LAWN MATE

DESIGN MANUAL
GROUP 18

TABLE OF CONTENTS

INTRODUCTION

This design manual is crafted to provide a thorough insight into an innovative system engineered to revolutionize grass-cutting processes through automation. With a focus on efficiency, safety, and user-friendly operation, the automated grass cutter system aims to redefine the conventional approach to lawn maintenance.

Designed to cater to both residential and commercial needs, this system offers a seamless and advanced solution to grass cutting, ensuring optimal results while minimizing user effort. The integration of cutting-edge hardware and intelligent software empowers users to achieve precision and reliability in lawn maintenance tasks.

Key Objectives:

Efficiency: The automated grass cutter system is engineered to significantly reduce the time and effort traditionally associated with manual grass cutting. It leverages automation to streamline the process, allowing users to achieve consistent and efficient results.

User-Friendly Operation: This design manual serves as a comprehensive guide for users at all levels, from installation to day-to-day operation. Clear instructions and intuitive controls make the automated grass cutter accessible to a broad audience, promoting ease of use.

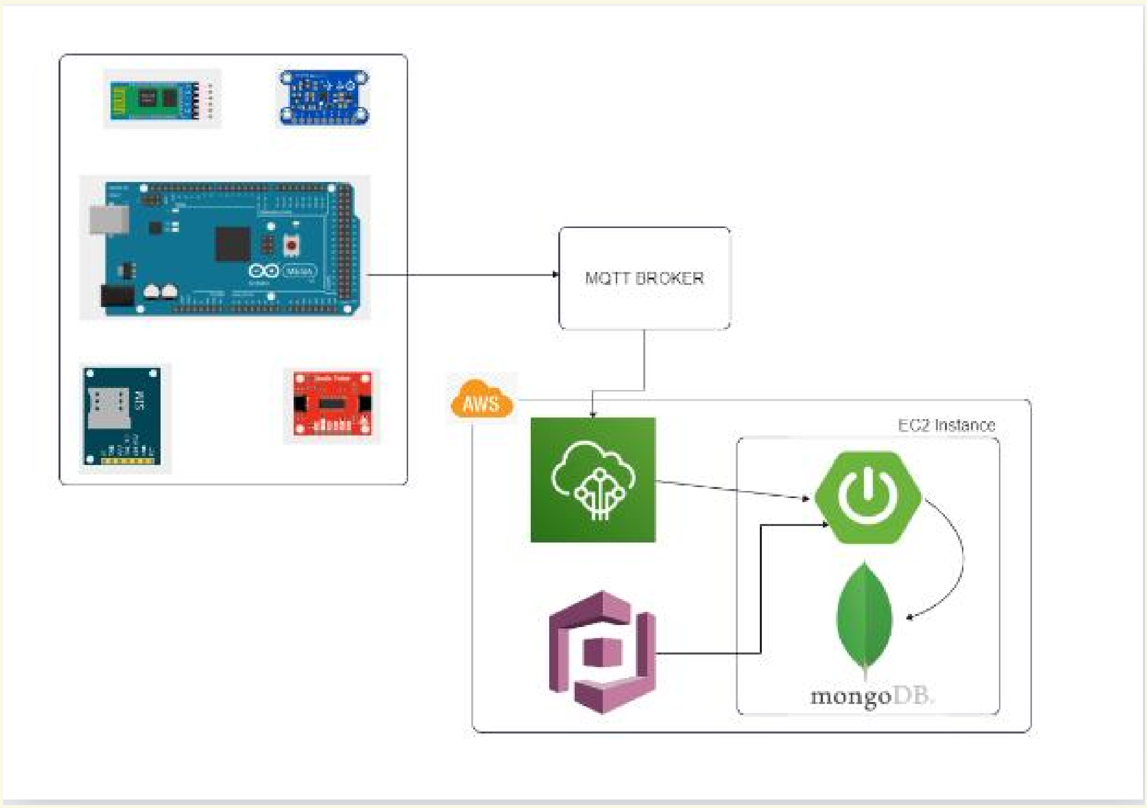
Environmental Considerations: The system is mindful of environmental impact. Electric-powered options are explored, minimizing noise pollution and reducing the carbon footprint associated with traditional gas-powered alternatives.

By delving into this design manual, users will gain valuable insights into the system's components, functionalities, and maintenance procedures. Whether one is a homeowner seeking an efficient lawn maintenance solution or a professional landscaper aiming to enhance service quality, this manual equips users with the knowledge needed to maximize the benefits of the automated grass cutter system.

Embark on a journey through the following sections to comprehend the intricacies of the automated grass cutter system, from its conceptualization to real-world application. This guide is tailored for individuals and professionals alike, fostering a deeper understanding of how automation can elevate the experience of grass cutting while contributing to a more sustainable and efficient future.

SYSTEM OVERVIEW

a) High level architecture



The automated grass cutter system employs sophisticated hardware and software to revolutionize grass cutting, ensuring an efficient and precise lawn maintenance experience. The hardware components consist of a cutting mechanism and a micro controller equipped with advanced capabilities. Sensors allow the micro controller to process real-time data and dynamically adjust the cutting mechanism to navigate. This integrated approach optimizes grass cutting for efficiency and precision.

