger font size and bold weight to draw attention to them.

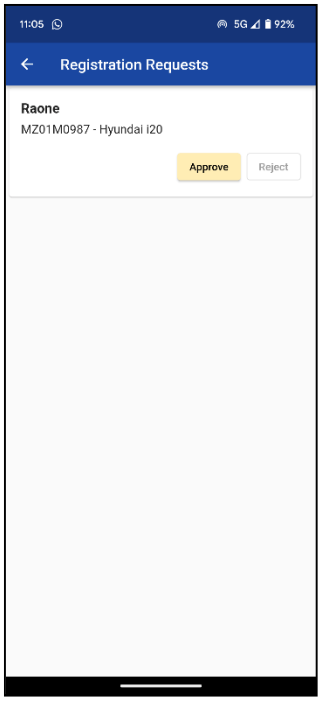
Each input field is represented by a Text Field widget. The input fields for owner details include "Full Name," "Phone Number," and "Address." The input fields for vehicle details include "Vehicle Number," "Vehicle Model," "Vehicle Type," and "Vehicle Color." Clear and concise labels are provided for each field, making it easy for users to understand the expected information.

The code includes logic to automatically capitalize the input in the "Vehicle Number" field and restricts the input of blank. These features help ensure data consistency and prevent common input errors.

At the bottom of the form, there is an elevated button labelled "REGISTER." This button stands out with a contrasting background color and spans the full width of the screen. When clicked, it triggers the a specific method, initiating the registration process.

[Admin] Registration Request Page –

The Registration Request page is designed to display a list of registration requests that can be approved or rejected.



(Fig. 2.16 – [Admin] Registration Requests Page)

The body of the page listens for changes in the "reg_requests" collection in Firestore. Based on the stream data, the UI is updated accordingly.

When there are new registration requests available, they are displayed as a list of cards. Each card represents a single request and contains the following information:

1. Owner Name: Displayed in bold with a font size of 18.
2. Vehicle Information: Displays the vehicle number and model with a font size of 16.
3. Actions: The card footer includes two buttons - "Approve" and "Reject" - placed on the right side of the card.

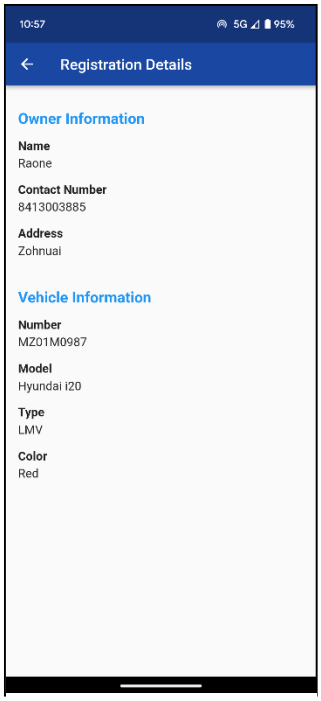
    - Approve Button: When clicked, it adds the request to the "veh_info" collection in Firestore, deletes the request from the "reg_requests" collection, and shows a success message in a snackbar.
    - Reject Button: When clicked, it stores a rejection notification, deletes the request from the "reg_requests" collection, and shows a success message in a snackbar.

Overall, the Registration Requests page provides a clear and intuitive interface for managing registration requests, allowing administrators to review and approve or reject requests efficiently.

[Admin] Registration Request Details Page –

This is the page that navigates to it when a registration request card is tapped.

This page displays detailed information about the selected registration request, including owner information (name, contact number, and address) and vehicle information (number, model, type, and color). The information is presented in a column layout with appropriate spacing.



(Fig. 2.17 – [Admin] Registration Requests Details Page)

Database Design

The database used in the app is Firebase Firestore. Firebase Firestore is a flexible, scalable NoSQL cloud database provided by Google as part of the Firebase platform. It is designed to store and sync data in real-time, making it suitable for building reactive and collaborative applications.

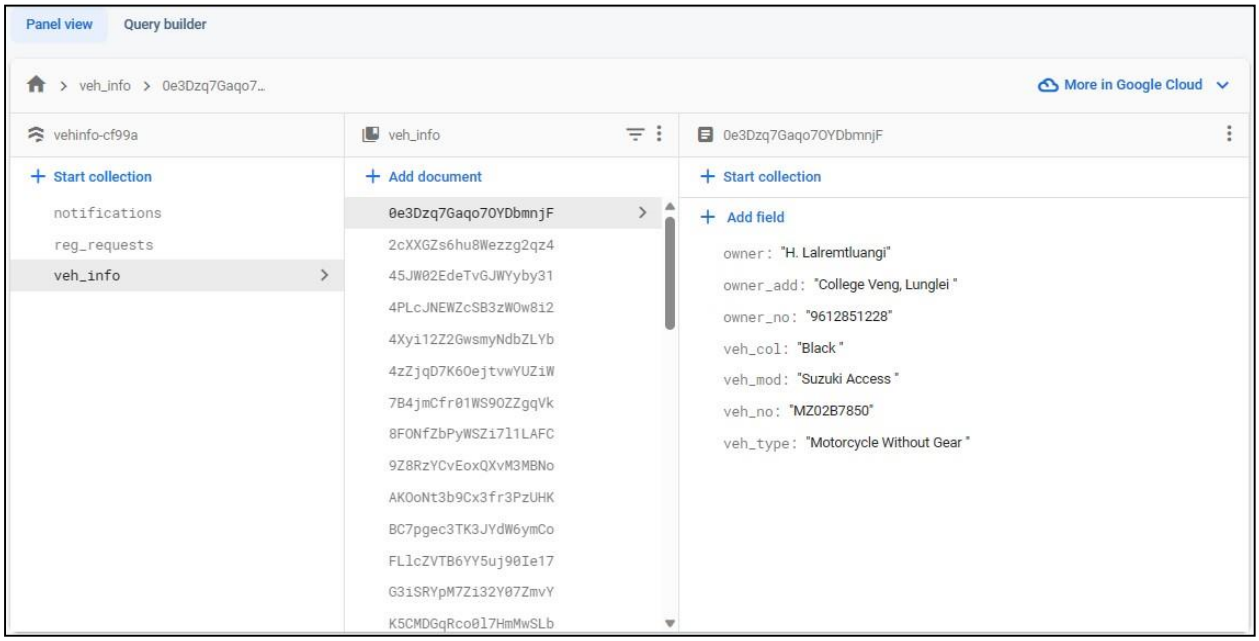
Firestore organizes data in collections, which are analogous to tables in traditional databases, and each document within a collection represents a record. Documents are stored in a key-value format, where each document has a unique identifier (ID) and contains a set of fields with their corresponding values. It provides real-time updates, which means that any changes made to the data in Firestore are immediately reflected in the app. This allows for real-time synchronization and collaboration between multiple users or devices.

For this project, three Firestore collections were created, namely –

1. **veh_info**:
   This collection stores all the vehicle information and details registered in the application.

2. **reg_requests**:
   This collection stores all the registration requests made by the users.

3. **notifications:**
   This collection stores all the events performed in the Registration Request page, New Registration page and Requests Approval page.



(Fig. 2.18 – Database Design)

In the above figure, the left-most column is the **collection** section. It displays all the collections created in the Firestore. New collection can be created by clicking the "+ Start Collection" text.

The middle column displays all the documents stored in the selected collection. Since we used Auto-generated ID, each document is represented by a mixed character. New document can be added by clicking the "+ Add document" text.

The right-most column represents the fields that are stored in each document. They are the main data that are being fetched and displayed by the application. Fields can be added by clicking the "+ Add field" text.

Data Dictionary

The app's database data dictionary provides an overview of the structure and meaning of the data stored in the Firebase Firestore database used by the application. Here's a summary of the database schema:

1. Collection: **veh_info**
    - This collection represents vehicle information stored in the app's database.

    Fields:
    - owner - It stores the name of the vehicle owner.
    - owner_add - It stores the address of the owner.
    - owner_no - It stores the contact number of the owner.
    - veh_no - It stores the vehicle number.
    - veh_mod - It stores the model of the vehicle.
    - veh_type - It stores the vehicle type.
    - veh_col - It stores the colour of the vehicle.

2. Collection: **reg_requests**
    - This collection represents registration requests sent by the user to the admin.

    Fields:
    - owner - It stores the name of the vehicle owner.
    - owner_add - It stores the address of the owner.
    - owner_no - It stores the contact number of the owner.
    - veh_no - It stores the vehicle number.
    - veh_mod - It stores the model of the vehicle.
    - veh_type - It stores the vehicle type.
    - veh_col - It stores the colour of the vehicle.
    - approved - It represents whether the request is approved or not.

3. Collection: **notifications**
    - This collection represents notifications that are generated by the events made in the New Registration Request page, New Registration page and the Registration Approval page.

Fields:

- date - It stores the date of the event being performed.
- message - It stores the message that is generated according to the event being performed.
- title - It stores the title of the event.

4. CODING

CONTENT

- **Complete Project Coding**
  - **Main**
  - **Splash Screen**
  - **Home**
  - **Edit Screen**
  - **Result Screen**
  - **Manual Search Screen**
  - **Detailed Information Page**
  - **New Registration Request**
  - **Rules and Regulations**
  - **About Page**
  - **Admin Login Page**
  - **Admin Panel**
  - **Registered Documents Page**
  - **New Document Registration Page**
  - **Registration Requests Page**
  - **Requests Factory**
- **Description of Coding Segments**

CODING

Complete Project Coding

Main –

```
import 'dart:io';

import 'package:camera/camera.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:google_mlkit_text_recognition/google_mlkit_text_recognition.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:vehicle_info/result_screen.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:vehicle_info/splash_screen.dart';
import 'edit_screen.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const App());
}

void fetchData() async {
  QuerySnapshot snapshot =
      await FirebaseFirestore.instance.collection('veh_info').get();
  for (var doc in snapshot.docs) {
    if (kDebugMode) {
      print(doc.data());
    }
  }
}

class App extends StatelessWidget {
```

```
  const App({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Vehicle Info',
      theme: ThemeData(
        primarySwatch: Colors.grey,
      ),
      home: const SplashScreen(),
    );
  }
}

class MainScreen extends StatefulWidget {
  const MainScreen({Key? key}) : super(key: key);

  @override
  State<MainScreen> createState() => _MainScreenState();
}

class _MainScreenState extends State<MainScreen> with WidgetsBindingObserver {
  bool _isPermissionGranted = false;
  late final Future<void> _future;
  CameraController? _cameraController;
  final textRecognizer = TextRecognizer();

  final double viewfinderWidthPercentage = 0.7;
  final double viewfinderHeightPercentage = 0.2;

  String? get editedText => null;

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
    _future = _requestCameraPermission();
  }
```

```
@override
void dispose() {
  WidgetsBinding.instance.removeObserver(this);
  _stopCamera();
  textRecognizer.close();
  super.dispose();
}

@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (_cameraController == null || !_cameraController!.value.isInitialized) {
    return;
  }

  if (state == AppLifecycleState.inactive) {
    _stopCamera();
  } else if (state == AppLifecycleState.resumed &&
      _cameraController != null &&
      _cameraController!.value.isInitialized) {
    _startCamera();
  }
}

bool _isFlashOn = false;

@override
Widget build(BuildContext context) {
  return FutureBuilder(
    future: _future,
    builder: (context, snapshot) {
      return Stack(
        children: [
          if (_isPermissionGranted)
            FutureBuilder<List<CameraDescription>>(
            future: availableCameras(),
              builder: (context, snapshot) {
                if (snapshot.hasData) {
                  _initCameraController(snapshot.data!);
```

```
        return Center(child: CameraPreview(_cameraController!));
      } else {
        return const LinearProgressIndicator();
      }
    },
  ),
 Scaffold(
  appBar: AppBar(
   title: const Text(
   'Scan',
    style: TextStyle(
      color: Colors.white,
    ),
   ),
   centerTitle: true,
   backgroundColor: Colors.blue[900],
   iconTheme: const IconThemeData(
   color: Colors.white,
   ),
   elevation: 0, // Remove the default shadow
   shape: const RoundedRectangleBorder(
   borderRadius: BorderRadius.vertical(
      bottom: Radius.circular(
         20), // Adjust the border radius as needed
    ),
   ),
  ),
  backgroundColor:
     _isPermissionGranted ? Colors.transparent : Colors.black,
  body: _isPermissionGranted
     ? Stack(
        children: [
         // Camera preview
         Positioned.fill(
           child: _cameraController != null
              ? CameraPreview(_cameraController!)
              : Container(),
         ),
```

```
// Futuristic viewfinder
Center(
  child: Container(
    decoration: BoxDecoration(
      border: Border.all(
        color: Colors.amber.shade50.withOpacity(0.7),
        width: 2.0,
      ),
      borderRadius: BorderRadius.circular(10.0),
    ),
    width: MediaQuery.of(context).size.width * 0.6,
    height: MediaQuery.of(context).size.width * 0.6,
  ),
),

// Buttons and controls
Positioned(
  top: 80.0,
  left: 30.0,
  right: 30.0,
  child: Container(
    decoration: BoxDecoration(
      color: Colors.amber.shade50.withOpacity(0.3),
      borderRadius: BorderRadius.circular(10.0),
    ),
    padding: const EdgeInsets.all(10.0),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        IconButton(
          icon: const Icon(Icons.close,
              color: Colors.white),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
        IconButton(
          icon: _isFlashOn ? const Icon(Icons.flash_on) : const Icon(Icons.flash_off),
```

```
                color: Colors.white,
              onPressed: () {
                _toggleFlash();
              },
            ),
            IconButton(
              icon: const Icon(
                Icons.flip_camera_ios_outlined,
                color: Colors.white),
              onPressed: () {
                _rotateCamera();
              },
            ),
          ],
        ),
      ),
    ),

    // Scan button
    Align(
      alignment: Alignment.bottomCenter,
      child: Container(
        padding: const EdgeInsets.only(bottom: 100.0),
        child: FloatingActionButton(
          onPressed: _scanImage,
          backgroundColor: Colors.amber.shade50.withOpacity(0.5),
          child: const Icon(Icons.camera_alt,
              color: Colors.black),
        ),
      ),
    ),
  ],
 )
: Center(
    child: Container(
      padding: const EdgeInsets.only(left: 24.0, right: 24.0),
      child: const Text(
        'Camera permission denied',
        textAlign: TextAlign.center,
```

```
                ),
              ),
            ),
          )
        ],
      );
    },
  );
}

void _rotateCamera() async {
  // Check if the device has multiple cameras
  final cameras = await availableCameras();
  if (cameras.length < 2) {
    return; // Do nothing if there's only one camera
  }

  // Get the index of the current camera
  final currentCameraIndex = cameras.indexOf(_cameraController!.description);

  // Calculate the index of the next camera
  final nextCameraIndex = (currentCameraIndex + 1) % cameras.length;

  // Dispose the current camera controller and switch to the next camera
  await _cameraController!.dispose();
  _cameraController =
      CameraController(cameras[nextCameraIndex], ResolutionPreset.high);
  await _cameraController!.initialize();

  // Refresh the camera preview
  setState(() {});
}

void _toggleFlash() async {
  try {
    if (_isFlashOn) {
      await _cameraController!.setFlashMode(FlashMode.off);
      setState(() {
        _isFlashOn = false;
```

```
      });
    } else {
      await _cameraController!.setFlashMode(FlashMode.torch);
      setState(() {
        _isFlashOn = true;
      });
    }
  } catch (e) {
    if (kDebugMode) {
      print(e);
    }
  }
}

Future<void> _requestCameraPermission() async {
  final status = await Permission.camera.request();
  _isPermissionGranted = status == PermissionStatus.granted;
}

void _startCamera() {
  if (_cameraController != null) {
    _cameraSelected(_cameraController!.description);
  }
}

void _stopCamera() {
  if (_cameraController != null) {
    _cameraController?.dispose();
  }
}

void _initCameraController(List<CameraDescription> cameras) {
  if (_cameraController != null) {
    return;
  }

  CameraDescription? camera;
  for (var i = 0; i < cameras.length; i++) {
    final CameraDescription current = cameras[i];
```

```
    if (current.lensDirection == CameraLensDirection.back) {
      camera = current;
      break;
    }
  }

  if (camera != null) {
    _cameraSelected(camera);
  }
}

Future<void> _cameraSelected(CameraDescription camera) async {
  _cameraController = CameraController(
    camera,
    ResolutionPreset.max,
    enableAudio: false,
  );

  await _cameraController!.initialize();
  await _cameraController!.setFlashMode(FlashMode.off);

  if (!mounted) {
    return;
  }
  setState(() {});
}

Future<void> _scanImage() async {
  if (_cameraController == null) return;

  final navigator = Navigator.of(context);

  try {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) {
      return const Center(
          child: CircularProgressIndicator(),
```

```
    );
  },
);

final pictureFile = await _cameraController!.takePicture();
final file = File(pictureFile.path);
final inputImage = InputImage.fromFile(file);
final recognizedText = await textRecognizer.processImage(inputImage);
String? searchText = recognizedText.text.replaceAll(' ', '');

Navigator.of(context).pop();

final stateCodes = [
  'AP', 'AR', 'AS', 'BR', 'CG', 'GA', 'GJ', 'HR', 'HP', 'JH', 'KA', 'KL', 'MP', 'MH',
  'MN', 'ML', 'MZ', 'NL', 'OD', 'PB', 'RJ', 'SK', 'TN', 'TS', 'TR', 'UP', 'UK', 'WB'
];

final regExp = RegExp(r"\b(" + stateCodes.join("|") + r")\d{1,2}[A-Z\d]{0,6}\b");
final match = regExp.firstMatch(searchText);
if (match != null) {
  searchText = match.group(0)!;
} else {
  searchText = '';
}

if (searchText.isNotEmpty) {
  final querySnapshot = await FirebaseFirestore.instance
      .collection('veh_info')
      .where('veh_no', isEqualTo: searchText)
      .where('owner', isEqualTo: searchText)
      .get();

  if (querySnapshot.docs.isNotEmpty) {
    await navigator.push(
    MaterialPageRoute(
        builder: (BuildContext context) => ResultScreen(
          text: searchText!,
          data: querySnapshot.docs,
        ),
```

```
        ),
      );
    } else {
      final editedText = await navigator.push(
        MaterialPageRoute(
          builder: (BuildContext context) => EditScreen(text: searchText),
        ),
      );

      if (editedText != null) {
        searchText = editedText;
      }

      await navigator.push(
        MaterialPageRoute(
          builder: (BuildContext context) => ResultScreen(
            text: searchText!,
            data: const [],
          ),
        ),
      );
    }
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
        content: Text(
          'Could not find valid Indian state vehicle code in the scanned text',
        ),),);
  }
} catch (e) {
  ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(
      content: Text('An error occurred when scanning text'),
    ),);
}}}
```

