

SMART PARKING SYSTEM PROJECT REPORT

TEAM MEMBERS

Akhil Kodalapuram 20BEE0094

Akshaya V S 20BEC1139

Kumar Krishav 20BEC0494

**Sreemathi Sivakumar
20BEC1174**



CONTENTS

TITLE	Page No.
CHAPTER 1: INTRODUCTION	
1.1 Overview	3
1.2 Purpose	3
CHAPTER 2: LITERATURE SURVEY	
2.1 Existing Problem	5
2.2 Proposed Solution	5
CHAPTER 3: THEORETICAL ANALYSIS	
3.1 Block Diagram	7
3.2 Hardware and Software Designing	7
3.2.1 ESP32	7
3.2.2 HC-SR04 Ultrasonic Sensor	8
3.2.3 LED	9
3.2.4 Resistor (200 Ohms)	9
3.2.5 LCD Screen (20 X 4)	9
3.2.6 Jumper wires	10
3.2.7 WOKWI Simulator	10
3.2.8 IBM Watson Iot Platform	11
3.2.9 Node-Red	11
3.2.10 Firebase Real-time Database	11
3.2.11 MIT App Inventor	12
CHAPTER 4: EXPERIMENTING INVESTIGATIONS	
4.1 Block Diagram	13
4.2 Hardware Experimental Analysis	13
4.3 Software Experimental Analysis	14

CHAPTER 5: FLOWCHART

5.1 Overview 18

5.2 Hardware Control Flow 18

5.3 User Interface Control Flow 19

CHAPTER 6: RESULTS 21-34

CHAPTER 7: ADVANTAGES AND DISADVANTAGES

7.1 Advantages 35

7.2 Disadvantages 35

CHAPTER 8: APPLICATIONS 37

CHAPTER 9: CONCLUSIONS 38

CHAPTER 9: FUTURE SCOPE 39

BIBLIOGRAPHY 40

APPENDIX 41-53

CHAPTER 1

INTRODUCTION

1.1 Overview

The proposed smart parking system designed using the Internet of Things (IoT) is an advanced solution that leverages connected devices and sensors to optimize the management and utilization of parking spaces and help users and parking space providers to connect with one another through an app and provide ease of access.

Our project uses an ESP32 microcontroller board with ultrasonic sensors which detect the presence or absence of vehicles and transmit the corresponding data to a cloud-based database (IBM Cloud and Google Firebase), which is then analyzed using Node-Red. Furthermore, an application is built using MIT App Inventor, which serves as the sole User Interface for the users and the consumers of this system. The app provides the users with distinctive features such as Paid Parking Space reservation, finding Parking spaces near you as well as a Machine Learning driven Parking Data analytics.

1.2 Purpose

The purpose of a smart parking system is to address the challenges and inefficiencies associated with traditional parking management. Here are the key purposes of implementing a smart parking system:

- **Optimize Parking Space Utilization:** By providing real-time information about parking availability, drivers can quickly locate vacant spots, reducing the time spent searching for parking. This optimization helps to minimize congestion and enhances the overall efficiency of parking operations.

- **Enhance User Experience:** By providing real-time parking availability information and guidance, drivers can plan their parking in advance, saving time and reducing frustration. Features like reservation and mobile payment options also make parking more convenient and user-friendly.
- **Reduce Traffic Congestion:** Inefficient parking practices, such as circling around in search of parking, contribute to traffic congestion in urban areas. Smart parking systems help alleviate this congestion by enabling drivers to quickly find available parking spaces, reducing unnecessary traffic circulation and associated emissions.
- **Increase Revenue Generation:** For parking lot owners and operators, implementing a smart parking system can lead to increased revenue. Additionally, features like reservation systems and dynamic pricing enable operators to generate additional income streams and maximize their revenue potential.
- **Efficient Resource Management:** By analyzing data on parking patterns and demand fluctuations, operators can make informed decisions on resource allocation, such as adjusting staffing levels, optimizing parking lot layout, and planning maintenance activities. This improves operational efficiency and reduces costs.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Problem

Pham et al. (2015) present a cloud-based smart parking system that utilizes RFID (Radio Frequency Identification) technology for vehicle identification and access control. RFID has limited read range and storage capacity and is significantly expensive.