Our grass cutter system utilizes cloud connectivity for enhanced functionality. The cloud stores crucial data, including user details, device details and location details. This facilitates remote access for users through a user-friendly interface, empowering them to customize schedules, and monitor grass cutting progress from any location.

Environmental considerations are paramount, with electric-powered options available to minimize noise pollution and reduce carbon emissions. The system aligns with sustainable lawn maintenance practices, contributing to a greener and healthier outdoor environment.

In conclusion, the high-level architecture of our automated grass cutter system seamlessly integrates cutting-edge hardware with cloud connectivity, delivering an advanced and user-friendly solution for efficient and environmentally conscious lawn maintenance.

b) Technology stack

React Native: Front-End
React is a JavaScript library that is used for building user interfaces. React is used for the development of single-page applications and mobile applications because of its ability to handle rapidly changing data. React allows users to code in JavaScript and create UI components.

Spring Boot: Back-End
Spring Boot is a robust Java-based framework tailored for the streamlined development and deployment of production-ready applications. With an opinionated approach, it simplifies the coding process by providing sensible defaults and reducing the need for extensive configuration. Noteworthy features include an embedded server for standalone execution, auto-configuration to minimize boilerplate code, and the convenience of starters for quick integration of common dependencies. Spring Boot excels in micro services development, offering built-in support for service discovery, distributed configuration, and seamless integration with Spring Cloud.

MongoDB: Cross-platform Document-Oriented Database
MongoDB is a NoSQL database where each record is a document consisting of key value pairs that are similar to JSON (JavaScript Object Notation) objects. MongoDB is flexible and allows its users to create

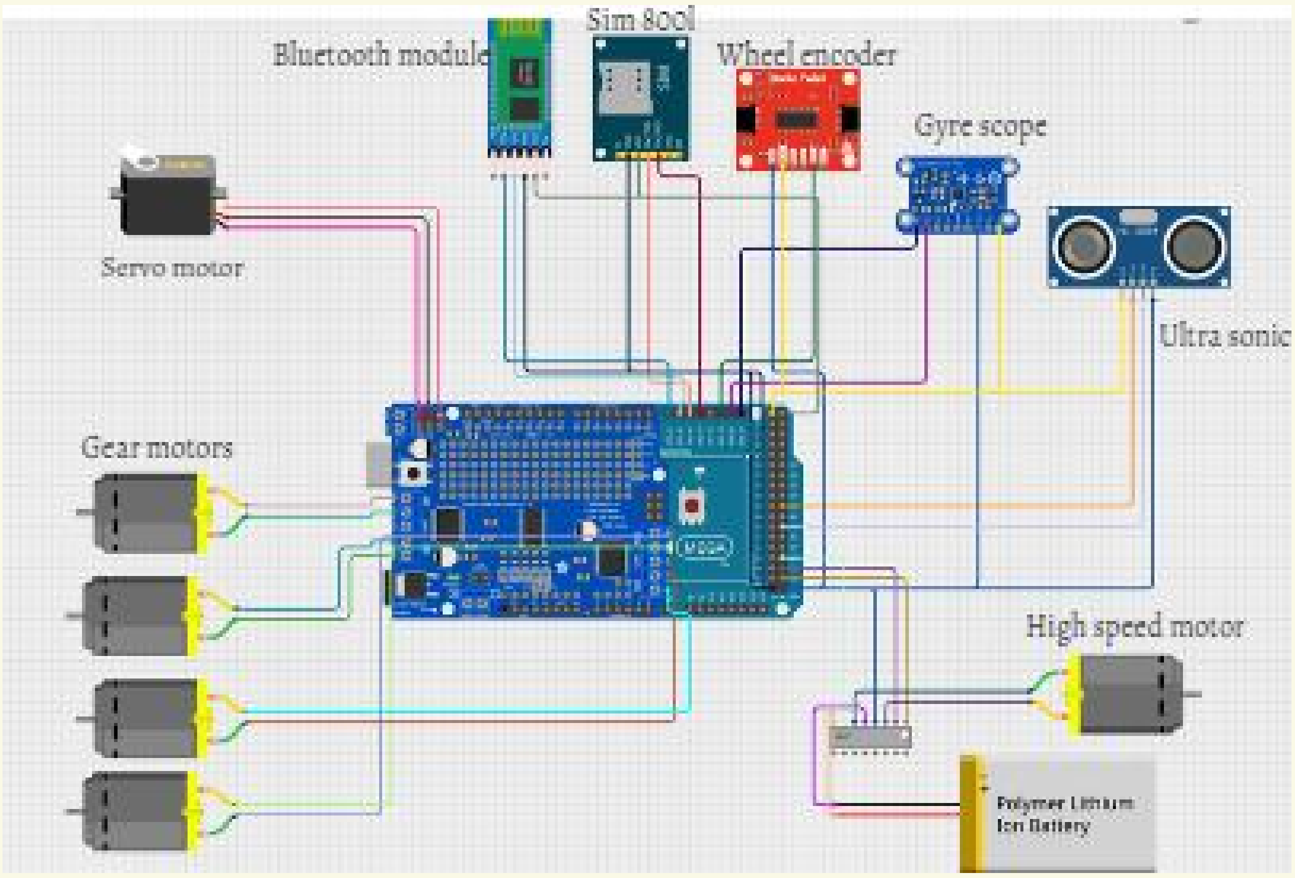
schema, databases, tables, etc. Documents that are identifiable by a primary key make up the basic unit of MongoDB. Once MongoDB is installed, users can make use of the Mongo shell as well. Mongo shell provides a JavaScript interface through which the users can interact and carry out operations (e.g.: querying, updating records, deleting records).