Splash Screen –

```
import 'package:flutter/material.dart';
import 'home_screen.dart';

class SplashScreen extends StatefulWidget {
  const SplashScreen({Key? key}) : super(key: key);

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  double opacityLevel = 0.0;

  @override
  void initState() {
    super.initState();
    Future.delayed(const Duration(seconds: 1), () {
      setState(() {
        opacityLevel = 1.0;
      });
    });
    Future.delayed(const Duration(seconds: 3), () {
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (context) => const HomeScreen(role: 'Home',)),
      );
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.amber[50],
      body: Center(
        child: AnimatedOpacity(
          opacity: opacityLevel,
          duration: const Duration(seconds: 1),
          curve: Curves.easeInOut,
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset(
              'assets/splash_image.png',
              width: 300,
              height: 300,
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Home –

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:url_launcher/url_launcher.dart';
import 'package:vehicle_info/new_reg_user.dart';
import 'package:vehicle_info/rules_regulations.dart';
import 'about.dart';
import 'login_page.dart';
import 'main.dart';
import 'manual_search_screen.dart';
import 'package:intl/intl.dart';

class HomeScreen extends StatefulWidget {
  const HomeScreen({Key? key, required String role}) : super(key: key);

  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  List<Map<String, dynamic>> notifications = []; // Define notifications list
  bool hasNewUpdates = false; // Define hasNewUpdates variable

  // Implement the fetchNotifications function
  Future<List<Map<String, dynamic>>> fetchNotifications() async {
  final CollectionReference notificationsCollection =
      FirebaseFirestore.instance.collection('notifications');

   final QuerySnapshot querySnapshot = await notificationsCollection.get();

   final List<QueryDocumentSnapshot> documents = querySnapshot.docs;
   final List<Map<String, dynamic>> notifications =
      documents.map((doc) => doc.data() as Map<String, dynamic>).toList();

   return notifications;
  }

  // Implement the checkForUpdates function
```

```
bool checkForUpdates(List<Map<String, dynamic>> notifications) {
  // Your logic to check for updates
  // Return true if there are new updates, false otherwise
  // Example:
  return notifications.isNotEmpty;
}

@override
Widget build(BuildContext context) {
  final buttonWidth = MediaQuery.of(context).size.width * 0.8;

  return Scaffold(
      appBar: AppBar(
      title: const Text(
          'Home',
          style: TextStyle(
            color: Colors.white,
          ),
        ),
        centerTitle: true,
        backgroundColor: Colors.transparent,
        elevation: 0,
        actions: [
          IconButton(
          onPressed: () {
              showDialog(
                context: context,
                builder: (context) {
                  return const LoginPage();
                },
              );
            },
            icon: const Icon(Icons.admin_panel_settings, color: Colors.white),
          ),
          IconButton(
            onPressed: () {
            Navigator.push(
                context,
                MaterialPageRoute(
```

```
            builder: (context) => const About(),
          ),
        );
      },
      icon: const Icon(Icons.info, color: Colors.white),
    ),
  ],
),
extendBodyBehindAppBar: true,
body: Container(
  decoration: const BoxDecoration(
    image: DecorationImage(
      image: AssetImage('assets/background_image.png'),
      fit: BoxFit.cover,
    ),
  ),
  child: ColorFiltered(
    colorFilter: ColorFilter.mode(
      Colors.blue.shade900.withOpacity(0.85),
      BlendMode.dstATop,
    ),
    child: Container(
      decoration: BoxDecoration(
        color: Colors.blue[900],
      ),
      child: Column(children: [
        const SizedBox(height: 100),
        SizedBox(
          width: buttonWidth,
          height: 60,
          child: ElevatedButton(
            onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => const MainScreen(),
              ),
            );
          },
```

```
    style: ElevatedButton.styleFrom(
     backgroundColor: Colors.amber[200],
     shape: RoundedRectangleBorder(
     borderRadius: BorderRadius.circular(20),
     ),
     elevation: 10,
    ),
    child: Row(
     mainAxisAlignment: MainAxisAlignment.start,
     children: const [
     Icon(Icons.text_snippet_outlined),
     SizedBox(width: 8.0),
      Text(
       'SCANNER',
       style: TextStyle(
        fontSize: 20,
        fontWeight: FontWeight.bold,
       ),
      ),
     ],
    ),
   ),
),
const SizedBox(height: 16.0),
SizedBox(
  width: buttonWidth,
  height: 60,
  child: ElevatedButton(
   onPressed: () {
   Navigator.push(
     context,
     MaterialPageRoute(
      builder: (context) => const ManualSearchScreen(),
     ),
    );
   },
   style: ElevatedButton.styleFrom(
    backgroundColor: Colors.amber[200],
    shape: RoundedRectangleBorder(
```

```
        borderRadius: BorderRadius.circular(20),
      ),
      elevation: 10,
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.start,
      children: const [
        Icon(Icons.search),
        SizedBox(width: 8.0),
        Text(
          'MANUAL SEARCH',
          style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ),
),
const SizedBox(height: 16.0),
SizedBox(
  width: buttonWidth,
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      SizedBox(
        width: buttonWidth / 3.5,
        height: 60,
        child: ElevatedButton(
          onPressed: () {
            const url =
                'https://sarathi.parivahan.gov.in/sarathiservice/stateSelection.do';
            launch(url);
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.amber[100],
            shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(20),
```

```
      ),
      elevation: 10,
    ),
    child: const Icon(Icons.badge_outlined),
  ),
),
SizedBox(
  width: buttonWidth / 3,
  height: 60,
  child: ElevatedButton(
    onPressed: () {
    Navigator.push(
       context,
       MaterialPageRoute(
       builder: (context) =>
            const NewRegistrationPageUser(),
       ),
     );
    },
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.amber[100],
      shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20),
      ),
      elevation: 10,
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: const [
      Icon(Icons.person_add_outlined),
      SizedBox(width: 8.0),
      ],
    ),
  ),
),
SizedBox(
  width: buttonWidth / 3.5,
  height: 60,
  child: ElevatedButton(
```

```
          onPressed: () {
            Navigator.push(
            context,
            MaterialPageRoute(
               builder: (context) =>
                   const RulesAndRegulations(),
              ),
            );
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.amber[100],
            shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(20),
            ),
            elevation: 10,
          ),
          child: const Icon(Icons.rule),
        ),
      ),
    ],
  ),
),
const SizedBox(height: 16.0),
Expanded(
  child: Container(
    decoration: BoxDecoration(
      color: Colors.amber[50],
      borderRadius: const BorderRadius.only(
        topLeft: Radius.circular(30),
        topRight: Radius.circular(30),
      ),
    ),
    child: Column(
      children: [
      const Padding(
          padding: EdgeInsets.all(16.0),
          child: Text(
            'Notifications',
            style: TextStyle(
```

```
      fontSize: 20,
      fontWeight: FontWeight.bold,
    ),
  ),
),
Expanded(
  child: RefreshIndicator(
    backgroundColor: Colors.amber[100],
    color: Colors.blue[900],
    onRefresh: () async {
      List<Map<String, dynamic>>
          refreshedNotifications =
          await fetchNotifications();

      bool hasUpdates =
          checkForUpdates(refreshedNotifications);

      setState(() {
        notifications = refreshedNotifications;
        hasNewUpdates = hasUpdates;
      });

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Notifications refreshed'),
        ),
      );
    },
    child:
        FutureBuilder<List<Map<String, dynamic>>>(
      future: fetchNotifications(),
      builder: (context, snapshot) {
        if (snapshot.connectionState ==
          ConnectionState.waiting) {
          return const Center(
              child: CircularProgressIndicator());
        }

        if (snapshot.hasError) {
```

```
 return const Center(
    child: Text(
      'Failed to fetch notifications'));
}

notifications = snapshot.data ?? [];
notifications.sort((a, b) => b['date']
  .compareTo(
    a['date'])); // Sort by newest first

final displayNotifications =
  notifications.take(8).toList();

return ListView.builder(
 itemCount: displayNotifications.length,
 itemBuilder: (context, index) {
  final notification =
    displayNotifications[index];
  final String title =
    notification['title'] ?? '';
  final String message =
    notification['message'] ?? '';
  final Timestamp timestamp =
    notification['date'];
  final DateTime dateTime =
    timestamp.toDate();
  final String formattedDate =
    DateFormat('dd-MM').format(
      dateTime); // Format the date as desired

  return ListTile(
   leading: Text(formattedDate),
   title: Text(
    title,
    style: const TextStyle(
     fontSize: 18,
     fontWeight: FontWeight.bold,
    ),
   ),
```

```
                        subtitle: Padding(
                          padding:
                              const EdgeInsets.only(top: 4.0),
                          child: Text(
                            message,
                            style:
                                const TextStyle(fontSize: 16),
                          ),
                        ),
                        trailing: getTrailingIcon(title),
                      );},);
                  },),),),
              ],),),)
          ]),
        )))); }

  Widget getTrailingIcon(String title) {
    IconData trailingIcon;
    switch (title) {
      case 'Registration Request Approved':
        trailingIcon = Icons.check_box_outlined;
        break;
      case 'Registration Request Rejected':
        trailingIcon = Icons.close;
        break;
      case 'New Document Registered':
        trailingIcon = Icons.person_add_alt_outlined;
        break;
      case 'New Registration Request':
        trailingIcon = Icons.info_outline;
        break;
      default:
        trailingIcon = Icons.info_outline;
        break;
    }
    return Icon(trailingIcon);

}}
```

Edit Screen –

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

class EditScreen extends StatefulWidget {
  final String? text;
  final String notNullString;

  EditScreen({Key? key, required this.text})
      : notNullString = myString ?? '',
        super(key: key);

  static get myString => null;

  @override
  EditScreenState createState() => EditScreenState();
}

class EditScreenState extends State<EditScreen> {
  late TextEditingController _textEditingController;

  @override
  void initState() {
    super.initState();
    _textEditingController = TextEditingController(text: widget.text);
  }

  @override
  void dispose() {
    _textEditingController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Dialog(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(16.0),
```