A study by Aydhin et al. (2017) proposes a hybrid approach combining genetic algorithm and fuzzy logic to optimize parking space allocation based on factors such as distance, parking duration, and vehicle type. Implementation of this technique is complex and time-consuming, and the results are often difficult to interpret.

Abdulkader et al. (2018) proposed an integrated parking management system that combines various technologies, including wireless sensor networks, mobile applications, and data analytics. Wireless sensor networks rely on battery power, and battery life is limited. Also deploying and maintaining multiple wireless sensor networks can be complex and expensive, hence reducing its feasibility.

2.2 Proposed Solution

Our Smart Parking system, ‘Park It’, consists of 4 ultrasonic sensors that record the real-time availability of parking slots, and this data is sent to the IBM cloud and Google Firebase by ESP32 controller for storage.

Ultrasonic sensors provide accurate readings of the presence/absence of vehicles and are cost-efficient, easy to install, and consume low power. Integrating with ESP32 allows the system to scale up to cover a larger area.

The collected data from the sensors and user interactions can be analyzed to gain insights into parking patterns, peak hours, and demand fluctuations. A mobile

application is created for reservation and payment options. Drivers can reserve a parking spot in advance, and payments can be made digitally, enabling seamless transactions.

CHAPTER 3

THEORETICAL ANALYSIS

3.1 Block Diagram

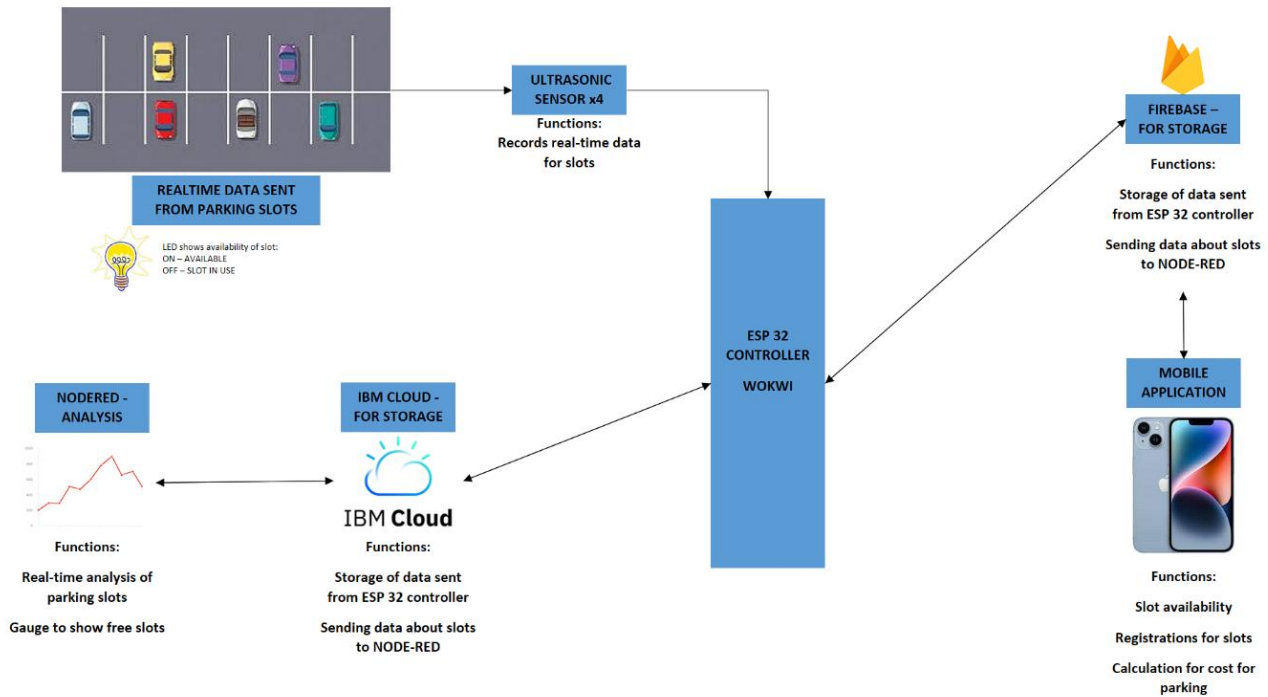


Figure 1. Block Diagram

This block diagram represents the flow of data from the parking lot to the various nodes of the system. The ultrasonic sensor collects the data of the occupied slot and sends it to the ESP32 Microcontroller. The data is sent to the cloud storage nodes, namely IBM Watson Cloud and Google Firebase. The data is then sent to the IBM cloud node in JSON format and Firebase in String format. According to the node, the data has transverse to, we go either the Real-time analysis of the requited data or the display of the same in the Mobile application.

3.2 Hardware and Software Designing

3.2.1 ESP32

ESP32 is a low-cost, low-power Microcontroller with an integrated Wi-Fi and Bluetooth. It is the successor to the ESP8266 which is also a low-cost Wi-Fi

