

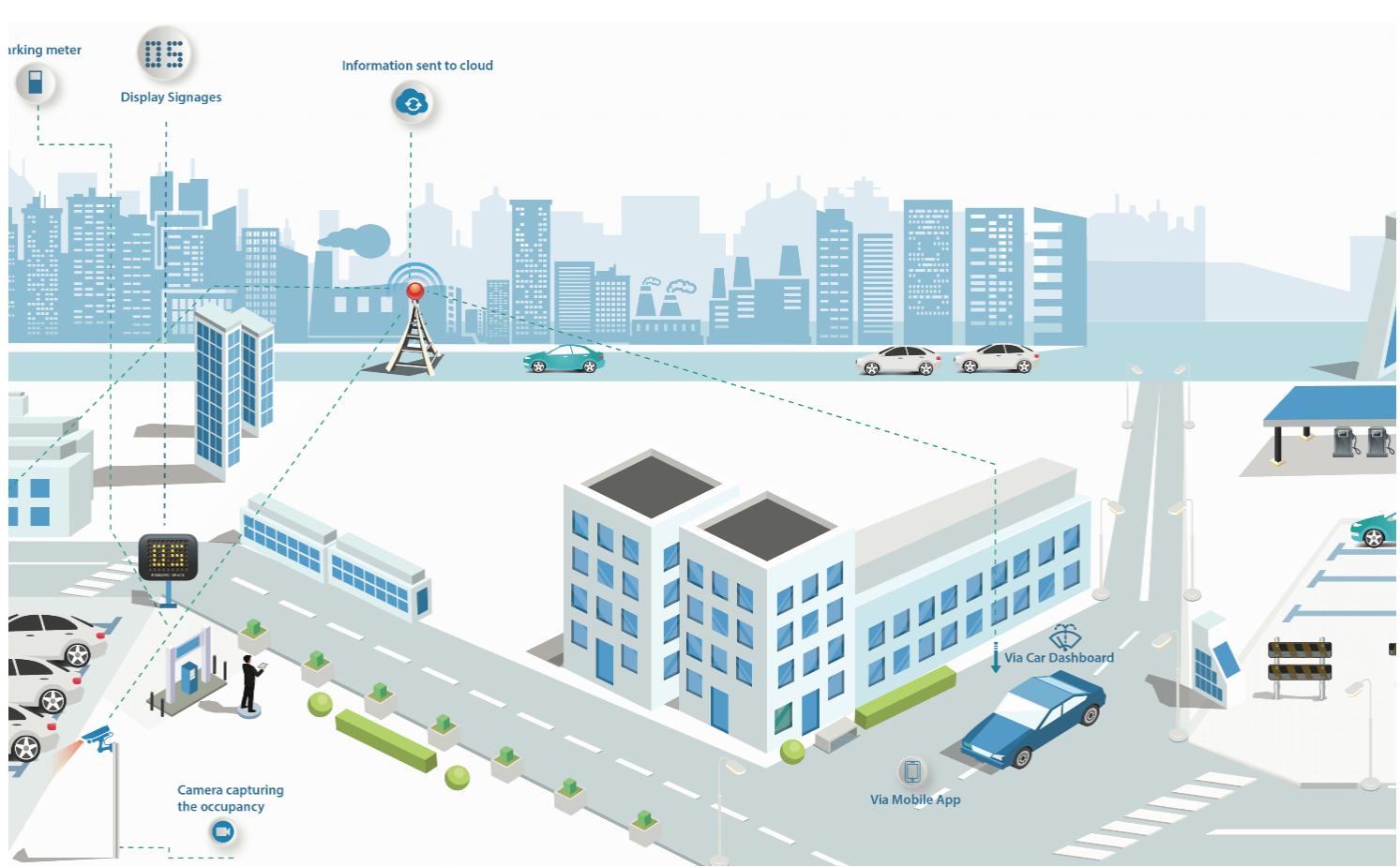
# SMART PARKING(IOT)