AWS: Cloud Hosting

Amazon Web Services (AWS) is a popular cloud hosting platform that provides a wide range of services for businesses, organizations. With AWS, users can easily and quickly set up and scale resources, including computing power, storage, and databases, without having to invest in and maintain their own physical infrastructure.
Additionally, AWS offers a variety of tools and services, such as Elastic Compute Cloud (EC2) and Simple Storage Service (S3), that allow users to deploy and manage their applications and data in the cloud. AWS also provides security, compliance, and data protection features, making it a secure and reliable option for hosting and managing critical business applications. Overall, AWS offers a flexible, cost-effective, and secure solution for hosting and managing data and applications in the cloud.

HARDWARE IMPLEMENTATION

a) Circuit Diagram



SERVER

In our project, we have employed Spring Boot as the robust foundation for our back-end infrastructure. Spring Boot, a Java-based framework, empowers our development process by providing an opinionated and convention-over-configuration approach. This choice not only accelerates the setup and development phases but also ensures a production-ready environment.

With Spring Boot, we benefit from an embedded server, eliminating the need for external server deployment. The framework's auto-configuration capabilities reduce boilerplate code, enabling us to focus on business logic rather than extensive setup. Additionally, Spring Boot Starters simplify dependency management, ensuring compatibility and facilitating the integration of common components seamlessly.

Our back-end, built on the Spring Boot framework, excels in handling the complexities of micro services architecture. It offers built-in support for crucial features such as service discovery, distributed configuration, and seamless integration with Spring Cloud.

The use of annotation-based configuration not only enhances the clarity of our code base but also aligns with modern development practices. This approach allows us to leverage the power of Java while ensuring a clean and concise implementation.

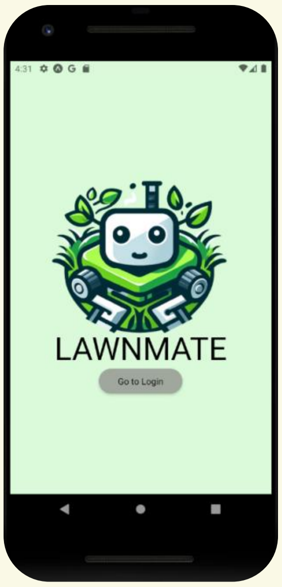
In summary, our project leverages the strengths of Spring Boot to deliver a robust, scalable, and efficient back-end infrastructure, providing a solid foundation for the development of our application.

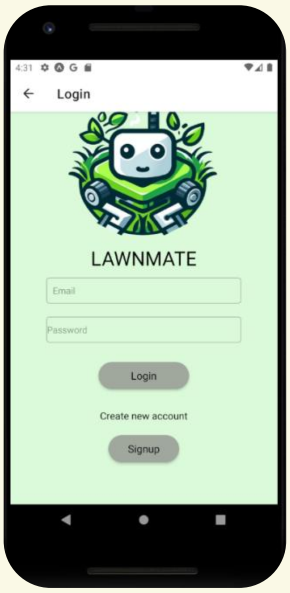
MOBILE APPLICATION

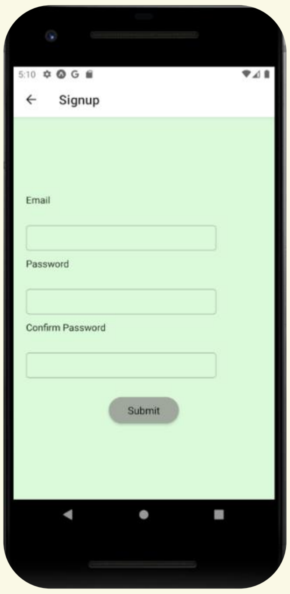
a. API End points

Method	Description	URI
POST	User sign up	/users/signup
POST	User login	/users/signin
GET	Get location data for user and device	/users/data
GET	Add device	/users/devices
GET	Add location	/users/locations
DELETE	Remove Device	/users/remove-device
POST	Authentication of the device	/device/authenticate
GET	Finish bluetooth handling	/data/stopAdding