```
),
backgroundColor: Colors.amber[100],
child: Padding(
 padding: const EdgeInsets.all(16.0),
 child: Column(
  mainAxisSize: MainAxisSize.min,
  children: [
   Align(
    alignment: Alignment.center,
    child: Text(
     'EDIT SCREEN',
     style: TextStyle(
      fontSize: 20,
      fontWeight: FontWeight.bold,
      color: Colors.grey[700],
     ),
    ),
   ),
   const SizedBox(height: 16),
   Row(
    children: [
     IconButton(
      onPressed: () => Navigator.of(context).pop(),
      icon: Icon(
       Icons.close,
       color: Colors.grey[700],
      ),
     ),
     Expanded(
      child: TextField(
       inputFormatters: [
        AlphaNumericInputFormatter(),
        FilteringTextInputFormatter.deny(RegExp(r'\\s')),
        FilteringTextInputFormatter.allow(RegExp(r'[A-Z\d]')),
       ],
       controller: _textEditingController,
       autofocus: true,
       keyboardType: TextInputType.text,
       decoration: InputDecoration(
```

```
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8.0),
          ),
          filled: true,
          fillColor: Colors.amber[50],
          labelText: 'Scanned text',
          labelStyle: TextStyle(
          fontSize: 14,
            color: Colors.grey[700],
          ),
        ),
        style: const TextStyle(
          fontSize: 18,
          color: Colors.black,
        ),
        textCapitalization: TextCapitalization.characters,
      ),
    ),
  ],
),
const SizedBox(height: 16),
ElevatedButton(
onPressed: () =>
    Navigator.of(context).pop(_textEditingController.text),
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.amber[50],
    elevation: 1,
    minimumSize: const Size(double.infinity, 50),
    shape: RoundedRectangleBorder(
    borderRadius: BorderRadius.circular(15.0),
    ),
  ),
  child: Text(
    'SEARCH',
    style: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.bold,
      color: Colors.grey[700],
    ),
```

```
            ),
          ),
        ],
      ),
    ),
  );
 }
}

class AlphaNumericInputFormatter extends TextInputFormatter {
 static final _regExp = RegExp(r'[a-zA-Z\d]');

 @override
 TextEditingValue formatEditUpdate(
    TextEditingValue oldValue, TextEditingValue newValue) {
  final filteredValue = String.fromCharCodes(
   newValue.text.runes.where(_regExp.hasMatch as bool Function(int element)),
  );

  return newValue.copyWith(text: filteredValue);
 }
}
```

Result Screen –

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class ResultScreen extends StatelessWidget {
  final String text;

  const ResultScreen(
     {Key? key, required this.text, required List<dynamic> data})
     : super(key: key);

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      backgroundColor: Colors.blue[900],
       appBar: AppBar(
        title: const Text('Result'),
        toolbarHeight: 83,
        backgroundColor: Colors.blue[900],
        elevation: 0,
        iconTheme: const IconThemeData(
          color: Colors.white, // Set icon color to white
        ),
        toolbarTextStyle: const TextTheme(
          titleLarge: TextStyle(
            color: Colors.white, // Set text color to white
            fontSize: 20,
            fontWeight: FontWeight.bold,
          ),
        ).bodyMedium,
        titleTextStyle: const TextTheme(
          titleLarge: TextStyle(
            color: Colors.white, // Set text color to white
            fontSize: 20,
            fontWeight: FontWeight.bold,
          ),
        ).titleLarge,
```

```
),
body: Container(
 decoration: BoxDecoration(
 color: Colors.amber[50],
  borderRadius: const BorderRadius.only(
   topLeft: Radius.circular(30),
   topRight: Radius.circular(30),
  ),
 ),
 child: StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
     .collection('veh_info')
     .where('veh_no', isEqualTo: text)
     .snapshots(),
  builder: (context, snapshot) {
   if (snapshot.hasError) {
    return Center(child: Text('Error: ${snapshot.error}'));
   }

   if (!snapshot.hasData || snapshot.data!.docs.isEmpty) {
    return const Center(child: Text('No matches found.'));
   }

   final data = snapshot.data!.docs;
   return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Padding(
      padding: const EdgeInsets.all(20.0),
      child: Text(
       'Scanned Text: $text',
       style: const TextStyle(
       fontSize: 24,
        fontWeight: FontWeight.bold,
        height: 2,
       ),
      ),
     ),
     const Divider(),
```

```
Expanded(
 child: ListView.builder(
  itemCount: data.length,
  itemBuilder: (context, index) {
  final doc = data[index];
   final owner = doc['owner'] as String;
   final ownerNo = doc['owner_no'] as String;
   final ownerAdd = doc['owner_add'] as String;
   final vehNo = doc['veh_no'] as String;
   final vehMod = doc['veh_mod'] as String;
   final vehType = doc['veh_type'] as String;
   final vehCol = doc['veh_col'] as String;
   return Card(
    color: Colors.amber[50],
    child: Padding(
     padding: const EdgeInsets.all(16.0),
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       Row(
        mainAxisAlignment:
          MainAxisAlignment.spaceBetween,
        children: [
         const Text(
          'Vehicle No:',
          style: TextStyle(
          fontSize: 20,
           fontWeight: FontWeight.bold,
          ),
         ),
         Text(
          vehNo,
          style: const TextStyle(
           fontSize: 20,
           fontWeight: FontWeight.bold,
          ),
         ),
        ],
       ),
```

```
const SizedBox(height: 30),
const Text(
  'Owner:',
  style: TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
  ),
),
Text(
    owner,
  style: const TextStyle(
    fontSize: 18,
  ),
),
const SizedBox(height: 15),
const Text(
  'Contact:',
  style: TextStyle(
  fontSize: 18,
    fontWeight: FontWeight.bold,
  ),
),
Text(ownerNo,
  style: const TextStyle(
    fontSize: 18,
  ),),
const SizedBox(height: 15),
const Text(
  'Address:',
  style: TextStyle(
  fontSize: 18,
    fontWeight: FontWeight.bold,
  ),
),
Text(ownerAdd,
  style: const TextStyle(
    fontSize: 18,
  ),),
const SizedBox(height: 15),
```

```
                const Text(
                  'Vehicle Model:',
                  style: TextStyle(
                  fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                Text(vehMod,
                  style: const TextStyle(
                    fontSize: 18,
                  ),),
                const SizedBox(height: 15),
                const Text(
                  'Vehicle Type:',
                  style: TextStyle(
                     fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                Text(vehType,
                  style: const TextStyle(
                    fontSize: 18,
                  ),),
                const SizedBox(height: 15),
                const Text(
                  'Vehicle  Color:',
                  style: TextStyle(
                  fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                Text(vehCol,
                  style: const TextStyle(
                    fontSize: 18,
                  ),),],),),
            );},),),],);
      },),));

}}
```

Manual Search Screen -

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:vehicle_info/detailed_information_page.dart';

class ManualSearchScreen extends StatefulWidget {
  const ManualSearchScreen({Key? key}) : super(key: key);

  @override
  ManualSearchScreenState createState() => ManualSearchScreenState();
}

class ManualSearchScreenState extends State<ManualSearchScreen> {
  FirebaseFirestore firestore = FirebaseFirestore.instance;

  List<DocumentSnapshot> _dataList = [];
  List<DocumentSnapshot> _filteredDataList = [];

  Future<void> fetchData() async {
    try {
      QuerySnapshot querySnapshot =
          await firestore.collection('veh_info').get();
      setState(() {
        _dataList = querySnapshot.docs;
        _filteredDataList = _dataList;
      });
    } catch (e) {
      if (kDebugMode) {
        print('Error fetching data: $e');
      }
    }
  }

  void _filterDataList(String searchText) {
    setState(() {
      _filteredDataList = _dataList.where((documentSnapshot) {
        final data = documentSnapshot.data() as Map<String, dynamic>?;
```

```
      final vehNo = data?['veh_no']?.toString().toLowerCase() ?? '';
      final vehMod = data?['veh_mod']?.toString().toLowerCase() ?? '';
      final vehCol = data?['veh_col']?.toString().toLowerCase() ?? '';
      final owner = data?['owner']?.toString().toLowerCase() ?? '';

      return vehNo.contains(searchText.toLowerCase()) ||
          vehMod.contains(searchText.toLowerCase()) ||
          vehCol.contains(searchText.toLowerCase()) ||
          owner.contains(searchText.toLowerCase());
    }).toList();
  });
}

void _onListItemTap(DocumentSnapshot documentSnapshot) {
  Navigator.push(
    context,
    MaterialPageRoute(
    builder: (context) =>
        DetailedInformationPage(documentSnapshot: documentSnapshot),
    ),
  );
}

@override
void initState() {
  super.initState();
  fetchData();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor:
        Colors.transparent, // Make the scaffold's background transparent
    body: Container(
      decoration: BoxDecoration(
        color: Colors.blue.shade900.withOpacity(0.85),
        image: const DecorationImage(
          image: AssetImage('assets/background_image.png'),
```

```
    fit: BoxFit.cover,
   ),
  ),
  child: Column(
   crossAxisAlignment: CrossAxisAlignment.stretch,
   children: [
    const SizedBox(
     height: 20,
    ),
    AppBar(
     elevation: 0,
     backgroundColor: Colors.transparent,
     iconTheme:
        const IconThemeData(color: Colors.white), // add this line
     title: const Text(
      'Manual Search',
      style: TextStyle(
        fontSize: 20,
        fontWeight: FontWeight.bold,
        color: Colors.white),
     ),
     centerTitle: true,
    ),
    const SizedBox(
     height: 20,
    ),
    Container(
     padding: const EdgeInsets.all(16),
     child: TextField(
      onChanged: _filterDataList,
      style: const TextStyle(fontSize: 16),
      decoration: InputDecoration(
      hintText: 'Search Vehicles',
       hintStyle: const TextStyle(fontSize: 18, color: Colors.grey),
       prefixIcon: const Icon(Icons.search, color: Colors.grey),
       filled: true,
       fillColor: Colors.amber[100],
       enabledBorder: const OutlineInputBorder(
        borderRadius: BorderRadius.all(Radius.circular(12)),
```

```
        borderSide: BorderSide(color: Colors.transparent),
      ),
      focusedBorder: const OutlineInputBorder(
        borderRadius: BorderRadius.all(Radius.circular(12)),
        borderSide: BorderSide(color: Colors.transparent),
      ),
    ),
  ),
),
const SizedBox(
  height: 20,
),
Expanded(
  child: _filteredDataList.isNotEmpty
      ? ListView.builder(
          itemCount: _filteredDataList.length,
          itemBuilder: (context, index) {
            final data = _filteredDataList[index].data()
                as Map<String, dynamic>?;

            if (data != null) {
              final vehNo = data['veh_no'] ?? '';
              final vehMod = data['veh_mod'] ?? '';
              final owner = data['owner'] ?? '';

              return GestureDetector(
                onTap: () =>
                    _onListItemTap(_filteredDataList[index]),
                child: Container(
                  margin: const EdgeInsets.symmetric(
                      horizontal: 16, vertical: 5),
                  padding: const EdgeInsets.all(20),
                  decoration: BoxDecoration(
                  color: Colors.amber[50],
                    borderRadius: BorderRadius.circular(20),
                  ),
                  child: Row(
                    mainAxisAlignment:
                        MainAxisAlignment.spaceBetween,
```

```
        children: [
          Column(
            crossAxisAlignment:
                CrossAxisAlignment.start,
            children: [
              Text(
              vehNo,
                style: const TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold),
              ),
              const SizedBox(
                height: 5,
              ),
              Text(
                vehMod,
                style: const TextStyle(
                    fontSize: 16, color: Colors.grey),
              ),
              const SizedBox(
                height: 5,
              ),
              Text(
                owner,
                style: const TextStyle(
                    fontSize: 16, color: Colors.grey),
              ),
            ],
          ),
          const Icon(
            Icons.arrow_forward_ios,
            color: Colors.grey,
          ),
        ],
      ),
    ),
  );
} else {
  return const ListTile(
```

```
                    title: Text('Error: Data is null'),
                  );
                }
              },
            )
          : Center(
              child: Text(
                'No Results Found',
                style: TextStyle(
                fontSize: 20,
                  color: Colors.grey[400],
                ),
              ),
            ),
        ),
      ],
    ),
  ),
 );
 }
}
```

Detailed Information Page –

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class DetailedInformationPage extends StatelessWidget {
  final DocumentSnapshot documentSnapshot;

  const DetailedInformationPage({Key? key, required this.documentSnapshot})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
   final data = documentSnapshot.data() as Map<String, dynamic>?;

   if (data != null) {
    final owner = data['owner'] ?? '';
    final ownerNo = data['owner_no'] ?? '';
    final ownerAdd = data['owner_add'] ?? '';
    final vehNo = data['veh_no'] ?? '';
    final vehMod = data['veh_mod'] ?? '';
    final vehType = data['veh_type'] ?? '';
    final vehCol = data['veh_col'] ?? '';

    return Scaffold(
     backgroundColor: Colors.blue[900],
     body: SafeArea(
      child: Column(
       crossAxisAlignment: CrossAxisAlignment.start,
       children: [
        Padding(
         padding: const EdgeInsets.all(16.0),
         child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
           IconButton(
            onPressed: () {
             Navigator.pop(context);
            },
```

```
        icon: const Icon(
          Icons.arrow_back,
          color: Colors.white,
        ),
      ),
      const Text(
        'Detailed Information',
        style: TextStyle(
        fontSize: 20.0,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
      const SizedBox(width: 48.0),
    ],
  ),
),
Expanded(
  child: Container(
    decoration: BoxDecoration(
      color: Colors.amber[50],
      borderRadius: const BorderRadius.only(
        topLeft: Radius.circular(30.0),
        topRight: Radius.circular(30.0),
      ),
    ),
    child: ListView(
      padding: const EdgeInsets.all(32.0),
      children: [
        const SizedBox(height: 10.0),
        _buildInfoItem('Owner', owner),
        const SizedBox(),
        _buildInfoItem('Contact Number', ownerNo),
        const SizedBox(),
        _buildInfoItem('Address', ownerAdd),
        const SizedBox(),
        _buildInfoItem('Vehicle Number', vehNo),
        const SizedBox(),
        _buildInfoItem('Vehicle Model', vehMod),
```

```
                const SizedBox(),
                _buildInfoItem('Vehicle Type', vehType),
                const SizedBox(),
                _buildInfoItem('Vehicle Color', vehCol),
                const SizedBox(),
              ],),),),
        ],),),
    );
  }

  return const SizedBox.shrink();
 }

 Widget _buildInfoItem(String label, String value) {
  return Column(
   crossAxisAlignment: CrossAxisAlignment.start,
   children: [
    Text(
     label,
     style: const TextStyle(
      fontSize: 18.0,
      fontWeight: FontWeight.bold,
      color: Colors.black,
     ),
    ),
    const SizedBox(height: 8.0),
    Text(
     value,
     style: const TextStyle(
      fontSize: 16.0,
      color: Colors.black,
     ),
    ),
    const SizedBox(height: 24.0),
   ],);
  }
}
```