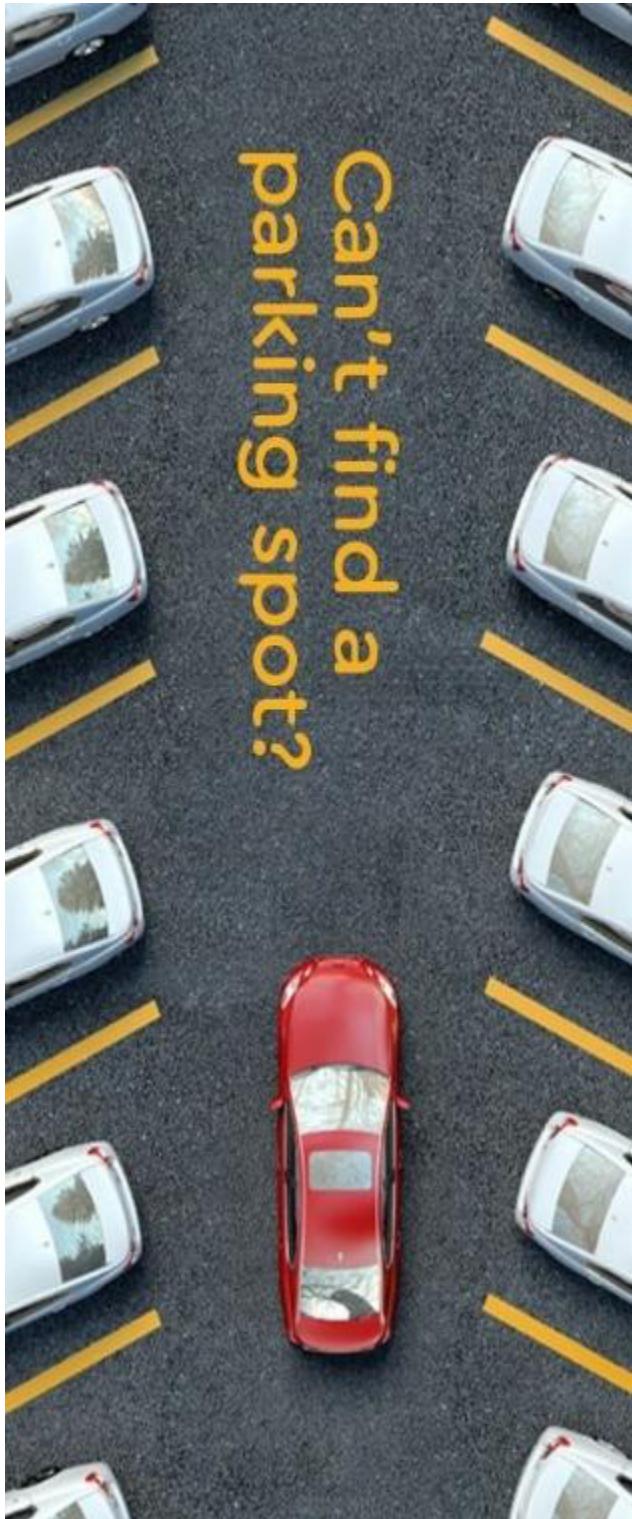
NAME:BINO LAL L.R

ID: AU962921104007

EMAIL: [binolal03@gmail.com](mailto:binolal03@gmail.com)

PHASE:4





## TABLE OF CONTENTS

Introduction.....	3
Development Part 2.....	4
Development Of Mobile App.....	4
FrameWork By Flutter.....	5
Data Received From Raspberry PI:.....	7
Working Of App.....	12
Conclusion.....	18

# INTRODUCTION:



## SMART PARKING APP DEVELOPMENT

Creating a mobile app in Flutter to display real-time parking availability data received from a Raspberry Pi involves several steps. Below is an extended guideline on how to design the app's functions to receive and display this data.

# **DEVELOPMENT PART 2**

## **DEVELOPING THE MOBILE APP USING PYTHON:**

- 1. Set Up Your Development Environment:**
  - ❖ Install Python on your development machine.
  - ❖ Install the required development tools for your chosen framework (e.g., Kivy or BeeWare).
- 2. Choose a Framework:**
  - ❖ Select a Python mobile app framework that suits your project's needs. Kivy and BeeWare are popular choices.
- 3. Design Your App:**
  - ❖ Plan the user interface and overall design of your mobile app.
  - ❖ Create wireframes or mockups to visualize the app's layout.
- 4. Coding Your App:**
  - ❖ Write Python code to implement the app's functionality and user interface.
  - ❖ Utilize the framework's libraries and features for creating mobile apps.
- 5. Testing and Debugging:**
  - ❖ Test your app on both Android and iOS emulators.
  - ❖ Debug and fix any issues or errors that arise during testing.
- 6. User Interface (UI):**
  - ❖ Implement the UI components using the framework's widgets and elements.
  - ❖ Ensure a responsive design that adapts to different screen sizes and orientations.
- 7. App Logic:**
  - ❖ Write Python code to handle user interactions, data processing, and other app-specific functions.
- 8. Data Storage:**
  - ❖ Implement data storage solutions, such as using SQLite databases, for storing app-related data.
- 9. Connectivity:**
  - ❖ Incorporate features for internet connectivity, if your app requires it, such as API calls or cloud services integration.
- 10. Testing on Physical Devices:**

- ❖ Test your app on physical Android and iOS devices to ensure compatibility and performance.
11. Optimization and Performance:
- ❖ Optimize your code and resources for better performance and battery efficiency.
12. Deployment:
- ❖ Package your app for the Google Play Store (Android) and App Store (iOS).
  - ❖ Follow the platform-specific guidelines for app submission and publication.
13. Maintenance and Updates:
- ❖ Continue to maintain and update your app as needed to fix bugs, add new features, and ensure compatibility with newer OS versions.