microchip albeit with limited vastly limited functionality.

It has an integrated antenna and RF transmitter, power amplifier, low-noise amplifiers, filters, and power management module. The entire solution takes up the least amount of printed circuit board area. This board is used with 2.4 GHz dual-mode Wi-Fi and Bluetooth chips by TSMC 40nm low power technology, power and RF properties best, which is safe, reliable, and scale-able to a variety of applications. It consists of 34 GPIO pins.

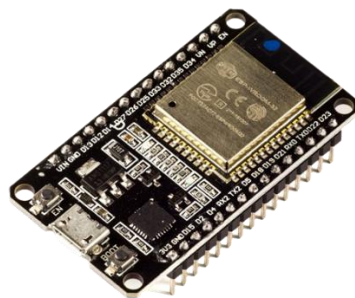


Figure 2. ESP32

3.2.2 HC-SR04 Ultrasonic Sensor

The HC-SR04 is a type of ultrasonic sensor which uses sonar to find out the distance of the object from the sensor. It provides an outstanding range of non-contact detection with high accuracy & stable readings. It includes two modules like ultrasonic transmitter & receiver. This sensor is used in a variety of applications like measurement of direction and speed, burglar alarms, medical, sonar, humidifiers, wireless charging, non-destructive testing, and ultrasonography.



Figure 3. HC-SR04 sensor

3.2.3 LED

Light-emitting diode (LED) is a widely used standard source of light in electrical equipment. It has a wide range of applications ranging from your mobile phone to large advertising billboards. They mostly find applications in devices that show the time and display different types of data. Here, we use LEDs to guide us to show an empty parking slot



Figure 4. LED

3.2.4 Resistor (200 Ohms)

A resistor is an electrical component that limits or regulates the flow of electrical current in an electronic circuit. Here, we use 200 Ohms resistors to limit the amount of current passing through the LED so as to not burn it.



Figure 5. 200 Ohm Resistor

3.2.5 LCD Screen (20 X 4)

The 20x4 LCD display module with high speed I2C interface, is able to display 20x4 characters on two lines, white characters on blue background.

It needs 6 digital pins and 2 power pin for a LCD display. Owing to its I2C interface, only 2 lines (I2C) are required to display the information hence utilizing lesser number of GPIO pins.



Figure 6. 20X4 LCD Display

3.2.6 Jumper wires

The jumper wires here are used for giving connection from our microcontroller the ESP32 and the the different sensors.