b) Sign up and Login





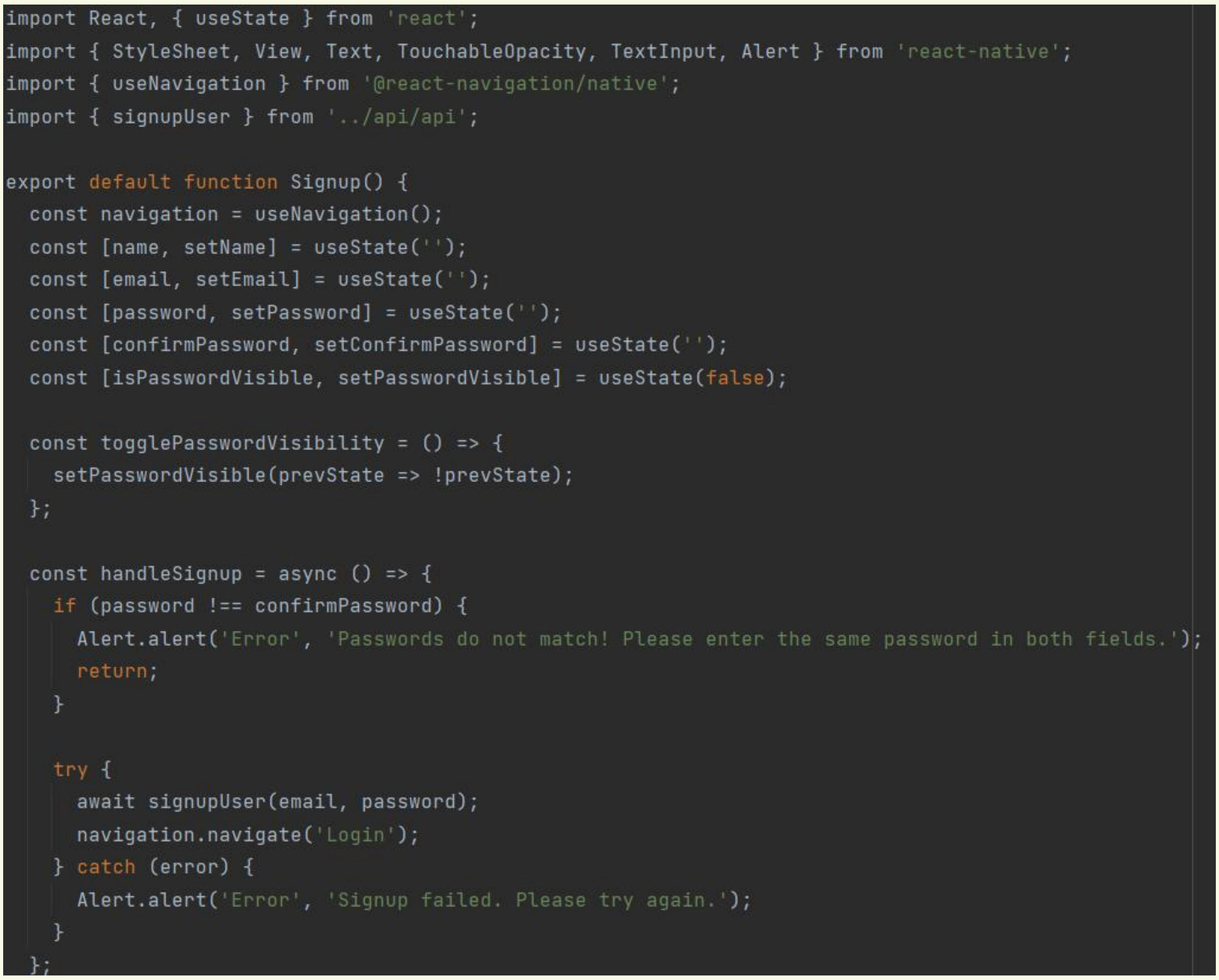


Sign up and login instructions:

You will be first directed to the Home page where you will be able to login and sign up.
If you are already a registered user, you can login to the application using the correct email and password.
If you are a new user, go to the sign up page and create an account. In order to create an account, enter the email, password and re-enter password.
The password should contain characters and symbols with acceptable length.