## FRAMEWORK BY FLUTTER:

- Here are the steps to create a real-time parking availability app using Flutter:
  - ❖ Set Up Your Development Environment:

Install Flutter and Dart on your development machine. Follow the official Flutter installation guide for your specific operating system.

❖ Create a New Flutter Project:  
Use the flutter create command to set up a new Flutter project.  
`flutter create parking_app`

❖ Design the User Interface (UI):  
Design the app's user interface, including screens, widgets, and layouts. Consider using Flutter's extensive library of pre-built widgets.

❖ Implement Real-Time Data Retrieval:  
Integrate with a real-time data source or API that provides parking availability information. This source could be sensors in parking lots, a web API, or any other relevant data provider.

- ❖ Display Real-Time Parking Availability:

Create components to display parking availability information. You can use widgets like Text, ListView, and GridView to show the status of parking spaces. Update this information in real-time.

dependencies:

```
flutter:  
  sdk: flutter  
  http: ^0.13.3  
  provider: ^6.0.2
```

Run flutter pub get to fetch the added dependencies.

❖ User Interactivity:

Implement features for users to interact with the app, such as filtering by location, reserving parking spaces, or getting directions to available spots.

❖ User Authentication (Optional):

Implement user authentication to allow features like reserving parking spaces or saving preferences.

❖ Testing:

Test your app on both Android and iOS simulators/emulators as well as physical devices.

❖ Optimization:

Optimize your app for performance, ensuring smooth real-time updates.

❖ Deployment:

Prepare and package your app for the Google Play Store (Android) and the Apple App Store (iOS).

❖ Publish and Maintain:

Publish your app and periodically update it to address any issues, improve user experience, and adapt to new requirements or operating system versions.



# **DATA RECEIVED FROM RASPBERRY PI:**

Designing an app to receive and display parking availability data received from a Raspberry Pi involves several key functions and features.

## **1. Data Reception:**

- The app should be able to receive real-time parking availability data from the Raspberry Pi. This data could be transmitted over the network using a communication protocol such as MQTT or HTTP.

## **2. User Authentication (Optional):**

- Implement user authentication if the app requires user-specific functionality, like reserving parking spaces or saving preferences.

## **3. Display Available Parking Spaces:**

- Create a user interface to display parking spaces. You can use a list or grid view to show the availability of each space.
- Use visual cues like colors (e.g., green for available spaces, red for occupied) or icons to represent space status.

## **4. Real-Time Updates:**

- Set up a mechanism for real-time updates of parking availability data. This can be achieved by periodically polling the Raspberry Pi or by using a push-based mechanism where the Raspberry Pi sends updates to the app whenever there's a change.

## **5. Filter and Sort Options:**

- Provide options for users to filter and sort parking spaces based on various criteria, such as location, distance, availability, and pricing.

## **6. Parking Space Details:**

- Allow users to tap on a parking space to view more details, including its location on a map, pricing, and any additional information.



## **7. Notifications and Alerts:**

- Implement notifications or alerts to inform users when a parking space they've reserved is about to expire or if there are sudden changes in availability.

## **8. Reserve Parking Spaces (Optional):**

- If your app allows users to reserve parking spaces, provide a reservation feature with payment processing, and send booking confirmations.

## **9. Maps Integration:**

- Integrate maps to show the locations of parking spaces and provide directions to the selected space.

## **10. User Profile and Preferences:**

- Enable users to create profiles and set preferences, such as favorite locations, vehicle information, or notification settings.

## **11. History and Reporting:**

- Maintain a history of users' parking activities, including past reservations, payments, and usage statistics.

## **12. User Feedback:**

- Include a feature for users to provide feedback or report issues with the app or parking facilities.

## **13. Data Privacy and Security:**

- Ensure that user data is handled securely and that sensitive information is encrypted, especially when dealing with user profiles and payment data.

## **14. Testing and Quality Assurance:**

- Thoroughly test the app on different devices, both Android and iOS, to ensure a consistent user experience.



## 15. User Education:

- Provide users with instructions and information on how to use the app effectively.

Here's a simplified Python script for the Raspberry Pi to send data to the mobile app:

```
import paho.mqtt.client as mqtt
import time

# MQTT settings
MQTT_BROKER = "your_mqtt_broker_address"
MQTT_PORT = 1883
MQTT_TOPIC = "your_mqtt_topic"

# Initialize MQTT client
client = mqtt.Client()

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker")
    else:
        print(f"Connection failed with code {rc}")

client.on_connect = on_connect
client.connect(MQTT_BROKER, MQTT_PORT, 60)

try:
    while True:
        # Your data collection logic here
        data = "Your parking availability data goes here"

        # Send data to the mobile app
        client.publish(MQTT_TOPIC, data)

        print(f"Data sent: {data}")
        time.sleep(10) # Adjust the interval as needed

except KeyboardInterrupt:
    print("Script terminated by user")
    client.disconnect()
```