New Registration Request Page –

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/services.dart';
import 'package:intl/intl.dart';

class RegistrationRequest {
  final String owner;
  final String ownerNo;
  final String ownerAdd;
  final String vehNo;
  final String vehMod;
  final String vehType;
  final String vehCol;
  final bool approved; // Added field for approval status

  RegistrationRequest({
    required this.owner,
    required this.ownerNo,
    required this.ownerAdd,
    required this.vehNo,
    required this.vehMod,
    required this.vehType,
    required this.vehCol,
    this.approved = false, // Set the default value to false
  });

  Map<String, dynamic> toJson() {
    return {
      'owner': owner,
      'owner_no': ownerNo,
      'owner_add': ownerAdd,
      'veh_no': vehNo,
      'veh_mod': vehMod,
      'veh_type': vehType,
      'veh_col': vehCol,
      'approved': approved, // Include the approved status in the JSON representation
```

```
    };
  }

  static fromJson(Object? data) {}
}

class NewRegistrationPageUser extends StatefulWidget {
  const NewRegistrationPageUser({Key? key}) : super(key: key);

  @override
  NewRegistrationPageUserState createState() => NewRegistrationPageUserState();
}

class NewRegistrationPageUserState extends State<NewRegistrationPageUser> {
  final _formKey = GlobalKey<FormState>();
  final _ownerController = TextEditingController();
  final _ownerNoController = TextEditingController();
  final _ownerAddController = TextEditingController();
  final _vehNoController = TextEditingController();
  final _vehModController = TextEditingController();
  final _vehTypeController = TextEditingController();
  final _vehColController = TextEditingController();

  Future<void> initializeFirebase() async {
    await Firebase.initializeApp();
  }

  Future<void> register(BuildContext context) async {
    if (_formKey.currentState != null && _formKey.currentState!.validate()) {
      final request = RegistrationRequest(
        owner: _ownerController.text,
        ownerNo: _ownerNoController.text,
        ownerAdd: _ownerAddController.text,
        vehNo: _vehNoController.text,
        vehMod: _vehModController.text,
        vehType: _vehTypeController.text,
        vehCol: _vehColController.text,
      );
```

```
    try {
      final CollectionReference requestsCollection =
      FirebaseFirestore.instance.collection('reg_requests');
      await requestsCollection.add(request.toJson());

      // Store the notification string, title, and date in the 'notifications' collection
      const String notificationTitle = 'New Registration Request';
      final String notificationMessage =
          'A new Registration Request has been submitted by ${request.owner}, for Vehicle
Number: ${request.vehNo}';
      final DateTime now = DateTime.now();
      final Timestamp timestamp = Timestamp.fromDate(now); // Convert DateTime to
Timestamp
      final CollectionReference notificationsCollection =
      FirebaseFirestore.instance.collection('notifications');
      await notificationsCollection.add({
        'title': notificationTitle,
        'message': notificationMessage,
        'date': timestamp, // Store the timestamp instead of the formatted date string
      });

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Registration request submitted')),
      );

      _ownerController.clear();
      _ownerNoController.clear();
      _ownerAddController.clear();
      _vehNoController.clear();
      _vehModController.clear();
      _vehTypeController.clear();
      _vehColController.clear();
    } catch (error) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Registration request failed: $error')),
      );
    }
  }
 }
```

```
@override
void initState() {
 super.initState();
 initializeFirebase();
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  backgroundColor: Colors.blue[900],
  appBar: AppBar(
   title: const Text('New Registration Request'),
   toolbarHeight: 83,
   backgroundColor: Colors.blue[900],
   elevation: 0,
   iconTheme: const IconThemeData(
    color: Colors.white, // Set icon color to white
   ),
   toolbarTextStyle: const TextTheme(
    titleLarge: TextStyle(
     color: Colors.white, // Set text color to white
     fontSize: 20,
     fontWeight: FontWeight.bold,
    ),
   ).bodyMedium,
   titleTextStyle: const TextTheme(
    titleLarge: TextStyle(
     color: Colors.white, // Set text color to white
     fontSize: 20,
     fontWeight: FontWeight.bold,
    ),
   ).titleLarge,
  ),
  body: Container(
   decoration: BoxDecoration(
   color: Colors.amber[50],
    borderRadius: const BorderRadius.only(
```

```
    topLeft: Radius.circular(30),
    topRight: Radius.circular(30),
   ),
  ),
  padding: const EdgeInsets.all(20),
  child: SingleChildScrollView(
  child: Form(
    key: _formKey,
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Text(
       'YOUR DETAILS',
       style: TextStyle(
        height: 2,
        fontSize: 18,
        fontWeight: FontWeight.w600,
        color: Colors.grey[400],
       ),
      ),
      const SizedBox(height: 16),
      TextFormField(
       controller: _ownerController,
       decoration: InputDecoration(
       labelText: 'Owner Name',
       border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(10),
       ),
      ),
       validator: (value) {
        if (value == null || value.isEmpty) {
         return 'Please enter your name';
        }
        return null;
       },
      ),
      const SizedBox(height: 16),
      TextFormField(
       controller: _ownerNoController,
```

```
  keyboardType: TextInputType.phone,
  decoration: InputDecoration(
  labelText: 'Owner Phone Number',
  border: OutlineInputBorder(
     borderRadius: BorderRadius.circular(10),
   ),
  ),
  validator: (value) {
   if (value == null || value.isEmpty) {
    return 'Please enter your phone number';
   } else if (value.length != 10) {
    return 'Please enter a valid phone number';
   }
   return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _ownerAddController,
  maxLines: 2,
  decoration:  InputDecoration(
   labelText:  'Owner  Address',
   border: OutlineInputBorder(
    borderRadius: BorderRadius.circular(10),
   ),
  ),
  validator: (value) {
   if (value == null || value.isEmpty) {
    return 'Please enter your address';
   }
   return null;
  },
),
const SizedBox(height: 32),
Text(
  'VEHICLE DETAILS',
  style: TextStyle(
   fontSize: 18,
   fontWeight: FontWeight.w600,
```

```
            color: Colors.grey[400],
          ),
        ),
        const SizedBox(height: 16),
        TextFormField(
          controller: _vehNoController,
          decoration: InputDecoration(
          labelText: 'Vehicle Number',
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
          ),
          ),
          inputFormatters: [
            FilteringTextInputFormatter.allow(RegExp('[A-Z0-9]')), // Only uppercase letters and
numbers are allowed
            TextInputFormatter.withFunction((oldValue, newValue) {
              // Always capitalize the input and remove spaces
              if (oldValue.text != newValue.text) {
                final capitalizedText = newValue.text.replaceAll(' ', '').toUpperCase();
                return TextEditingValue(
                  text: capitalizedText,
                  selection: TextSelection.collapsed(offset: capitalizedText.length),
                );
              }
              return newValue;
            }),
          ],
          validator: (value) {
            if (value == null || value.isEmpty) {
              return 'Please enter vehicle number';
            }
            return null;
          },
        ),
        const SizedBox(height: 16),
        TextFormField(
          controller: _vehModController,
          decoration: InputDecoration(
          labelText: 'Vehicle Model',
```

```
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter vehicle model';
    }
    return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _vehTypeController,
  decoration: InputDecoration(
  labelText: 'Vehicle Type',
  border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter vehicle type';
    }
    return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _vehColController,
  decoration: InputDecoration(
  labelText: 'Vehicle Color',
  border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter vehicle color';
```

```
            }
            return null;
          },
        ),
        const SizedBox(height: 32),
        Center(
          child: SizedBox(
            width: 500,
            height: 50,
            child: ElevatedButton(
              onPressed: () {
              register(context);
              },
              style: ElevatedButton.styleFrom(
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(16),
                ),
                backgroundColor: Colors.amber[100],
              ),
              child: const Text('SUBMIT'),
            ),
          ),
        ),
      ],
    ),
  ),
 ),
 );
 }
}
```

Rules and Regulations Page –

```
import 'package:flutter/material.dart';

class RulesAndRegulations extends StatelessWidget {
  const RulesAndRegulations({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
   return Scaffold(
     backgroundColor: Colors.blue[900],
     appBar: AppBar(
      title: const Text('Rules & Regulations'),
      toolbarHeight: 83,
      backgroundColor: Colors.blue[900],
      elevation: 0,
      iconTheme: const IconThemeData(
       color: Colors.white, // Set icon color to white
      ),
      toolbarTextStyle: const TextTheme(
       titleLarge: TextStyle(
        color: Colors.white, // Set text color to white
        fontSize: 20,
        fontWeight: FontWeight.bold,
       ),
      ).bodyMedium,
      titleTextStyle: const TextTheme(
       titleLarge: TextStyle(
        color: Colors.white, // Set text color to white
        fontSize: 20,
        fontWeight: FontWeight.bold,
       ),
      ).titleLarge,
     ),
     body: Container(
      decoration: BoxDecoration(
      color: Colors.amber[50],
       borderRadius: const BorderRadius.only(
        topLeft: Radius.circular(30),
```

```
    topRight: Radius.circular(30),
  ),
),
child: Padding(
  padding: const EdgeInsets.all(20.0),
  child: SingleChildScrollView(
    child: Column(
      children: [
      const Text(
        'Offences and Penalties',
        style: TextStyle(
        fontSize: 20,
          fontWeight: FontWeight.bold,
        ),
      ),
      const SizedBox(height: 5),
      InteractiveViewer(
        child: Image.asset(
          'assets/offence2022_1.png', // replace with your image asset path
          height: 400, // set image height
          width: double.infinity, // set image width to fill available space
        ),
      ),
      const SizedBox(),
      InteractiveViewer(
      child: Image.asset(
          'assets/offence2022_2.png', // replace with your image asset path
          height: 400, // set image height
          width: double.infinity, // set image width to fill available space
        ),
      ),
      const SizedBox(),
      InteractiveViewer(
      child: Image.asset(
          'assets/offence2022_3.png', // replace with your image asset path
          height: 400, // set image height
          width: double.infinity, // set image width to fill available space
        ),
      ),
```

```
            const SizedBox(),
            InteractiveViewer(
            child: Image.asset(
              'assets/offence2022_4.png', // replace with your image asset path
              height: 400, // set image height
              width: double.infinity, // set image width to fill available space
            ),
            ),
            const SizedBox(),
            InteractiveViewer(
            child: Image.asset(
              'assets/offence2022_5.png', // replace with your image asset path
              height: 400, // set image height
              width: double.infinity, // set image width to fill available space
            ),
            ),
          ],
        ),
      ),
    ),
  ),
 );
 }
}
```

About Page –

```
import 'package:flutter/material.dart';
import 'package:intl/intl.dart';

class About extends StatelessWidget {
  const About({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
   final String copyright =
      '© ${DateFormat('yyyy').format(DateTime.now())} Vehicle Information Identification';

   return Scaffold(
    backgroundColor: Colors.blue[900],
    appBar: AppBar(
     title: const Text('About'),
     toolbarHeight: 83,
     backgroundColor: Colors.blue[900],
     elevation: 0,
     iconTheme: const IconThemeData(
      color: Colors.white, // Set icon color to white
     ),
     toolbarTextStyle: const TextStyle(
      color: Colors.white, // Set text color to white
      fontSize: 20,
      fontWeight: FontWeight.bold,
     ),
     titleTextStyle: const TextStyle(
      color: Colors.white, // Set text color to white
      fontSize: 20,
      fontWeight: FontWeight.bold,
     ),
    ),
    body: Container(
     decoration: BoxDecoration(
     color: Colors.amber[50],
      borderRadius: const BorderRadius.only(
       topLeft: Radius.circular(30),
```

```
          topRight: Radius.circular(30),
        ),
      ),
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: SingleChildScrollView(
          child: Column(
            children: [
              // Add the image here
              Center(
                child: Image.asset(
                  'assets/hatim_logo.png',
                  width: 100,
                  height: 100,
                ),
              ),
              const SizedBox(height: 16),
              const Text(
                'The Vehicle Info app is a powerful tool that allows you to scan vehicle information using your device\'s camera. '
                'It leverages advanced technologies to recognize and extract relevant details from vehicle registration documents. '
                'With this app, you can easily retrieve important information about vehicles, such as the vehicle number, owner details, and more.',
                style: TextStyle(fontSize: 16),
              ),
              const SizedBox(height: 24),
              const Text(
                'Key  Features:',
                style: TextStyle(
                fontSize: 20,
                  fontWeight: FontWeight.bold,
                ),
              ),
              const SizedBox(height: 16),
              const Text(
                '1. Camera Scanning: Use your device\'s camera to capture vehicle registration documents and extract information for further processing.\n\n'
                '2. Text Recognition: The app utilizes Google ML Kit for text recognition, enabling
```

```
accurate extraction of vehicle details from images.\n\n'
            '3. Database Integration: The app seamlessly integrates with a Firestore database, allowing you to store and retrieve vehicle information efficiently.\n\n'
            '4. Real-time Updates: The app fetches data from the Firestore database, ensuring that you have access to the latest vehicle information.\n\n'
            '5. Intuitive UI: The user interface is designed with a futuristic viewfinder and user-friendly controls, providing a seamless and engaging scanning experience.\n\n'
            '6. Editable Results: In case the scanned information needs correction, the app offers an edit screen where you can modify the extracted data before further processing.\n\n'
            '7. Indian Vehicle Codes: The app focuses on Indian vehicles and performs a validation check to ensure the scanned text contains a valid Indian state vehicle code.\n\n'
            '8. Multiple Camera Support: If your device has multiple cameras, you can switch between them to choose the optimal camera for scanning.',
            style: TextStyle(fontSize: 16),
          ),
          const SizedBox(height: 24),
          const Text(
            'Permissions:',
            style: TextStyle(
            fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(height: 16),
          const Text(
            'The Vehicle Info app requires access to your device\'s camera to scan vehicle registration documents. '
            'Rest assured that the app respects your privacy and only accesses the camera for scanning purposes.',
            style: TextStyle(fontSize: 16),
          ),
          const SizedBox(height: 24),
          const Text(
            'Disclaimer:',
            style: TextStyle(
            fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
```