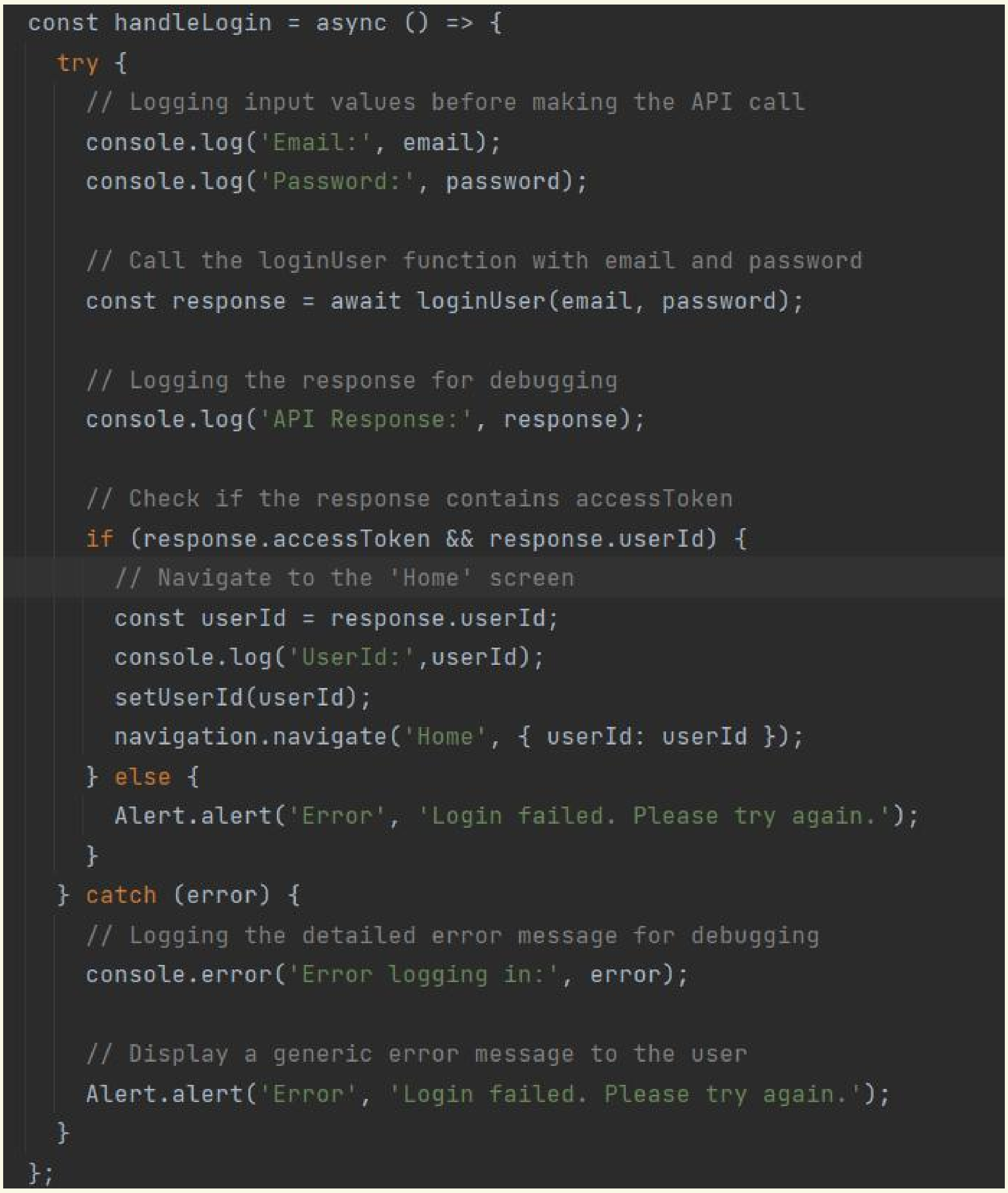
Extracted code segments

Signup

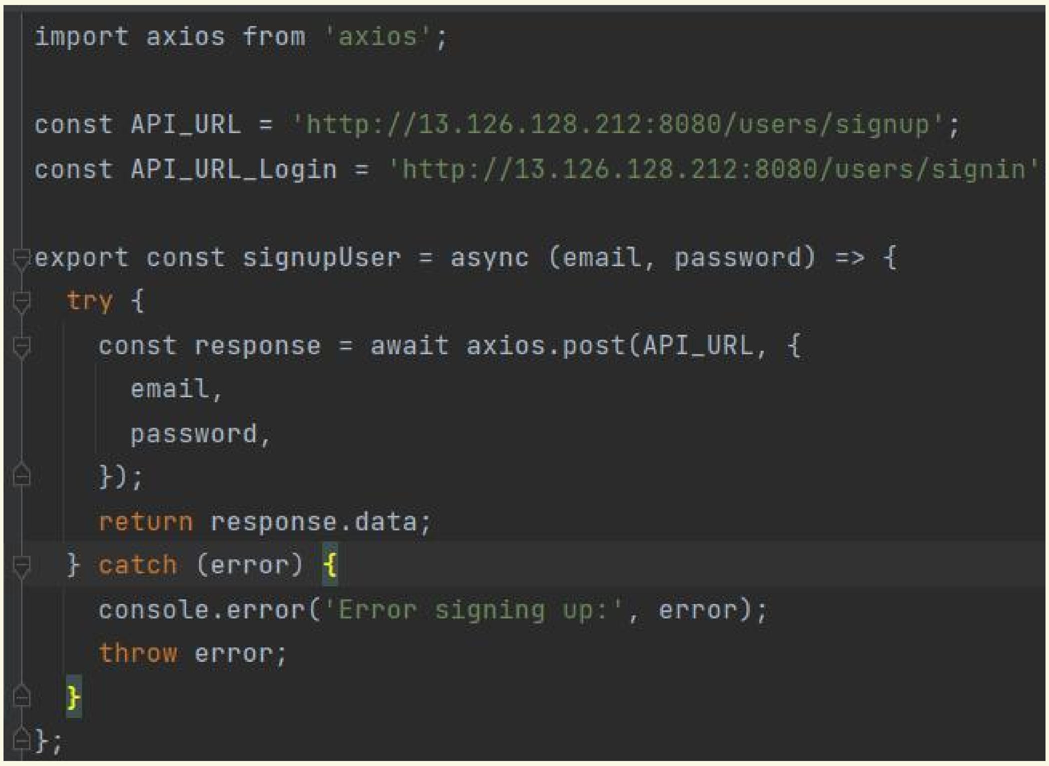


```
import React, { useState } from 'react';
import { StyleSheet, View, Text, TouchableOpacity, TextInput, Alert } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import { signupUser } from '../api/api';

export default function Signup() {
  const navigation = useNavigation();
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [isPasswordVisible, setPasswordVisible] = useState(false);

  const togglePasswordVisibility = () => {
    setPasswordVisible(prevState => !prevState);
  };

  const handleSignup = async () => {
    if (password !== confirmPassword) {
      Alert.alert('Error', 'Passwords do not match! Please enter the same password in both fields.');
      return;
    }

    try {
      await signupUser(email, password);
      navigation.navigate('Login');
    } catch (error) {
      Alert.alert('Error', 'Signup failed. Please try again.');
    }
  };
```