# FRAMEWORK IN FLUTTER:

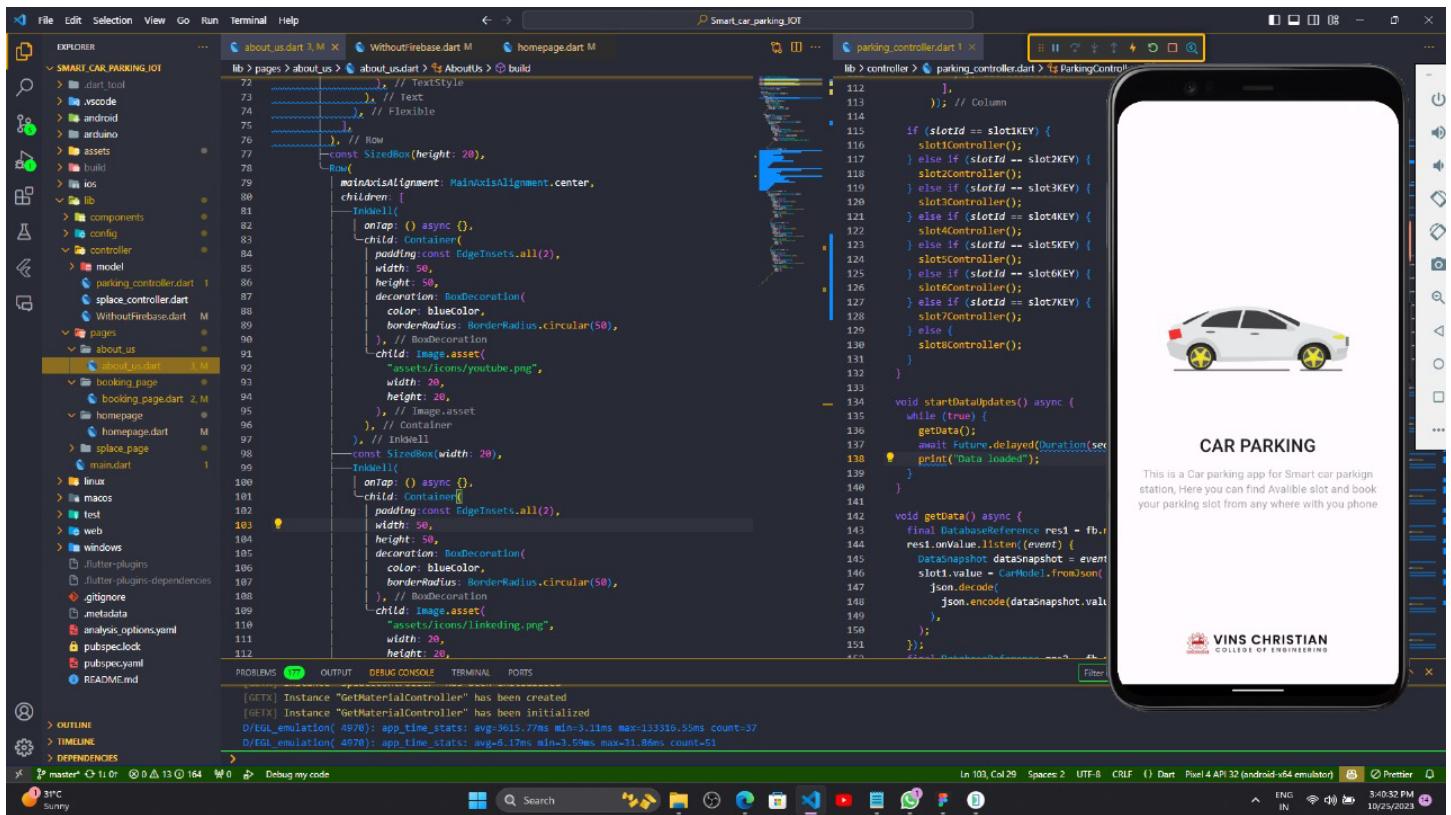


FIG:1

Design the app's user interface, including screens, widgets, and layouts. Consider using Flutter's extensive library of pre-built widgets.

The screenshot shows the development environment of an IDE (Android Studio) with two code files open:

- `WithoutFirebase.dart`: Contains logic for booking slots based on slot ID and key.
- `parking_controller.dart`: Contains methods for slot controllers and data loading.