```
const SizedBox(height: 16),
const Text(
 'The information provided by the Vehicle Info app is obtained from publicly available sources and the Firestore database. '
 'While every effort is made to ensure accuracy, the app cannot guarantee the correctness or completeness of the retrieved data. '
 'Please use the app\'s results as a reference and verify the information with the respective authorities if needed.',
 style: TextStyle(fontSize: 16),
),
Column(
 children: [
 const Text(
   'Developers:',
   style: TextStyle(
   height: 2.5,
    fontSize: 20,
    fontWeight: FontWeight.bold,
   ),
  ),
  const SizedBox(height: 16),
  Row(
   children: [
    const CircleAvatar(
     radius: 30,
     backgroundImage: AssetImage('assets/03.png'),
    ),
    const SizedBox(width: 16),
    Expanded(
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: const [
       Text(
        'Lalbiakhluzuala Samte',
        style: TextStyle(
        fontSize: 16,
         fontWeight: FontWeight.bold,
        ),
       ),
```

```
        SizedBox(height: 8),
        Text(
         'Roll No. 2023BCA003\n'
            'Regn No. 2001180\n'
            'VI Semester, BCA\n'
            'HATIM',
         style: TextStyle(fontSize: 14),
        ),
       ],
     ),
    ),
  ],
),
const SizedBox(height: 16),
Row(
 children: [
  const CircleAvatar(
   radius: 30,
   backgroundImage: AssetImage('assets/05.png'),
  ),
  const SizedBox(width: 16),
  Expanded(
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: const [
     Text(
      'Nicholas Beichatlaisa',
      style: TextStyle(
      fontSize: 16,
       fontWeight: FontWeight.bold,
      ),
     ),
     SizedBox(height: 8),
     Text(
      'Roll No. 2023BCA005\n'
         'Regn No. 2001181\n'
         'VI Semester, BCA\n'
         'HATIM',
      style: TextStyle(fontSize: 14),
```

```
          ),
        ],
      ),
    ),
  ],
),
const SizedBox(height: 16),
const Text(
  'Contact Us:',
  style: TextStyle(
  height: 2.5,
    fontSize: 20,
    fontWeight: FontWeight.bold,
  ),
),
const SizedBox(height: 8),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Expanded(
      child: Column(
        children: const [
          Icon(Icons.phone),
          SizedBox(height: 8),
          Text(
            'Phone',
            style: TextStyle(fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 4),
          Text('9366006092\n'
              '9366262436', style: TextStyle(fontSize: 12)),
        ],
      ),
    ),
    Expanded(
      child: Column(
        children: const [
          Icon(Icons.email),
          SizedBox(height: 8),
```

```
                    Text(
                      'Email',
                      style: TextStyle(fontWeight: FontWeight.bold),
                    ),
                    SizedBox(height: 4),
                    Text('hatimdev@gmail.com', style: TextStyle(fontSize: 12)),
                  ],
                ),
              ),
              Expanded(
                child: Column(
                  children: const [
                    Icon(Icons.location_on),
                    SizedBox(height: 8),
                    Text(
                      'Address',
                      style: TextStyle(fontWeight: FontWeight.bold),
                    ),
                    SizedBox(height: 4),
                    Text('Pukpui,\n'
                        'Lunglei - 796691', style: TextStyle(fontSize: 12)),
                  ],),),],),],),],),),),),),
    bottomNavigationBar: BottomAppBar(
      color: Colors.amber[50],
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              copyright,
              style: const TextStyle(fontSize: 16.0, color: Colors.grey),
            ),
            const SizedBox(
              width: 5.0,
            ),],),),), );
  }}
```

Admin Login Page –

```
import 'dart:async';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

import 'admin_page.dart';

class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  LoginPageState createState() => LoginPageState();
}

class LoginPageState extends State<LoginPage> {
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  bool _isLoading = false;
  bool _passwordVisible = false; // Added state for password visibility

  Future<void> _login(BuildContext context) async {
    setState(() {
      _isLoading = true;
    });

    try {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: _emailController.text.trim(),
        password: _passwordController.text,
      );

      Timer(const Duration(seconds: 1), () {
        Navigator.of(context).pushReplacement(
        MaterialPageRoute(builder: (_) => AdminPage(role: 'Admin')),
        );
      });

      showDialog(
```

```
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Login Successful'),
        content: Row(
          children: const [
            Icon(Icons.check, color: Colors.green),
            SizedBox(width: 8),
            Text('Login Successful'),
          ],
        ),
      ),
    );
  } on FirebaseAuthException catch (e) {
    showDialog(
      context: context,
      builder: (_) => AlertDialog(
        title: const Text('Login Failed'),
        content: Text(e.message ?? 'Unknown error occurred.'),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: const Text('OK'),
          ),
        ],
      ),
    );
  }

  setState(() {
    _isLoading = false;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.amber[50],
```

```
body: SafeArea(
  child: Padding(
    padding: const EdgeInsets.symmetric(horizontal: 32),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Expanded(
          child: Center(
            child: Image.asset(
              'assets/splash_image.png',
              height: MediaQuery.of(context).size.height * 0.3,
            ),
          ),
        ),
        const SizedBox(height: 30),
        const Text(
          'ADMIN LOGIN',
          style: TextStyle(
            color: Colors.black,
            fontWeight: FontWeight.bold,
            fontSize: 24,
          ),
        ),
        const SizedBox(height: 20),
        TextFormField(
          style: const TextStyle(color: Colors.black),
          decoration: InputDecoration(
            labelText: 'Email',
            labelStyle: const TextStyle(color: Colors.grey),
            border: OutlineInputBorder(
              borderSide: const BorderSide(
                color: Colors.white,
                width: 2,
              ),
              borderRadius: BorderRadius.circular(16),
            ),
          ),
          controller: _emailController,
        ),
```

```
const SizedBox(height: 20),
Stack(
 alignment: Alignment.centerRight,
 children: [
  TextFormField(
   style: const TextStyle(color: Colors.black),
   decoration: InputDecoration(
    labelText: 'Password',
    labelStyle: const TextStyle(color: Colors.grey),
    border: OutlineInputBorder(
     borderSide: const BorderSide(
      color: Colors.white,
      width: 2,
     ),
     borderRadius: BorderRadius.circular(16),
    ),
   ),
   obscureText: !_passwordVisible, // Use the visibility state
   controller: _passwordController,
  ),
  IconButton(
   icon: Icon(
    _passwordVisible ? Icons.visibility : Icons.visibility_off,
   ),
   onPressed: () {
    setState(() {
     _passwordVisible = !_passwordVisible; // Toggle visibility state
    });
   },
  ),
 ],
),
const SizedBox(height: 20),
Container(
 width: double.infinity,
 decoration: BoxDecoration(
  borderRadius: BorderRadius.circular(20),
  gradient: const LinearGradient(
   colors: [
```

```
              Color(0xFF5AFFD6),
              Color(0xFF00C6FF),
            ],
          ),
        ),
        child: ElevatedButton(
          onPressed: _isLoading ? null : () => _login(context),
          style: ElevatedButton.styleFrom(
          backgroundColor: Colors.grey[700],
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(20),
            ),
            elevation: 5,
            padding: const EdgeInsets.symmetric(vertical: 16),
          ),
          child: _isLoading
              ? const SizedBox(
                  height: 24,
                  width: 24,
                  child: CircularProgressIndicator(
                    valueColor:
                        AlwaysStoppedAnimation<Color>(Colors.white),
                  ),)
              : const Text(
                  'Login',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: 20,
                    fontWeight: FontWeight.bold,
                  ),
                ),
          ),
        ),
        const SizedBox(height: 20),
      ],
    ),),),);
}}
```

Admin Panel –

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';

import 'package:vehicle_info/reg_approval_page.dart';
import 'package:vehicle_info/registered_docs.dart';
import 'new_reg_admin.dart';

class AdminPage extends StatelessWidget {
  const AdminPage({Key? key, required this.role}) : super(key: key);

  final String role;

  @override
  Widget build(BuildContext context) {
   final buttonWidth = MediaQuery.of(context).size.width * 0.9;

   return Scaffold(
     appBar: AppBar(
       backgroundColor: Colors.blue[900],
       title: const Text('Admin Panel'),
       foregroundColor: Colors.amber[50],
       leading: IconButton(
         icon: const Icon(Icons.logout),
         onPressed: () {
           // Navigate to the Home Screen
           Navigator.popUntil(context, (route) => route.isFirst);
         },
       ),
     ),
     body: Stack(
       children: [
       Positioned.fill(
          child: Image.asset(
            'assets/background_image.png',
            fit: BoxFit.cover,
          ),
```

```
),
Column(
 crossAxisAlignment: CrossAxisAlignment.center,
 children: [
  Container(
   decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(8.0),
    border: Border.all(color: Colors.grey.shade300, width: 1.0),
   ),
   padding: const EdgeInsets.all(16.0),
   margin: const EdgeInsets.symmetric(
       horizontal: 16.0, vertical: 16.0),
   child: FutureBuilder<int>(
    future: getDocumentCount('veh_info'),
    // Fetch document count for 'veh_info' collection
    builder: (context, snapshot) {
     if (snapshot.connectionState == ConnectionState.waiting) {
      return const CircularProgressIndicator();
     } else if (snapshot.hasError) {
      return Text('Error: ${snapshot.error}');
     } else {
      final int documentCount = snapshot.data ?? 0;
      return FutureBuilder<int>(
       future: getDocumentCount('reg_requests'),
       // Fetch registration requests count
       builder: (context, snapshot) {
        if (snapshot.connectionState ==
            ConnectionState.waiting) {
         return const CircularProgressIndicator();
        } else if (snapshot.hasError) {
         return Text('Error: ${snapshot.error}');
        } else {
         final int registrationRequestsCount =
             snapshot.data ?? 0;

         return Column(
          children: [
          ListTile(
```

```
                title: const Text('Documents Registered:'),
                trailing: Text('$documentCount'),
              ),
              const Divider(),
              ListTile(
                title: const Text('Registration Requests:'),
                trailing: Text('$registrationRequestsCount'),
              ),
            ],
          );
        }
      },
    );
  }
 },
),
SizedBox(
 width: buttonWidth,
 height: 50,
 child: ElevatedButton(
  onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(
     builder: (context) => const RegisteredDocumentPage(),
    ),
  );
  },
  style: ButtonStyle(
   backgroundColor: MaterialStateProperty.all<Color>(Colors.amber[100]!),
  ),
  child: Row(
   mainAxisAlignment: MainAxisAlignment.start,
   children: const [
    Icon(Icons.check_box),
    SizedBox(width: 8.0),
    Text('REGISTERED DOCUMENTS'),
   ],
```

```
    ),
  ),
),
const SizedBox(height: 16.0),
SizedBox(
  width: buttonWidth,
  height: 50,
  child: ElevatedButton(
    onPressed: () {
      // Navigate to New Registration page
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => const NewRegistrationPageAdmin(),
        ),
      );
    },
    style: ButtonStyle(
      backgroundColor: MaterialStateProperty.all<Color>(Colors.amber[100]!),
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.start,
      children: const [
        Icon(Icons.add),
        SizedBox(width: 8.0),
        Text('NEW REGISTRATION'),
      ],
    ),
  ),
),
const SizedBox(height: 16.0),
SizedBox(
  width: buttonWidth,
  height: 50,
  child: ElevatedButton(
    onPressed: () {
    Navigator.push(
        context,
        MaterialPageRoute(
```

```
                builder: (context) => const RegistrationApprovalPage(),
              ),
            );
          },
          style: ButtonStyle(
            backgroundColor: MaterialStateProperty.all<Color>(Colors.amber[100]!),
          ),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.start,
            children: const [
              Icon(Icons.approval),
              SizedBox(width: 8.0),
              Text('REGISTRATION REQUESTS'),
            ],
          ),
        ),
      ),

    ],
   ),
  ],
 ),
);
}

Future<int> getDocumentCount(String collectionName) async {
 final snapshot =
    await FirebaseFirestore.instance.collection(collectionName).get();
 return snapshot.docs.length;
}
}
```

[Admin] Registered Documents Page –

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class RegisteredDocumentPage extends StatefulWidget {
  const RegisteredDocumentPage({Key? key}) : super(key: key);

  @override
  RegisteredDocumentPageState createState() => RegisteredDocumentPageState();
}

class RegisteredDocumentPageState extends State<RegisteredDocumentPage> {
  final ownerController = TextEditingController();
  final ownerNoController = TextEditingController();
  final ownerAddController = TextEditingController();
  final vehNoController = TextEditingController();
  final vehModController = TextEditingController();
  final vehTypeController = TextEditingController();
  final vehColController = TextEditingController();

  @override
  void dispose() {
    // Dispose the controllers when the widget is disposed
    ownerController.dispose();
    ownerNoController.dispose();
    ownerAddController.dispose();
    vehNoController.dispose();
    vehModController.dispose();
    vehTypeController.dispose();
    vehColController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blue[900],
```

```
  foregroundColor: Colors.white,
  title: const Text('Registered Documents'),
 ),
 body: StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance.collection('veh_info').snapshots(),
  builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
  if (snapshot.hasError) {
     return Center(child: Text('Error: ${snapshot.error}'));
   }

   if (snapshot.connectionState == ConnectionState.waiting) {
    return const Center(child: CircularProgressIndicator());
   }

   final documents = snapshot.data?.docs ?? [];

   return ListView.builder(
    itemCount: documents.length,
    itemBuilder: (BuildContext context, int index) {
     final data = documents[index].data() as Map<String, dynamic>?;

     // Check if data is not null before accessing its fields
     final owner = data?['owner'] ?? '';
     final ownerNo = data?['owner_no'] ?? '';
     final ownerAdd = data?['owner_add'] ?? '';
     final vehNo = data?['veh_no'] ?? '';
     final vehMod = data?['veh_mod'] ?? '';
     final vehType = data?['veh_type'] ?? '';
     final vehCol = data?['veh_col'] ?? '';

     ownerController.text = owner;
     ownerNoController.text = ownerNo;
     ownerAddController.text = ownerAdd;
     vehNoController.text = vehNo;
     vehModController.text = vehMod;
     vehTypeController.text = vehType;
     vehColController.text = vehCol;

     return Card(
```

```
elevation: 2,
margin: const EdgeInsets.all(10.0),
child: Padding(
 padding: const EdgeInsets.all(20.0),
 child: Row(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   Expanded(
    flex: 1,
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Text(
       vehNo,
       style: const TextStyle(
        fontSize: 20,
        fontWeight: FontWeight.bold,
       ),
      ),
      const SizedBox(height: 8.0),
      Text(
       owner,
       style: const TextStyle(fontSize: 18, height: 1.5),
      ),
      Text(
       ownerNo,
       style: const TextStyle(fontSize: 18, height: 1.5),
      ),
      Text(
       ownerAdd,
       style: const TextStyle(fontSize: 18, height: 1.5),
      ),
      Text(
       vehMod,
       style: const TextStyle(fontSize: 18, height: 1.5),
      ),
      Text(
       vehType,
       style: const TextStyle(fontSize: 18, height: 1.5),
```