LOGIN



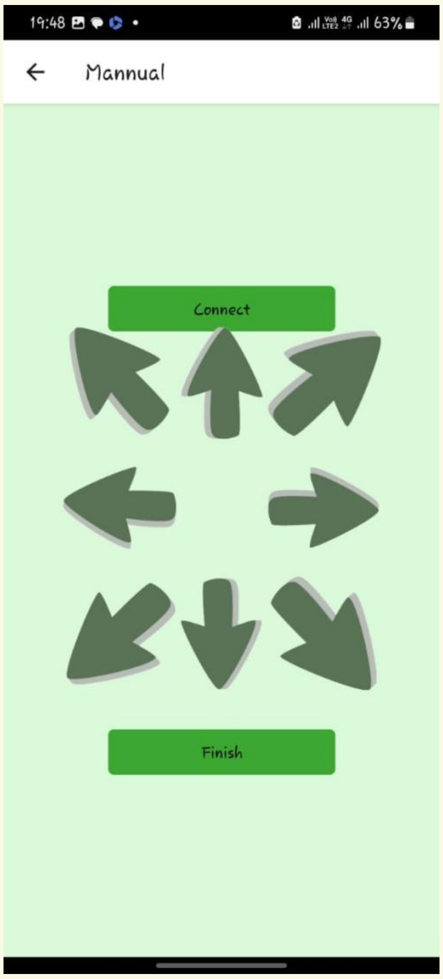
```
const handleLogin = async () => {
  try {
    // Logging input values before making the API call
    console.log('Email:', email);
    console.log('Password:', password);

    // Call the loginUser function with email and password
    const response = await loginUser(email, password);

    // Logging the response for debugging
    console.log('API Response:', response);

    // Check if the response contains accessToken
    if (response.accessToken && response.userId) {
      // Navigate to the 'Home' screen
      const userId = response.userId;
      console.log('UserId:',userId);
      setUserId(userId);
      navigation.navigate('Home', { userId: userId });
    } else {
      Alert.alert('Error', 'Login failed. Please try again.');
    }
  } catch (error) {
    // Logging the detailed error message for debugging
    console.error('Error logging in:', error);

    // Display a generic error message to the user
    Alert.alert('Error', 'Login failed. Please try again.');
  }
};
```



```
import axios from 'axios';

const API_URL = 'http://13.126.128.212:8080/users/signup';
const API_URL_Login = 'http://13.126.128.212:8080/users/signin'

export const signupUser = async (email, password) => {
  try {
    const response = await axios.post(API_URL, {
      email,
      password,
    });
    return response.data;
  } catch (error) {
    console.error('Error signing up:', error);
    throw error;
  }
};
```