The mobile application interface displays a grid of 8 parking slots (A-1 to A-8). Each slot contains a blue rectangular button labeled "BOOK". The application title is "SMART CAR PARKING".

```

    WithoutFirebase.dart
    package:Smart_Car_parking_IOT
    import 'package:flutter/material.dart';
    import 'package:firebase_database/firebase_database.dart';

    class WithoutFirebase {
        void slotController() async {
            double time = parkingHours.value;
            slot1.value = CarModel(
                booked: true,
                isParked: true,
                parkingHours: time.toString(),
            );
            print("slot one booked");
        }

        void slot2Controller() async {
            double time = double.parse(slot2.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot2KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot3Controller() async {
            double time = double.parse(slot3.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot3KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot4Controller() async {
            double time = double.parse(slot4.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot4KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot5Controller() async {
            double time = double.parse(slot5.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot5KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot6Controller() async {
            double time = double.parse(slot6.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot6KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot7Controller() async {
            double time = double.parse(slot7.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot7KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }

        void slot8Controller() async {
            double time = double.parse(slot8.value.parkingHours.toString());
            while (time != 0) {
                await Future.delayed(Duration(seconds: 1));
                time--;
                print(time);
                await fb.ref().child(slot8KEY).update(
                    {"secondsLeft": time, "isAvailable": false});
            }
        }
    }

    void slot1Controller() {
        slot1.value = CarModel(
            booked: true,
            isParked: true,
            parkingHours: time.toString(),
        );
        print("slot one booked");
    }

    void slot2Controller() {
        double time = double.parse(slot2.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot2KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot3Controller() {
        double time = double.parse(slot3.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot3KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot4Controller() {
        double time = double.parse(slot4.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot4KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot5Controller() {
        double time = double.parse(slot5.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot5KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot6Controller() {
        double time = double.parse(slot6.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot6KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot7Controller() {
        double time = double.parse(slot7.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot7KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void slot8Controller() {
        double time = double.parse(slot8.value.parkingHours.toString());
        while (time != 0) {
            await Future.delayed(Duration(seconds: 1));
            time--;
            print(time);
            await fb.ref().child(slot8KEY).update(
                {"secondsLeft": time, "isAvailable": false});
        }
    }

    void startDataUpdates() {
        while (true) {
            getdata();
            await Future.delayed(Duration(seconds: 1));
            print("Data loaded");
        }
    }

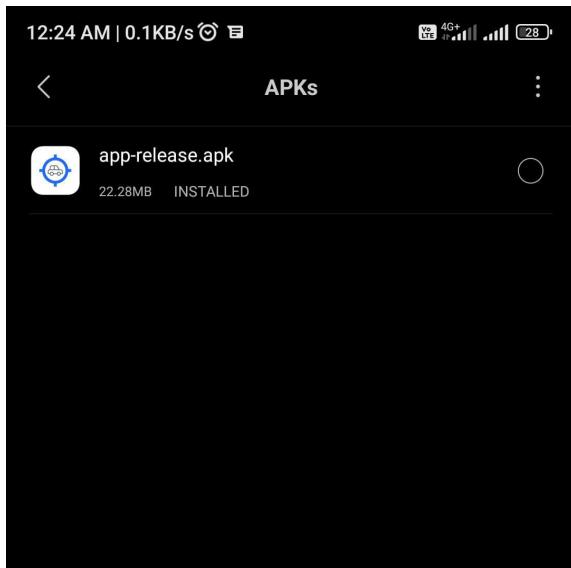
    void getData() {
        final DatabaseReference ref = fb.ref().child(slot);
        ref.onValue.listen(event) {
            DataSnapshot dataSnapshot = event.snapshot;
            slot1.value = CarModel.fromJson(
                json.decode(
                    json.encode(dataSnapshot.value),
                ),
            );
        }
    }
}

```

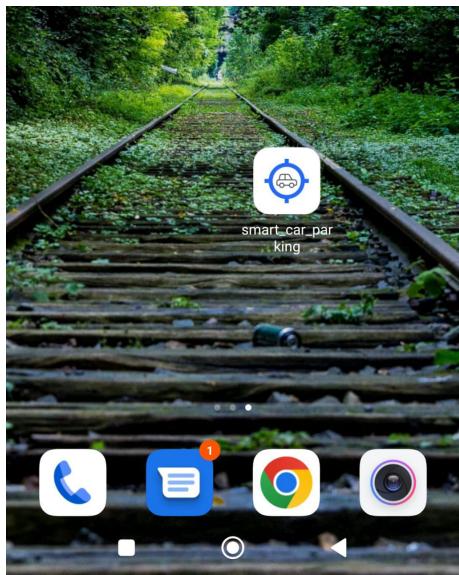
FIG :2

**The Design of the Mobile APP For SMART PARKING frame work has been successfully Completed..Kindly refer above flutter code and pictures.**