```
      ),
      Text(
        vehCol,
        style: const TextStyle(fontSize: 18, height: 1.5),
      ),
    ],
  ),
),
const SizedBox(width: 16.0),
SizedBox(
  width: 50.0,
  child: Column(
  children: [
    IconButton(
      onPressed: () {
      showDialog(
        context: context,
        builder: (BuildContext context) {
          // Set the initial values of the controllers
          ownerController.text = owner;
          ownerNoController.text = ownerNo;
          ownerAddController.text = ownerAdd;
          vehNoController.text = vehNo;
          vehModController.text = vehMod;
          vehTypeController.text = vehType;
          vehColController.text = vehCol;

          return AlertDialog(
            title: const Text('Edit Document'),
            content: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                TextFormField(
                  decoration: const InputDecoration(
                    labelText: 'Owner',
                  ),
                  controller: ownerController,
                  textInputAction: TextInputAction.next,
```

```
  onChanged: (value) {
    // Handle owner field changes
  },
  onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
),
TextFormField(
  decoration: const InputDecoration(
    labelText: 'Owner Contact',
  ),
  controller: ownerNoController,
  textInputAction: TextInputAction.next,
  onChanged: (value) {
    // Handle ownerNo field changes
  },
  onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
),
TextFormField(
  decoration: const InputDecoration(
    labelText: 'Owner Address',
  ),
  controller: ownerAddController,
  textInputAction: TextInputAction.next,
  onChanged: (value) {
    // Handle ownerAdd field changes
  },
  onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
),
TextFormField(
  decoration: const InputDecoration(
    labelText: 'Vehicle No',
  ),
  controller: vehNoController,
  textInputAction: TextInputAction.next,
  onChanged: (value) {
    // Handle vehNo field changes
  },
  onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
),
TextFormField(
```

```
          decoration: const InputDecoration(
            labelText: 'Vehicle Model',
          ),
          controller: vehModController,
          textInputAction: TextInputAction.next,
          onChanged: (value) {
            // Handle vehMod field changes
          },
          onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
        ),
        TextFormField(
          decoration: const InputDecoration(
            labelText: 'Vehicle Type',
          ),
          controller: vehTypeController,
          textInputAction: TextInputAction.next,
          onChanged: (value) {
            // Handle vehType field changes
          },
          onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
        ),
        TextFormField(
          decoration: const InputDecoration(
            labelText: 'Vehicle Color',
          ),
          controller: vehColController,
          textInputAction: TextInputAction.done,
          onChanged: (value) {
            // Handle vehCol field changes
          },
          onFieldSubmitted: (_) => Navigator.of(context).pop(),
        ),
      ],
    ),
  ),
  actions: [
    TextButton(
      onPressed: () async {
        // Handle save button press
```

```
    final newData = {
      'owner': ownerController.text,
      'owner_no': ownerNoController.text,
      'owner_add': ownerAddController.text,
      'veh_no': vehNoController.text,
      'veh_mod': vehModController.text,
      'veh_type': vehTypeController.text,
      'veh_col': vehColController.text,
    };

    try {
      final docId = documents[index].id;
      await FirebaseFirestore.instance
          .collection('veh_info')
          .doc(docId)
          .update(newData);

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Document updated successfully.'),
        ),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
          content: Text('Failed to update document.'),
        ),
      );
    }

    Navigator.of(context).pop();
  },
  child: const Text('Save'),
),
TextButton(
  onPressed: () {
    // Handle cancel button press
    Navigator.of(context).pop();
  },
```

```
                    child: const Text('Cancel'),
                  ),
                ],
              );
            },
          );
        },
        icon: const Icon(Icons.edit),
      ),
      IconButton(
        onPressed: () async {
        try {
          final docId = documents[index].id;
          await FirebaseFirestore.instance
              .collection('veh_info')
              .doc(docId)
              .delete();

          ScaffoldMessenger.of(context).showSnackBar(
            const SnackBar(
              content: Text(
                  'Document deleted successfully.'),
            ),
          );
        } catch (e) {
          ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(
              content:
                  Text('Failed to delete document.'),
            ),
          );
        }},
        icon: const Icon(Icons.delete),
      ),],),),],), ),);
    },);
  },),);
}}
```

[Admin] New Document Registration Page –

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:intl/intl.dart';

class NewRegistrationPageAdmin extends StatefulWidget {
  const NewRegistrationPageAdmin({super.key});
  @override
  NewRegistrationPageAdminState createState() =>
      NewRegistrationPageAdminState();
}

class NewRegistrationPageAdminState extends State<NewRegistrationPageAdmin> {
  final TextEditingController ownerController = TextEditingController();
  final TextEditingController ownerNoController = TextEditingController();
  final TextEditingController ownerAddController =TextEditingController();
  final TextEditingController vehNoController = TextEditingController();
  final TextEditingController vehModController = TextEditingController();
  final TextEditingController vehTypeController = TextEditingController();
  final TextEditingController vehColController = TextEditingController();

  @override
  void initState() {
    super.initState();

    // Attach a listener to the controller
    vehNoController.addListener(_capitalizeInput);
  }

  @override
  void dispose() {
    // Clean up the controller
    vehNoController.dispose();
    super.dispose();
  }

  void _capitalizeInput() {
```

```
  final text = vehNoController.text;
  final newText = text.replaceAll(' ', '').toUpperCase();

  // Only update the controller's text if it has changed
  if (text != newText) {
   vehNoController.value = TextEditingValue(
    text: newText,
    selection: TextSelection.fromPosition(
     TextPosition(offset: newText.length),
    ),
   );
  }
 }

 Future<void> _register(BuildContext context) async {
  final CollectionReference vehInfoCollection =
  FirebaseFirestore.instance.collection('veh_info');
  final CollectionReference notificationsCollection =
  FirebaseFirestore.instance.collection('notifications');
  final DateTime now = DateTime.now();
  final Timestamp timestamp = Timestamp.fromDate(now); // Convert DateTime to Timestamp
  final String formattedDate =
  DateFormat('yyyy-MM-dd').format(now); // Format the date as desired

  try {
   DocumentReference newDocRef = await vehInfoCollection.add({
    'owner': ownerController.text,
    'owner_no': ownerNoController.text,
    'owner_add': ownerAddController.text,
    'veh_no': vehNoController.text,
    'veh_mod': vehModController.text,
    'veh_type': vehTypeController.text,
    'veh_col': vehColController.text,
   });

   String notificationMessage =
     'A new document of ${ownerController.text} for Vehicle Number: ${vehNoController.text}
has been registered.';
```

```
    await notificationsCollection.add({
      'title': 'New Document Registered',
      'message': notificationMessage,
      'date': timestamp, // Store the timestamp instead of the formatted date string
    });

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Registration Successful')),
    );

    // Clear input fields
    ownerController.clear();
    ownerNoController.clear();
    ownerAddController.clear();
    vehNoController.clear();
    vehModController.clear();
    vehTypeController.clear();
    vehColController.clear();
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Registration Failed: $e')),
    );
  }
}

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      foregroundColor: Colors.white,
      backgroundColor: Colors.blue[900],
      title: const Text('New Registration'),
    ),
    body: SingleChildScrollView(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const SizedBox(height: 16.0),
```

```
Row(
  children: const [
    Icon(Icons.person),
    SizedBox(width: 8.0),
    Text(
      'OWNER DETAILS',
      style: TextStyle(
        fontSize: 18.0,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
),
const SizedBox(height: 16.0),
TextField(
  decoration: const InputDecoration(
    labelText: 'Full Name',
  ),
  controller: ownerController,
),
const SizedBox(height: 16.0),
TextField(
  decoration: const InputDecoration(
    labelText: 'Phone Number',
  ),
  controller: ownerNoController,
),
const SizedBox(height: 16.0),
TextField(
  decoration: const InputDecoration(
    labelText: 'Address',
  ),
  controller: ownerAddController,
),
const SizedBox(height: 16.0),
Row(
  children: const [
    Icon(Icons.directions_car),
    SizedBox(width: 8.0),
```

```
    Text(
     'VEHICLE DETAILS',
     style: TextStyle(
      fontSize: 18.0,
      fontWeight: FontWeight.bold,
     ),
    ),
   ],
  ),
  SizedBox(height: 16.0),
  TextField(
   decoration: const InputDecoration(
    labelText: 'Vehicle Number',
   ),
   controller: vehNoController,
   inputFormatters: [
    FilteringTextInputFormatter.deny(' '), // Disable blank spaces
   ],
  ),
  const SizedBox(height: 16.0),
  TextField(
   decoration: const InputDecoration(
    labelText: 'Vehicle Model',
   ),
   controller: vehModController,
  ),
  const SizedBox(height: 16.0),
  TextField(
   decoration: const InputDecoration(
    labelText: 'Vehicle Type',
   ),
   controller: vehTypeController,
  ),
  const SizedBox(height: 16.0),
  TextField(
   decoration: const InputDecoration(
    labelText: 'Vehicle Color',
   ),
   controller: vehColController,
```

```
          ),
          const SizedBox(height: 32.0),
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () => _register(context),
              style: ElevatedButton.styleFrom(
              backgroundColor: Colors.amber[100],
                padding: const EdgeInsets.symmetric(vertical: 16.0),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(10.0),
                ),
              ),
              child: const Text('REGISTER'),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

[Admin] Registration Requests Page –

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'registration_request.dart';
import 'package:intl/intl.dart';

class RegistrationApprovalPage extends StatefulWidget {
  const RegistrationApprovalPage({Key? key}) : super(key: key);

  @override
  RegistrationApprovalPageState createState() =>
      RegistrationApprovalPageState();
}

class RegistrationApprovalPageState extends State<RegistrationApprovalPage> {
  final _scaffoldKey = GlobalKey<ScaffoldState>();

  Future<void> storeRejectionNotification(String owner, String vehNo) async {
    final DateTime now = DateTime.now();
    final Timestamp timestamp = Timestamp.fromDate(now); // Convert DateTime to Timestamp
    final String formattedDate =
    DateFormat('yyyy-MM-dd').format(now); // Format the date as desired
    final notification = 'Registration request for $owner, Vehicle Number: $vehNo has been
rejected.';
    await FirebaseFirestore.instance.collection('notifications').add({
      'title': 'Registration Request Rejected',
      'message': notification,
      'date': timestamp,
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blue[900],
        foregroundColor: Colors.white,
        title: const Text('Registration Requests'),
```

```
),
body: StreamBuilder<QuerySnapshot>(
 stream:
 FirebaseFirestore.instance.collection('reg_requests').snapshots(),
 builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot) {
  if (snapshot.hasError) {
   return const Center(
    child: Text('Something went wrong'),
   );
  }

  if (snapshot.connectionState == ConnectionState.waiting) {
   return const Center(
    child: CircularProgressIndicator(),
   );
  }

  final List<RegistrationRequest> requests = snapshot.data!.docs
     .map((DocumentSnapshot document) => RegistrationRequest.fromJson(
     document.data() as Map<String, dynamic>))
     .toList();

  if (requests.isEmpty) {
   return Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: const [
      Icon(
       Icons.check_circle_outline,
       size: 100,
       color: Colors.grey,
      ),
      SizedBox(height: 16.0),
      Text(
       'No new requests',
       style: TextStyle(fontSize: 20.0),
      ),
     ],
    ),
```

```
  );
}

return ListView.builder(
  itemCount: requests.length,
  itemBuilder: (BuildContext context, int index) {
    final RegistrationRequest request = requests[index];

    return InkWell(
      onTap: () {
      Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => RegistrationDetailsPage(request: request),
          ),
        );
      },
      child: Card(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                request.owner,
                style: const TextStyle(
                    fontSize: 18.0, fontWeight: FontWeight.bold),
              ),
              const SizedBox(height: 8.0),
              Text(
                '${request.vehNo} - ${request.vehMod}',
                style: const TextStyle(fontSize: 16.0),
              ),
              const SizedBox(height: 16.0),
              Row(
                mainAxisAlignment: MainAxisAlignment.end,
                children: [
                  ElevatedButton(
                    onPressed: () async {
```

```
                    // Move the document to the 'veh_info' collection
                    final DocumentReference vehInfoDocRef =
                    await FirebaseFirestore.instance
                       .collection('veh_info')
                       .add(request.toJson());

                    // Delete the request from the 'reg_requests' collection
                    final QuerySnapshot requestQuerySnapshot =
                    await FirebaseFirestore.instance
                       .collection('reg_requests')
                       .where('veh_no', isEqualTo: request.vehNo)
                       .limit(1)
                       .get();

                    if (requestQuerySnapshot.docs.isNotEmpty) {
                     await requestQuerySnapshot.docs.first.reference.delete();
                    }

                    // Store the notification in the 'notifications' collection
                    final String notificationMessage =
                       'Registration request for ${request.owner}, Vehicle Number:
${request.vehNo} has been approved and is registered successfully.';
                    final DateTime now = DateTime.now();
                    final Timestamp timestamp = Timestamp.fromDate(now); // Convert
DateTime to Timestamp
                    final String formattedDate =
                    DateFormat('yyyy-MM-dd').format(now); // Format the date as desired

                    await FirebaseFirestore.instance.collection('notifications').add({
                     'title': 'Registration Request Approved',
                     'message': notificationMessage,
                     'date': timestamp,
                    });

                    // Show a success message
                    ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(
                       content: const Text('Request approved'),
                       action: SnackBarAction(
```

```
                        label: 'UNDO',
                        onPressed: () {
                          // Undo the approval by deleting the newly created document in
'veh_info'
                          vehInfoDocRef.delete();
                        },
                      ),
                    ),
                  );
                },
                style: ButtonStyle(
                  backgroundColor: MaterialStateProperty.all<Color>(Colors.amber[100]!),
                ),
                child: const Text('Approve'),
              ),
              const SizedBox(width: 8.0),
              OutlinedButton(
              onPressed: () async {
                // Store rejection notification
                await storeRejectionNotification(request.owner, request.vehNo);

                // Delete the document from the 'reg_requests' collection
                await FirebaseFirestore.instance
                    .collection('reg_requests')
                    .doc(snapshot.data!.docs[index].id)
                    .delete();

                // Show a success message
                ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                    content: Text('Request rejected'),
                  ),
                );
              },
              child: const Text('Reject'),
            ),
          ],
        ),
      ],
```

```
                ),
              ),
            ),
          );

        },
      );
    },
  ),
 );
 }
}

class RegistrationDetailsPage extends StatelessWidget {
 final RegistrationRequest request;

 const RegistrationDetailsPage({Key? key, required this.request})
   : super(key: key);

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    backgroundColor: Colors.blue[900],
    foregroundColor: Colors.white,
    title: const Text('Registration Details'),
   ),
   body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      const SizedBox(height: 16.0),
      const Text(
       'Owner Information',
       style: TextStyle(
       fontSize: 20.0,
        fontWeight: FontWeight.bold,
        color: Colors.blue,
```

```
          ),),
        const SizedBox(height: 16.0),
        _buildInfoRow('Name', request.owner),
        _buildInfoRow('Contact Number', request.ownerNo),
        _buildInfoRow('Address', request.ownerAdd),
        const SizedBox(height: 24.0),
        const Text(
          'Vehicle Information',
          style: TextStyle(
          fontSize: 20.0,
            fontWeight: FontWeight.bold,
            color: Colors.blue,
          ),),
        const SizedBox(height: 16.0),
        _buildInfoRow('Number', request.vehNo),
        _buildInfoRow('Model', request.vehMod),
        _buildInfoRow('Type', request.vehType),
        _buildInfoRow('Color', request.vehCol),
      ],),),);
}

 Widget _buildInfoRow(String label, String value) {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        label,
        style: const TextStyle(fontSize: 16.0, fontWeight: FontWeight.bold),
      ),
      const SizedBox(height: 4.0),
      Text(
        value,
        style: const TextStyle(fontSize: 16.0),
      ),
      const SizedBox(height: 16.0),
    ],);
 }}
```