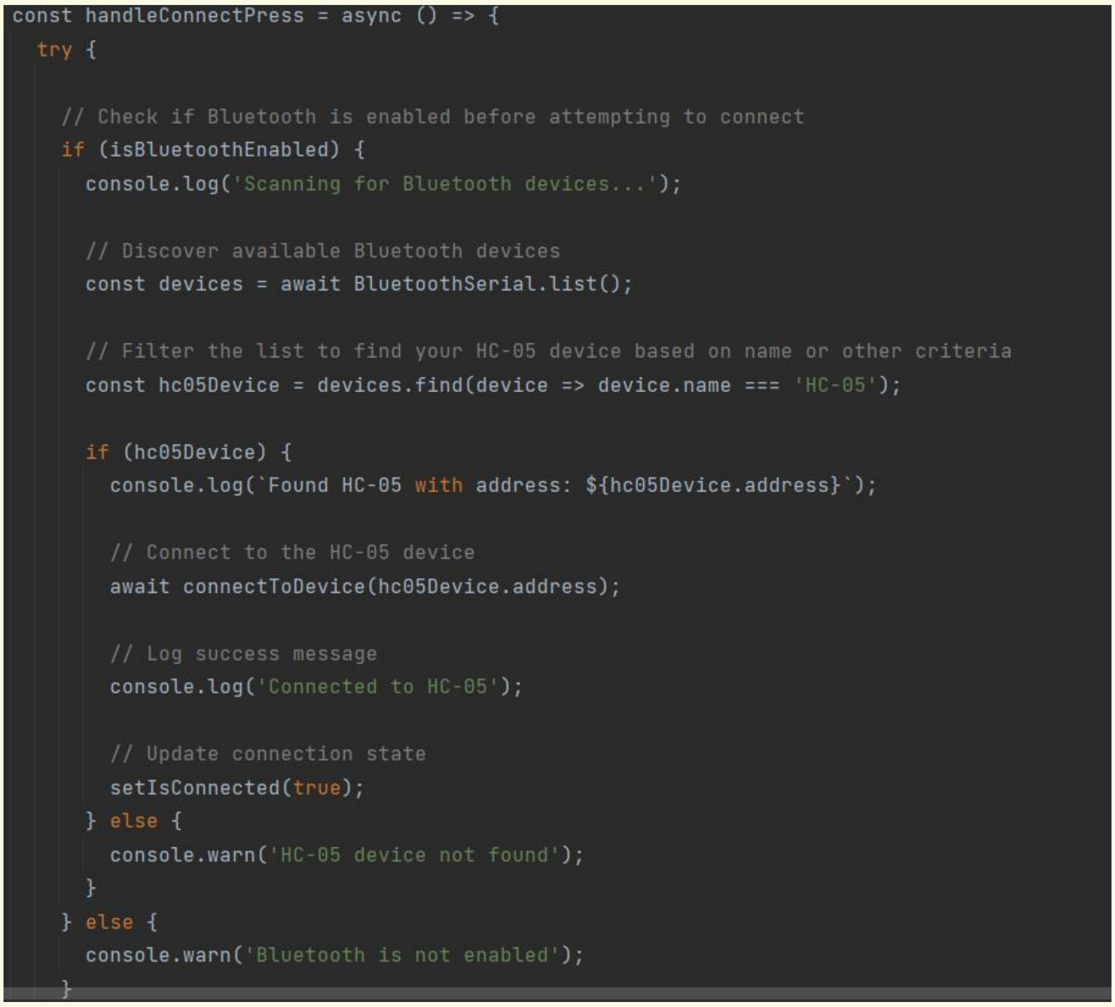
Connection of end points for login and sign up.

c) Bluetooth handling



Get connected to bluetooth and control the device using these eight buttons.
After controlling click on “Finish”.