# **WORKING OF (SMART PARKING) APP:**



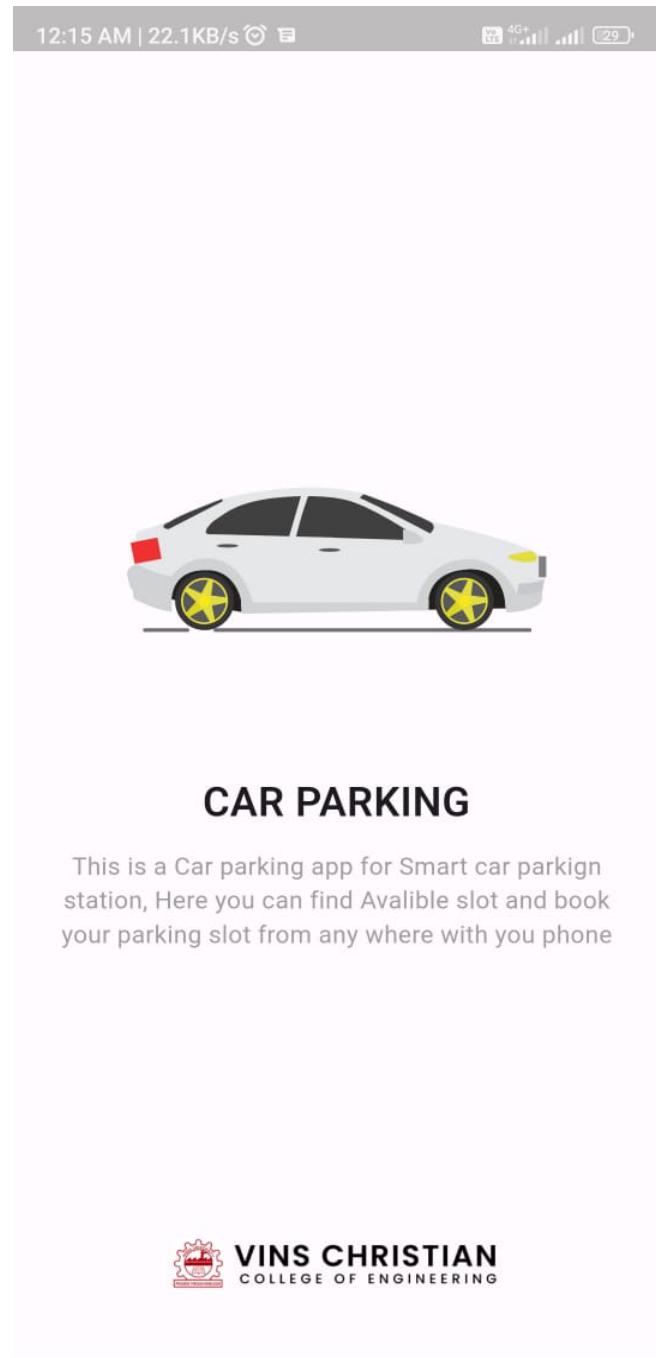
**FIG:3 APP LOGO(smart\_car\_parking)**



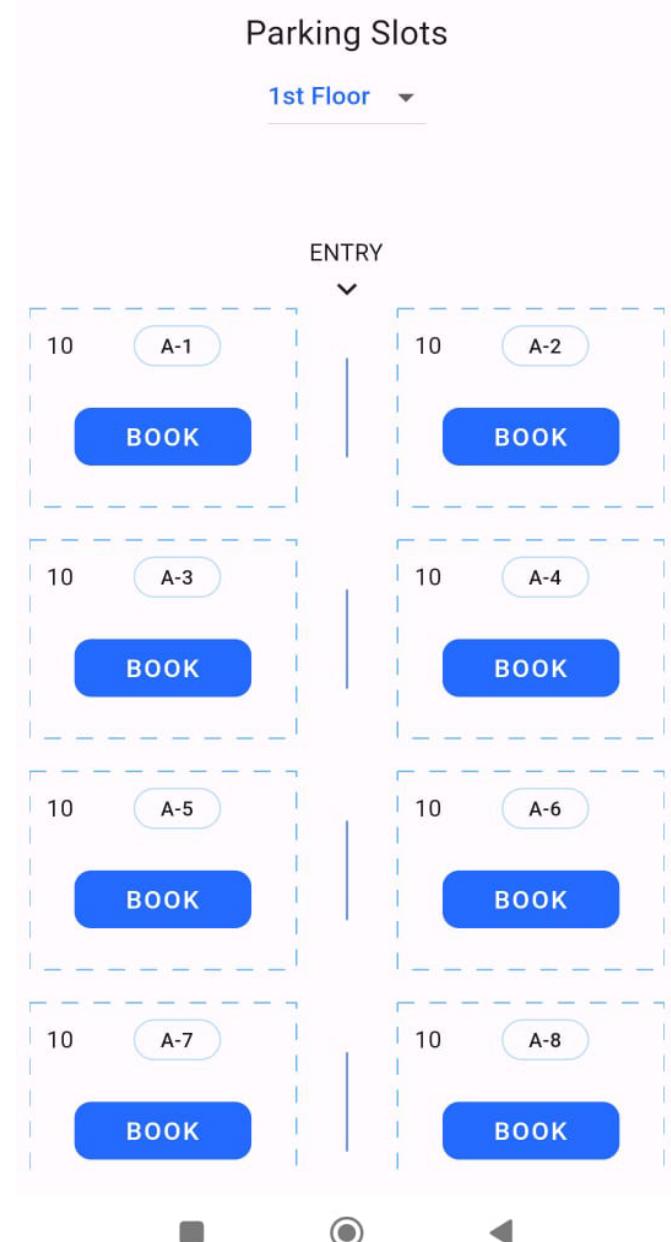
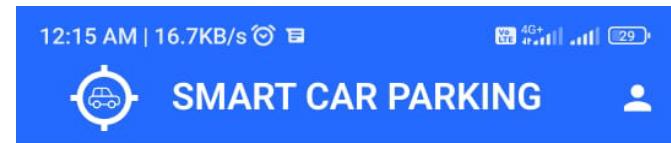
**FIG:4 APP INSTALLED IN MOBILE**