Requests Factory –

```
class RegistrationRequest {
  final String owner;
  final String ownerNo;
  final String ownerAdd;
  final String vehNo;
  final String vehMod;
  final String vehType;
  final String vehCol;

  RegistrationRequest({
    required this.owner,
    required this.ownerNo,
    required this.ownerAdd,
    required this.vehNo,
    required this.vehMod,
    required this.vehType,
    required this.vehCol,
  });

  factory RegistrationRequest.fromJson(Map<String, dynamic> json) {
    return RegistrationRequest(
      owner: json['owner'],
      ownerNo: json['owner_no'],
      ownerAdd: json['owner_add'],
      vehNo: json['veh_no'],
      vehMod: json['veh_mod'],
      vehType: json['veh_type'],
      vehCol: json['veh_col'],
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'owner': owner,
      'owner_no': ownerNo,
      'owner_add': ownerAdd,
      'veh_no': vehNo,
```

```
    'veh_mod': vehMod,
    'veh_type': vehType,
    'veh_col': vehCol,
   };
  }
}
```

Description of Coding Segments

Main –

Here's a description of the code and its segments:

1. Import Statements:
    - These statements import the necessary packages and files required for the code to work correctly. They provide access to functionalities like camera handling, text recognition, Firebase integration, UI widgets, and permissions.
2. **main()** function:
    - This function is the entry point of the application.
    - It ensures that Flutter is initialized and then initializes Firebase using the default options for the current platform.
    - Finally, it runs the application by passing the **App** widget as the root widget.
3. **fetchData()** function:
    - This function fetches data from the Firestore collection named "veh_info" and prints the data if the application is in debug mode (**kDebugMode**).

4. **App** class:
    - This class represents the root of the application and extends **StatelessWidget**.
    - It configures the **MaterialApp** widget, which defines the overall theme, title, and initial route of the application.
    - The **home** property is set to **SplashScreen()**, which is likely a custom widget representing the splash screen of the application.
5. **MainScreen** class:
    - This class represents the main screen of the application and extends **StatefulWidget**.
    - It manages the camera and text recognition functionality.
    - It also includes the **WidgetsBindingObserver** mixin to observe changes in the application's lifecycle.
6. **_MainScreenState** class:
    - This class represents the state associated with the **MainScreen** widget and extends **State**.
    - It contains lifecycle-related methods like **initState()** and **dispose()** to handle the initialization and disposal of camera resources.
    - The **didChangeAppLifecycleState()** method is overridden to start and stop the camera based on changes in the application's lifecycle.
    - The class also manages camera and flash control, including toggling the flash and rotating the camera.
    - The **_scanImage()** method captures an image from the camera, processes it using text recognition, and navigates to the result screen based on the recognized text.

Splash Screen –

Here's a breakdown of what the code does:

1. It imports the necessary dependencies from the **flutter/material.dart** library and the **home_screen.dart** file.
2. The **SplashScreen** class is defined as a **StatefulWidget** that represents the splash screen.
3. The **SplashScreen** class overrides the **createState** method to create an instance of the **_SplashScreenState** class, which manages the state of the splash screen.
4. The **_SplashScreenState** class is defined as a **State** that extends the **SplashScreen** state.
5. Inside the **_SplashScreenState** class, there is an **initState** method that is automatically called when the state is initialized. In this method, a **Future.delayed** function is used to wait for 1 second and then set the **opacityLevel** variable to 1.0. This causes the image in the splash screen to fade in.
6. Another **Future.delayed** function is used to wait for 3 seconds after the opacity is set. Once the 3 seconds have passed, the **Navigator** is used to push the **HomeScreen** onto the navigation stack, replacing the current screen. The **HomeScreen** is instantiated with a role parameter set to 'Home'.
7. The **build** method is overridden to build the UI for the splash screen. It returns a **Scaffold** widget with a background color set to **Colors.amber[50]**. The body of the **Scaffold** contains a **Center** widget, which centers its child vertically and horizontally.
8. Inside the **Center** widget, there is an **AnimatedOpacity** widget. This widget animates the opacity of its child based on the value of the **opacityLevel** variable.

Home –

Here's a breakdown of what the code does:

1. It imports the necessary dependencies from various packages and files.
2. The **HomeScreen** class is defined as a **StatefulWidget** that represents the home screen of the application. It takes a **role** parameter as input.
3. The **HomeScreen** class overrides the **createState** method to create an instance of the **_HomeScreenState** class, which manages the state of the home screen.
4. The **_HomeScreenState** class is defined as a **State** that extends the **HomeScreen** state.
5. Inside the **_HomeScreenState** class, there is a **notifications** list that holds notification data and a **hasNewUpdates** variable to track whether there are new updates.
6. The **fetchNotifications** function is implemented to fetch notifications from the Firestore database using the **cloud_firestore** package.
7. The **checkForUpdates** function is implemented to check whether there are new updates in the notifications list.
8. The **build** method is overridden to build the UI for the home screen. It sets up an **AppBar** with title and action buttons.
9. The body of the **Scaffold** contains a container with a background image. Inside the container, there's a **ColorFiltered** widget to apply a color filter on top of the background image.
10. Inside the **ColorFiltered** widget, there's a container with a blue background color. Within this container, several **ElevatedButton** widgets are arranged to represent different functionalities of the home screen.
11. There's an expanded container at the bottom, which represents the notification section. It displays the notifications fetched from the Firestore

database. The **RefreshIndicator** widget is used to implement pull-to-refresh functionality for the notifications.

12. The **FutureBuilder** widget is used to handle the asynchronous fetching of notifications. It displays a loading spinner while waiting for the data, shows an error message if there's an error, and builds the list of notifications once the data is available.

13. The **getTrailingIcon** function is implemented to determine the appropriate trailing icon based on the title of a notification. It returns the corresponding **IconData**.

Edit Screen –

Here's a breakdown of the code:

1. The **EditScreen** class is a stateful widget that extends **StatefulWidget**. It takes two parameters: **text** (nullable String) and **notNullString** (required String). The **text** parameter is used to initialize the **TextField** with an initial value, and the **notNullString** parameter ensures that **myString** is not null. It also overrides the **createState** method to create the corresponding state object.

2. The **EditScreenState** class is the state class for the **EditScreen** widget. It initializes a **TextEditingController** and sets its initial value based on the **text** parameter. The **TextEditingController** is responsible for managing the text field's input and providing its current value.

3. The **initState** method is overridden to initialize the **TextEditingController** and set its initial value.

4. The **dispose** method is overridden to dispose of the **TextEditingController** when the widget is removed from the widget tree.

5. The **build** method constructs the dialog UI using a **Dialog** widget. It contains a title, a text field for editing the text, and a search button.

6. The **AlphaNumericInputFormatter** class is a custom **TextInputFormatter** that allows only alphanumeric characters in the text field. It uses a regular expression to filter out non-alphanumeric characters.

Result Screen –

Here's a breakdown of the code:

1. The **ResultScreen** class is a stateless widget that takes a **text** parameter, representing the search query, and a **data** parameter, which is a list of dynamic objects. The **data** parameter is not used within the widget.
2. The **build** method constructs the UI for the result screen using a **Scaffold** widget. It includes an app bar and a body.
3. The app bar has a title, a custom height, a blue color background, and white text color.
4. The body is a **Container** widget with a decoration that gives it a rounded border at the top and sets its color to amber.
5. Inside the body, a **StreamBuilder** widget is used to listen to the query snapshot stream from Firestore. The stream is created by querying the 'veh_info' collection where the 'veh_no' field is equal to the provided **text**.
6. Depending on the snapshot state and data, the builder function renders different UI components:
    - If there's an error in the snapshot, it shows an error message.
    - If there's no data or the query results are empty, it shows a 'No matches found' message.
    - If there are query results, it constructs a column with the scanned text, a divider, and a **ListView.builder** to display the search results.

7. The **ListView.builder** builds a list of cards, where each card represents a document from the query results. It extracts the necessary data fields from the document and displays them in a column layout.

Manual Search Screen –

Here's a breakdown of the code:

1. The **ManualSearchScreen** class is a stateful widget that represents the screen where users can manually search for vehicles.
2. The **ManualSearchScreenState** class extends **State** and manages the state for the **ManualSearchScreen** widget.
3. The **fetchData** method fetches data from the Firestore collection 'veh_info' and stores it in the **_dataList** and **_filteredDataList** variables.
4. The **_filterDataList** method filters the data list based on a search text. It updates the **_filteredDataList** variable with the filtered results.
5. The **_onListItemTap** method is called when a list item is tapped. It navigates to the **DetailedInformationPage** passing the corresponding **DocumentSnapshot**.
6. In the **initState** method, the **fetchData** method is called to load the data when the widget is initialized.
7. The **build** method constructs the UI for the manual search screen using a **Scaffold** widget. The scaffold's background color is set to transparent.
8. The UI consists of a container with a background image, an app bar, a search input field, and a list of vehicles.
9. The app bar has a transparent background and a title that reads "Manual Search".

10. The search input field is a **TextField** wrapped in a container. It has a prefix icon, hint text, and styling.

11. The list of vehicles is displayed using a **ListView.builder**. Each list item represents a vehicle and shows its details. When tapped, it navigates to the **DetailedInformationPage**.

12. The **_filteredDataList** is used to populate the list view, and it is checked for emptiness. If there are no results, a "No Results Found" message is displayed.

Detailed Information Page –

Here's a breakdown of the code:

1. The **DetailedInformationPage** class is a stateless widget that represents the page displaying detailed information about a vehicle. It takes a **DocumentSnapshot** as a parameter to retrieve the data.

2. In the **build** method, the data is extracted from the **documentSnapshot** and stored in variables such as **owner**, **ownerNo**, **ownerAdd**, **vehNo**, **vehMod**, **vehType**, and **vehCol**.

3. The UI is constructed using a **Scaffold** widget as the root widget. The background color is set to **Colors.blue[900]**.

4. The content of the page is wrapped in a **SafeArea** to ensure it's visible on devices with notches or system UI overlays.

5. The page contains a column with two main sections: a header section and an expanded section for displaying the detailed information.

6. The header section contains an app bar-like layout with an arrow back button and a text widget displaying "Detailed Information". Tapping the back button will navigate back to the previous screen.

7. The expanded section is wrapped in a container with a light background color (**Colors.amber[50]**) and rounded top corners.

8. Inside the expanded section, a **ListView** is used to display the detailed information. It has padding and contains several instances of the **_buildInfoItem** widget.
9. The **_buildInfoItem** widget is a helper method that creates a column with a label and a value. It takes a **label** and a **value** as parameters and returns the formatted widget.
10. Each **_buildInfoItem** widget displays a specific piece of information, such as owner name, contact number, address, vehicle number, vehicle model, vehicle type, and vehicle color. The label is styled as bold, and the value is displayed below it.

New Registration Request Page –

Here's a breakdown of the code:

1. The **NewRegistrationPageUser** class is a stateful widget that represents the page where users can submit a new registration request. It contains a form with various input fields for capturing user details and vehicle details.
2. The form fields are controlled using **TextEditingController** instances, which allow you to get and set the text entered by the user.
3. The form fields are wrapped in a **Form** widget, which allows you to validate the form data and submit it when the user taps the submit button.
4. The **register** method is called when the form is submitted. It validates the form data and creates a **RegistrationRequest** object with the entered values.
5. If the form is valid, the **RegistrationRequest** object is serialized to JSON using the **toJson** method defined in the **RegistrationRequest** class.
6. The serialized request is then stored in the Firestore collection **reg_requests** using the **add** method on a **CollectionReference**.

7. Additionally, a notification string is created with the request details and stored in the Firestore collection **notifications** along with the current timestamp.
8. If the registration request and notification creation are successful, a success message is displayed using a **SnackBar**, and the form fields are cleared.
9. If any error occurs during the registration process, an error message is displayed using a **SnackBar**.
10. The **initializeFirebase** method is called in the **initState** method to initialize Firebase.
11. The UI of the page consists of a **Scaffold** widget with a blue background color. The app bar is customized with a title, color, and text styles.
12. The body of the page is a container with a light amber background color and rounded top corners. It contains the form fields arranged in a column.
13. The form fields include text form fields for capturing the owner's name, phone number, address, and vehicle details such as number, model, type, and color.
14. Each text form field is wrapped in a **TextFormField** widget with appropriate validation logic.
15. The submit button is a centered elevated button that triggers the **register** method when pressed.

Rules and Regulations Page –

Here's a breakdown of the code:

1. The **RulesAndRegulations** class is a stateless widget that represents the page displaying rules and regulations. It contains a **Scaffold** widget as the top-level widget for the page.
2. The **Scaffold** has a blue background color and an app bar with a title of "Rules & Regulations".
3. The body of the page is a container with a light amber background color and rounded top corners. It contains a **Padding** widget to add padding around the content.
4. Inside the **Padding**, there's a **SingleChildScrollView** to enable scrolling if the content exceeds the screen height.
5. The main content of the page is a column widget that contains the rules and regulations information.
6. The first element in the column is a **Text** widget with the text "Offenses and Penalties". It has a font size of 20 and a bold font weight.
7. Following the text, there are multiple **InteractiveViewer** widgets with **Image** widgets inside them. Each **Image** widget displays an image related to offenses and penalties. You need to replace the image asset paths (**'assets/offence2022_1.png'**, etc.) with the actual paths to your image assets.
8. Each **Image** widget has a specified height of 400 pixels and a width set to **double.infinity** to fill the available space horizontally.
9. Between each **Image** widget, there's a **SizedBox** to add some vertical spacing.