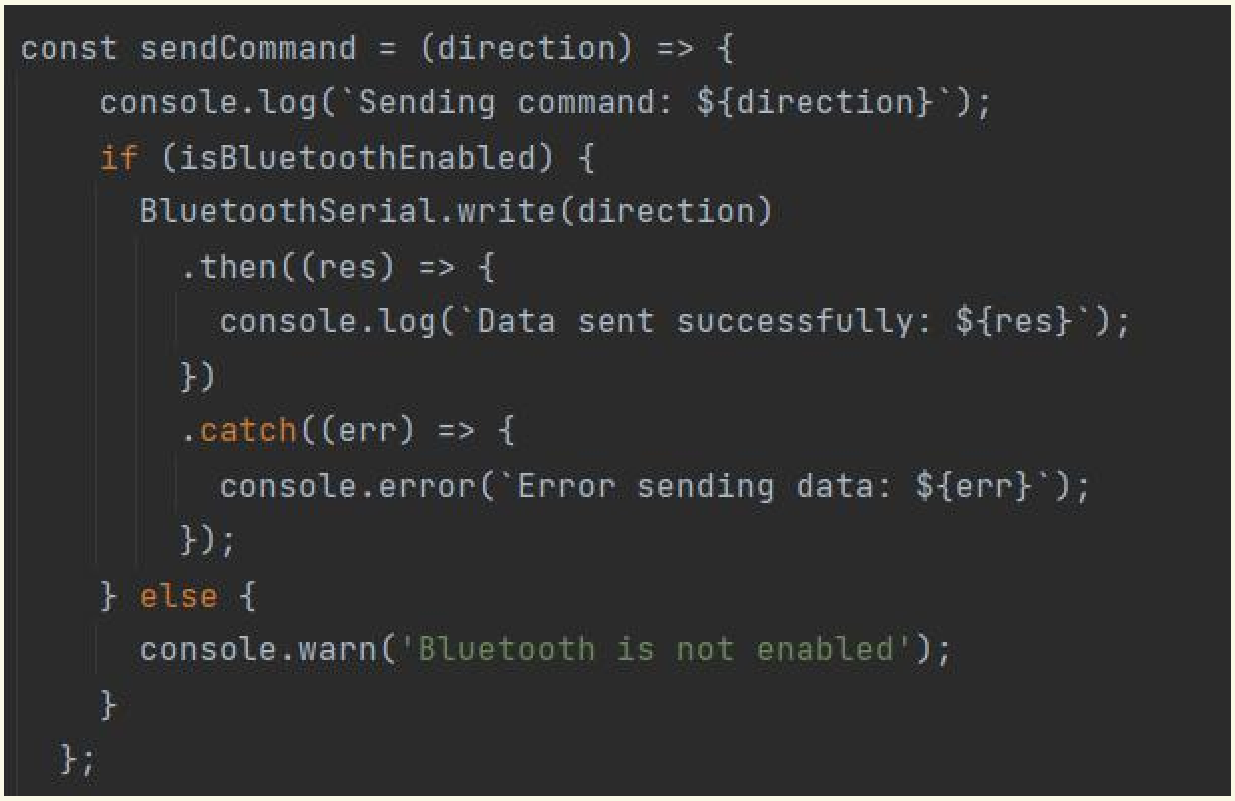
Extracted code segments



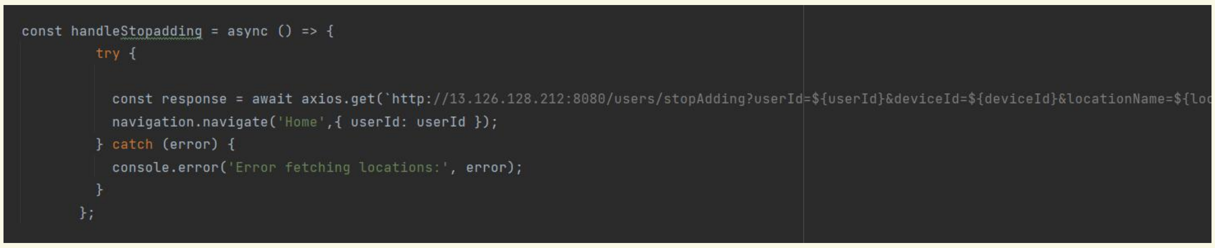
```
const handleConnectPress = async () => {
  try {

    // Check if Bluetooth is enabled before attempting to connect
    if (isBluetoothEnabled) {
      console.log('Scanning for Bluetooth devices...');

      // Discover available Bluetooth devices
      const devices = await BluetoothSerial.list();

      // Filter the list to find your HC-05 device based on name or other criteria
      const hc05Device = devices.find(device => device.name === 'HC-05');

      if (hc05Device) {
        console.log(`Found HC-05 with address: ${hc05Device.address}`);

        // Connect to the HC-05 device
        await connectToDevice(hc05Device.address);

        // Log success message
        console.log('Connected to HC-05');

        // Update connection state
        setIsConnected(true);
      } else {
        console.warn('HC-05 device not found');
      }
    } else {
      console.warn('Bluetooth is not enabled');
    }
```

Connect bluetooth

Sending commands to bluetooth



```
const sendCommand = (direction) => {
    console.log(`Sending command: ${direction}`);
    if (isBluetoothEnabled) {
      BluetoothSerial.write(direction)
        .then((res) => {
          console.log(`Data sent successfully: ${res}`);
        })
        .catch((err) => {
          console.error(`Error sending data: ${err}`);
        });
    } else {
      console.warn('Bluetooth is not enabled');
    }
  };
```



```
const handleStopadding = async () => {
        try {

          const response = await axios.get(`http://13.126.128.212:8080/users/stopAdding?userId=${userId}&deviceId=${deviceId}&locationName=${loc
          navigation.navigate('Home',{ userId: userId });
        } catch (error) {
          console.error('Error fetching locations:', error);
        }
      };
```

Stops adding datas related to manual mapping when you click on “FINISH” button

DATABASE

MongoDB Atlas, the cloud-based version of MongoDB, is the database system used in this project. MongoDB is a NoSQL database, which means it does not use a fixed schema and can store data in various formats. This allows for greater flexibility and ease of scalability, as the data model and formats can be modified without impacting applications. Additionally, MongoDB Atlas is a cross-platform document-oriented database, which means it stores data in a format similar to JSON, making it easy to
work with for developers.

One of the key benefits of MongoDB Atlas is its scalability. The database can be expanded horizontally by adding more machines to the resources, making it easy to handle large amounts of data and high traffic. It also supports integrated caching, which improves the performance of data output by utilizing system memory. This is particularly useful for applications with large amounts of data or high traffic.

MongoDB Atlas also has built-in security features such as user authentication and access controls, making it easy to secure data and protect against unauthorized access. Additionally, MongoDB Atlas supports automatic backups and disaster recovery, ensuring that data is always available and can be easily recovered in the event of a failure.

In summary, MongoDB Atlas is a powerful, flexible and scalable database system that is well suited for this project. It allows for easy modification of the data model and formats without impacting applications and provides robust security features, automatic backups and disaster recovery, and integrated caching to boost performance.

SECURITY

In order to ensure the security and integrity of our application, we have implemented several key security features. First and foremost, all user accounts are password protected. This means that only authorized users will be able to access the application and sensitive information. Additionally, we have implemented user authentication to ensure that only verified and legitimate users are able to access the application. This is done by requiring users to provide valid login credentials before granting access.

We also ensure data integrity by implementing appropriate measures to protect against unauthorized access, modification or deletion of data. This includes implementing strict access controls and regular monitoring and auditing of data access.

To ensure the security of the data stored in our application, we have also built in AWS protection. This includes implementing security best practices, such as encryption and hashing.

Overall, these security features are in place to protect the sensitive data and information stored in the application and to ensure that the application can only be accessed by authorized users.

TESTING

We used postman, a powerful API testing tool, to test the RESTful APIs that are used in our application. This allows us to ensure that the APIs are functioning correctly and that they are able to handle a wide range of inputs and scenarios.

Also, we used MQTT test client, a tool or software used to simulate MQTT clients for testing and debugging purposes.