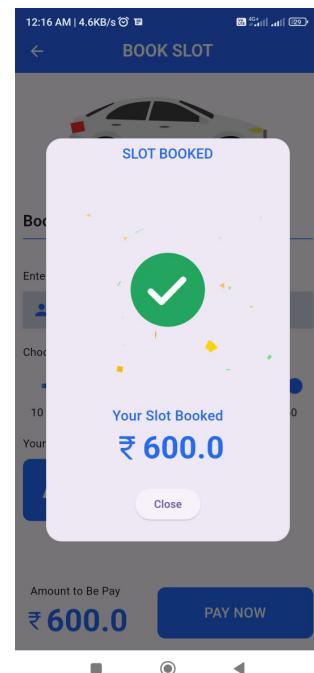
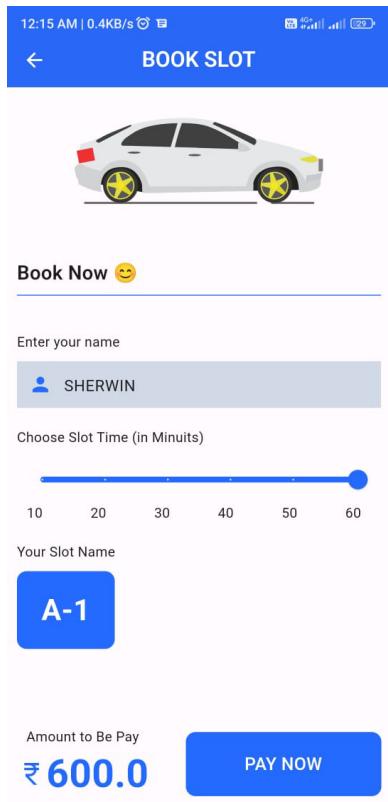
## PROCESS IN MOBILE APP WITH SCEENSHOT:



**FIG:5 FRONT PAGE OF APP**



**FIG:6 FREE SPACE CAN BE BOOKED**



**FIG:7,8 PARKING SPACE BOOKING**



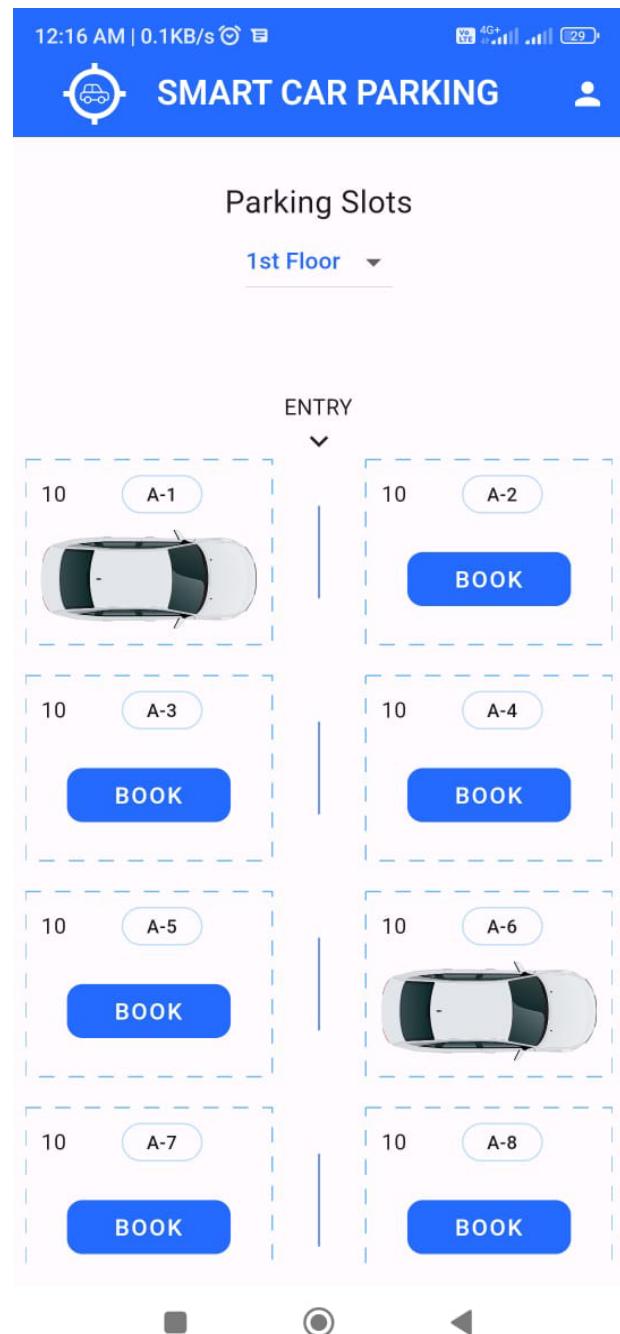


FIG:9 IT REPRESENT THE BOOKED SPACE

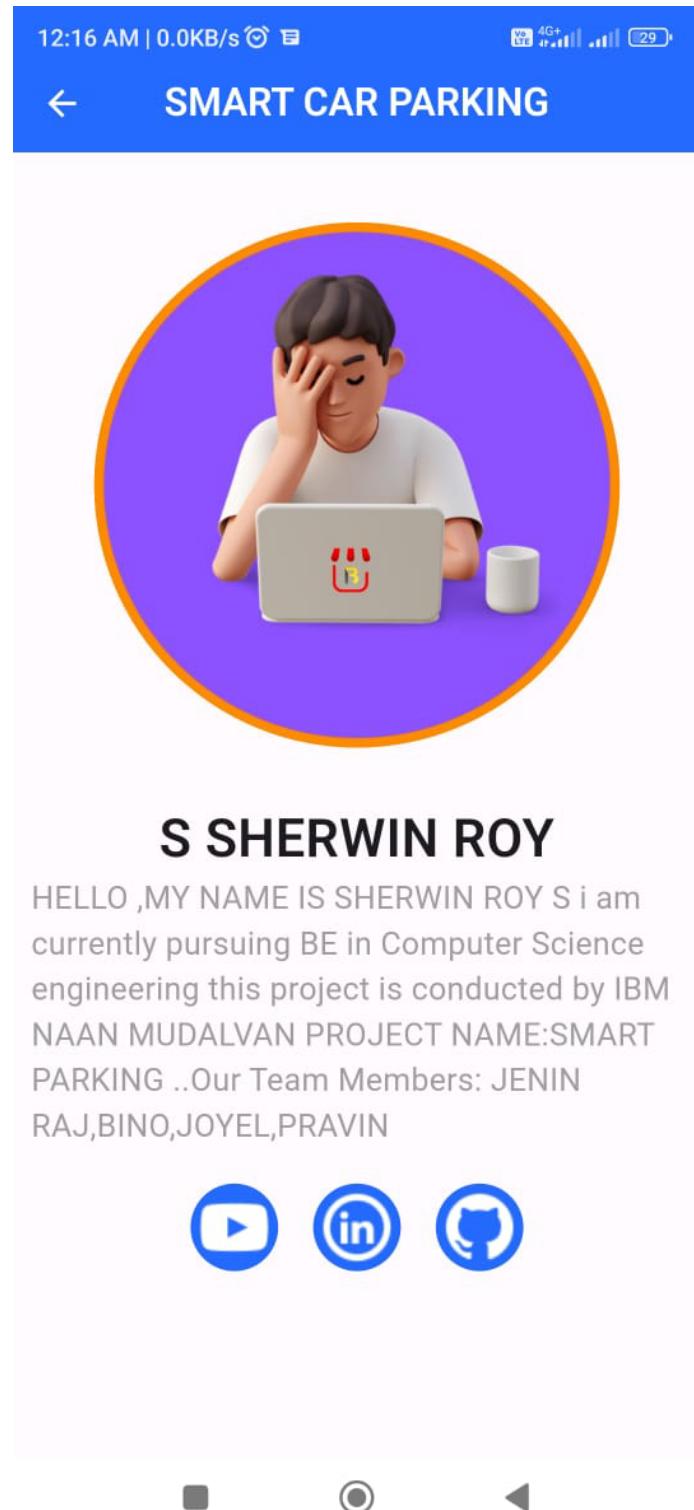


FIG:10 IT SHOWS THE MOBILE APP DASHBOARD

## **APP LINK:**

[https://github.com/Sherwinroy/Sherwinroy/tree/main/IOT\\_Phase4/APP](https://github.com/Sherwinroy/Sherwinroy/tree/main/IOT_Phase4/APP)

## **APP WORKING VIDEO:**

[https://youtube.com/shorts/\\_5Xx0uObm3g?si=W5h9OMCQ8nMlbqBw](https://youtube.com/shorts/_5Xx0uObm3g?si=W5h9OMCQ8nMlbqBw)

## **CONCLUSION:**

In conclusion, developing a real-time parking availability app using a mobile app development framework like Flutter is a significant and complex project that involves several critical steps. It requires creating a user-friendly interface, handling real-time data from a Raspberry Pi or other data source, and providing valuable features for users.

Key takeaways for this project include:

- Mobile App Framework
- User Interface Design
- Data Handling
- Real-Time Updates
- User Authentication (Optional)
- Map Integration (Optional)
- User Profiles and Preferences
- Notifications and Alerts
- Testing and Deployment
- Ongoing Maintenance

**THANKING YOU**