About Page –

Here's a breakdown of the code:

1. The **About** class is a stateless widget that represents the page displaying information about the app. It contains a **Scaffold** widget as the top-level widget for the page.
2. The **Scaffold** has a blue background color and an app bar with a title of "About".
3. The body of the page is a container with a light amber background color and rounded top corners. It contains a **Padding** widget to add padding around the content.
4. Inside the **Padding**, there's a **SingleChildScrollView** to enable scrolling if the content exceeds the screen height.
5. The main content of the page is a column widget that contains the information about the app.
6. The first element in the column is an **Image.asset** widget that displays an app logo. You need to replace the image asset path (**'assets/hatim_logo.png'**) with the actual path to your app logo image.
7. Following the logo, there's a **Text** widget that provides a description of the app.
8. Next, there's a section with the title "Key Features" and a list of key features described using bullet points.
9. After that, there's a section with the title "Permissions" and a description of the permissions required by the app.
10. Following that, there's a section with the title "Disclaimer" and a disclaimer about the information provided by the app.

11. Below the disclaimer, there's a section with the title "Developers" that displays information about the app developers. It includes their names, roll numbers, registration numbers, semester, and institution.

12. After the developers' section, there's a section with the title "Contact Us" that displays contact information. It includes icons for phone, email, and address, along with the corresponding contact details.

13. Finally, there's a **BottomAppBar** widget at the bottom of the page that displays the copyright information. The copyright text is dynamically generated using the current year obtained from **DateTime.now()** and the **DateFormat** class from the **intl** package.

Admin Login Page –

Here's a breakdown of the code:

1. The **LoginPage** class is a stateful widget that represents the login page.
2. The **_emailController** and **_passwordController** are **TextEditingController** instances used to retrieve the user's email and password input.
3. The **_isLoading** boolean variable is used to track the loading state of the login process.
4. The **_passwordVisible** boolean variable is used to toggle the visibility of the password field.
5. The **_login** method is an asynchronous function that handles the login process. It uses **FirebaseAuth** to sign in with the provided email and password. If the login is successful, it redirects to the **AdminPage**. If there's an error, it shows an error dialog.
6. The **build** method builds the UI of the login page. It consists of a **Scaffold** widget with a **Column** as its child. The **Column** contains the login form,

including email and password fields, a visibility toggle button for the password field, and a login button.

7. The email and password fields are implemented using **TextFormField** widgets with appropriate styling and controllers.
8. The password field is wrapped in a **Stack** widget along with an **IconButton** to toggle the password visibility. The visibility state is managed by the **_passwordVisible** variable.
9. The login button is implemented using an **ElevatedButton** wrapped in a **Container**. The button is disabled when the login process is ongoing (controlled by **_isLoading**), and it shows a loading indicator when in the loading state.
10. The login page has a gradient background and uses **SafeArea** to avoid overlapping with system UI elements.

Admin Panel –

Here's a breakdown of the code:

1. The **AdminPage** class is a stateless widget that represents the admin panel page.
2. The **role** parameter is used to determine the role of the admin.
3. The **AppBar** is displayed at the top of the page, showing the title "Admin Panel" and a logout button that navigates back to the home screen when pressed.
4. The body of the page consists of a **Stack** widget with an image background and a column of widgets.
5. Inside the column, there is a container that displays the document count for the "veh_info" collection and the registration requests count.

This is done using **FutureBuilder** widgets to asynchronously fetch the counts from Firestore.

6. Below the container, there are three **ElevatedButton** widgets wrapped in **SizedBox** widgets. Each button represents a different functionality of the admin panel.
    - The first button, when pressed, navigates to the **RegisteredDocumentPage**.
    - The second button navigates to the **NewRegistrationPageAdmin**.
    - The third button navigates to the **RegistrationApprovalPage**.
7. The buttons have custom styling and are wrapped in **SizedBox** widgets to specify their width and height.
8. The **getDocumentCount** method is an asynchronous function that retrieves the document count for a given collection from Firestore. It returns the number of documents as an **int** value.

[Admin] Registered Documents Page –

Here's a breakdown of the code:

1. The **RegisteredDocumentPage** class is a stateful widget that represents the page for displaying registered documents.
2. It has several **TextEditingController** instances to control the text fields for editing document details.
3. The **dispose** method is overridden to dispose of the text controllers when the widget is disposed.
4. The **build** method constructs the UI for the page.
5. The **AppBar** at the top of the page displays the title "Registered Documents."

6. The **body** consists of a **StreamBuilder** widget that listens to a stream of snapshots from the Firestore collection "veh_info."
7. Inside the **StreamBuilder**, the snapshot's connection state is checked to display different UI based on the connection status.
8. If there's an error, it shows an error message.
9. If the connection is waiting, it displays a loading indicator.
10. If the connection is active and there are documents available, it builds a **ListView.builder** to display the registered documents.
11. Inside the **ListView.builder**, it retrieves the data of each document and assigns it to variables.
12. It sets the text of the text controllers to the corresponding document data, allowing for editing the document details.
13. Each document is displayed as a **Card** in the list, showing the vehicle number and owner details.
14. Each **Card** has an "Edit" button and a "Delete" button displayed as icons.
15. When the "Edit" button is pressed, it shows an **AlertDialog** with text fields pre-filled with the document's data for editing.
16. When the "Save" button is pressed in the **AlertDialog**, it updates the Firestore document with the edited data.
17. When the "Cancel" button is pressed in the **AlertDialog**, it closes the dialog without saving any changes.
18. When the "Delete" button is pressed, it deletes the corresponding document from Firestore.

[Admin] New Document Registration Page –

Here's a breakdown of the code and its description:

1. Import Statements:
    - The code begins with importing necessary dependencies such as **cloud_firestore**, **flutter/material.dart**, **flutter/services.dart**, and **intl** packages.
2. NewRegistrationPageAdmin Class:
    - This class extends **StatefulWidget** and represents the main page widget for the new registration form.
3. Constructor:
    - The constructor for **NewRegistrationPageAdmin** class is defined, which calls the superclass constructor and initializes the key parameter.
4. NewRegistrationPageAdminState Class:
    - This class extends **State** and represents the state of the **NewRegistrationPageAdmin** widget.
5. Text Editing Controllers:
    - Several **TextEditingController** objects are created to control the input fields in the form. These controllers are used to get the values entered by the user.
6. initState() Method:
    - The **initState** method is overridden to attach a listener to the vehicle number controller (**vehNoController**). The listener calls **_capitalizeInput** method whenever the text in the controller changes.

7. dispose() Method:
    - The **dispose** method is overridden to clean up the vehicle number controller (**vehNoController**) when the widget is removed from the tree.
8. _capitalizeInput() Method:
    - This method is called when the vehicle number changes.
    - It capitalizes the text and removes any blank spaces from the vehicle number.
    - The updated text is set back to the controller.
9. _register() Method:
    - This method is called when the "REGISTER" button is pressed.
    - It retrieves the values from the text controllers and creates a new document in the "veh_info" collection of Firestore.
    - It also adds a new document in the "notifications" collection, notifying about the new registration.
    - If the registration is successful, it displays a success message using **ScaffoldMessenger**.
    - If there is an error during registration, it displays an error message using **ScaffoldMessenger**.
10. build() Method:
    - This method builds the UI of the widget using the Flutter widget tree.
    - It creates a Scaffold widget with an AppBar and a body.
    - The body consists of a SingleChildScrollView widget with a column of various form fields and a "REGISTER" button.
    - The AppBar displays a title of "New Registration".

11. TextField Widgets:

    - Several TextField widgets are used to display and capture user input for owner details and vehicle details.
    - Each TextField widget is associated with a respective TextEditingController for retrieving the entered values.

12. ElevatedButton Widget:

    - The "REGISTER" button is an ElevatedButton widget that triggers the **_register** method when pressed.
    - It has a custom style defined using **ElevatedButton.styleFrom** to customize its appearance.

[Admin] Registration Requests Page –

Here's the breakdown of the code:

1. **RegistrationApprovalPage**:

    - This class extends **StatefulWidget** and represents the main page widget for registration approval.
    - It builds a list of registration requests, allowing the user to approve or reject each request.

    Constructor:

    - The constructor initializes the **key** parameter.

    storeRejectionNotification Method:

    - This method is responsible for storing a rejection notification in the Firestore database.
    - It receives the owner's name and vehicle number as parameters.

- The method generates the current timestamp and formats the date using the **intl** package.
- It adds a new document to the "notifications" collection in Firestore, containing information about the rejection.

build Method:

- This method builds the UI of the widget using the Flutter widget tree.
- It creates a Scaffold widget with an AppBar and a body.
- The body uses a StreamBuilder to listen for changes in the "reg_requests" collection in Firestore.
- If there is an error while retrieving the data, it displays an error message.
- If the data is still loading, it displays a progress indicator.
- If there are no new registration requests, it displays a message indicating so.
- If there are requests, it builds a ListView.builder to display each request as a card.
- Each card includes information about the owner and the vehicle.
- The card also contains buttons to approve or reject the request.
- When the approve button is pressed, it moves the request document to the "veh_info" collection, deletes the request document from "reg_requests," stores an approval notification, and displays a success message.
- When the reject button is pressed, it stores a rejection notification, deletes the request document from "reg_requests," and displays a success message.

2. **RegistrationDetailsPage**:

    - This class extends **StatelessWidget** and represents the page widget that displays detailed information about a registration request.

    Constructor:

    - The constructor initializes the **request** parameter, which contains the details of the registration request.

    build Method:

    - This method builds the UI of the widget using the Flutter widget tree.
    - It creates a Scaffold widget with an AppBar and a body.
    - The body displays owner information and vehicle information in a column format.
    - Each piece of information is displayed using the **_buildInfoRow** method, which creates a row with a label and its corresponding value.

Registration Factory –

Here's a breakdown of the code:

Properties:

- **owner**: Represents the owner's name.
- **ownerNo**: Represents the owner's contact number.
- **ownerAdd**: Represents the owner's address.
- **vehNo**: Represents the vehicle number.
- **vehMod**: Represents the vehicle model.
- **vehType**: Represents the vehicle type.

- **vehCol**: Represents the vehicle color.

Constructor:

- The constructor initializes the properties of the **RegistrationRequest** class with the required parameters.

fromJson Factory Method:

- This factory method is responsible for creating an instance of **RegistrationRequest** from a JSON object.
- It takes a **Map<String, dynamic>** as a parameter, which represents the JSON data.
- It extracts the values from the JSON map and assigns them to the corresponding properties of the **RegistrationRequest** instance.
- The factory method then returns the created instance.

toJson Method:

- This method converts the **RegistrationRequest** instance to a JSON object.
- It creates a **Map<String, dynamic>** and assigns the property values to the corresponding keys in the map.
- Finally, it returns the JSON map representation of the **RegistrationRequest** instance.

Overall, the **RegistrationRequest** class serves as a model for representing registration requests. It provides methods for converting instances to and from JSON format, allowing for easy serialization and deserialization of data.

5. TESTING AND IMPLEMENTATION

CONTENT

- **Testing Techniques and Strategies**
- **Testing Reports**

TESTING AND IMPLEMENTATION

Testing Techniques and Strategies

When it comes to testing techinques and strategies, we performed the following techniques:

1. Unit Testing.
   Tests are performed separately for each pages.

2. Widget Testing.
   This test mainly considers UI testing.

3. Integration Testing
   The relationships and functionalities between different pages are tested after performing Unit Testing.

4. Device and Platform Testing
   Several prototypes are being created ocassionally so that Device and Platform testing could be performed.

5. User Acceptance Testing
   Our protoypes are being tested by our team, as well as by external users so that we can collect feedbacks and gather more requirements for our application.

Testing Reports

This project has undergone many series of tests, and errors present have been dealt with. But there may be unforeseen errors which are however unlikely. With the amount of time we have, we perfected it and run trial many a time, so that we can concurred it is usable for all.

6. DRAWBACKS AND LIMITATIONS

This project is undertaken with careful analysis to meet the given criteria of this final project. Corrective measures have been taken. But our insufficiency of knowledge, skill and experience, this project definitely has its own limitations and drawbacks. It definitely cannot have a wide scope as found in other big projects due to the limitation of time. It is primarily a compilation of only what information we thought would be relevant to satisfy the basic requirements of the project.

Here are some potential drawbacks and limitations of the app:

1. Accuracy of Text Recognition: The accuracy of text recognition depends on various factors such as image quality, lighting conditions, and the complexity of the text. The app may not always accurately extract the vehicle information from the scanned image, leading to incorrect results.
2. Dependence on Camera Quality: The quality of the device's camera can significantly impact the app's performance. Low-quality cameras may produce blurry or distorted images, making it difficult for the text recognition algorithm to extract accurate information.
3. Limited to Indian Vehicles: The app seems to be designed specifically for Indian vehicles, as it searches for Indian state vehicle codes. It may not work as intended for vehicles from other countries, as the text recognition and database queries are tailored to Indian vehicle information.
4. Dependency on Network Connectivity: The app requires an internet connection to retrieve vehicle information from the Firestore database. If the user's device has poor or no network connectivity, it may affect the app's functionality.
5. Limited Database: The app relies on a specific Firestore database collection named 'veh_info' to retrieve vehicle information. Additionally, it is limited for use within HATIM as it serves as an app for HATIM.

7. CONCLUSION

This project is basically designed to be used for all the student or teacher of every department remotely from anyplace through online. This chapter seeks to conclude the work that has been done, and to present the areas that mark to the completion of this project work.

This project work is the compilation of the ideas, views and thoughts of our group, Group 2 of 6th Semester BCA, 2023.

We are deeply conscious of the fact that this project would neither have been undertaken nor pursued and completed but for the tremendous support that we received from Mr. H. Lalruatkima, our Project Guide and K. Lalmuanpuia Head of Department, who not only give us their full support but provide us with all kinds of necessities that we required in the project we are working.

We would also like to place and record our sincere thanks and gratitude to our Principal, Mr. Vuansanga Vanchhawng, for extending his full support and consent to this project undertaken.

8. BIBLIOGRAPHY

1. K. K. Agarwal and Y. Singh: Software Engineering, New Age International (2005)

2. Coding with T. (n.d.). *Playlists*. Retrieved from https://www.youtube.com/@CodingwithT/playlists

3. Mitch Koko. (n.d.). *Playlists*. Retrieved from https://www.youtube.com/@createdbykoko/playlists