Figure 7. Jumper wires

3.2.7 WOKWI Simulator

The Wokwi Arduino Simulator is a great online tool for simulating projects that use a development board. The app supports various popular development boards and MCUs such as the Arduino UNO, the Raspberry Pi Pico, the ESP32, and the ATtiny85.

The simulation program uses a straightforward workflow. One needs to define the behavior of the development board using source code, and then add the necessary components to the design. The resistor values have to be defined in the tool's text editor. The text editor offers advanced code suggestions and a handy

autocomplete mechanism. Furthermore, one can add external libraries to the project to extend the functionality of the design. The program accurately replicates how parts would behave in an actual setup, and hence is a great alternative.

3.2.8 IBM Watson Iot Platform

IBM Watson® IoT Platform is a fully managed, cloud-hosted service that makes it simple to derive value from Internet of Things (IoT) devices. By simply registering and connecting the device, be it a sensor, a gateway, or something else, to the Watson IoT Platform one can start sending data securely up to the cloud using the open, lightweight MQTT messaging protocol. One can set up and manage devices using the online dashboard or IBM's secure APIs so that the apps can access and use your live and historical data.

3.2.9 Node-Red

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single click. JavaScript functions can be created within the editor using a rich text editor. Here, we have used Node Red to visualize our parking data and create innovative and eye-catching Dashboards

3.2.10 Firebase Real-time Database

The Firebase Real time Database is a cloud-hosted database. Data is stored as JSON and synchronized in real time to every connected client. When the user builds cross-platform apps with the Apple platforms, Android, and JavaScript SDKs, all of the user's clients share one Real time Database instance and automatically receive updates with the newest data. Instead of typical HTTP requests, the Firebase Real time Database uses data synchronization—every time data changes, any connected device receives that update within

milliseconds. Provide collaborative and immersive experiences without thinking about networking code. We have used the Firebase as our extensive database for storing user as well as sensor data since firebase has lot of crucial capabilities

3.2.11 MIT App Inventor

MIT App Inventor is a web-based visual development environment that allows users to create mobile apps for Android devices. App Inventor provides a simplified drag-and-drop interface that enables individuals with little to no programming experience to design and build functional applications. The platform utilizes a block-based programming language similar to Scratch, where users can connect different blocks to create the desired functionality of their app. It also offers a visual interface where users can arrange components, such as buttons, labels, and images, on a virtual phone screen. Blocks Editor is used to define the behavior of the app. Users can use the built-in emulator or connect an Android device to test and see how their app functions in real-time.

CHAPTER 4

EXPERIMENTAL ANALYSIS

4.1 Overview

Our hardware model of the proposed system consists of four parking slots whose occupancy is monitored continually through the Ultrasonic sensors positioned on each of the slots. Furthermore, an LED is attached near each of the Parking spaces. These LEDs glow when the space is empty, hence guiding the user to find an empty parking slot with ease.

The Software model of the proposed system is a user-friendly app that assists the users in locating nearby parking areas, reserving parking slots along with the reservation cost calculator, and provides the users with efficient data analytics, which will aid them in planning the parking type during their time of visit.

During the implementation of the model it is noticed that Internet connectivity plays a crucial role for real-time updating of data to IBM and Firebase as well as the app functionalities. It is also observed that the OCR recognition extension and the QR generator fail to work in the apk version of the app since the extensions are not supported anymore. Hence, these shortcomings were noted down and should be informed to the users in prior.

4.2 Hardware Experimental Analysis

Initially, the circuit is set up such that the Ultrasonic sensors continuously detect the proximity of objects present in their allocated parking space and that all the slots occupancy variables (U1, U2, U3, U4) in the Firebase are set to 0, depicting that they are empty. The total free slots are set to 4, as per the circuit, and the individual slot variable depicted by S1, S2, S3, and S4 are set to 0, depicting that all slots are free. The LEDs corresponding to each of the slots are set to be ON since all the slots are initialized to be free. These details can also be viewed on the LCD Display for user convenience.

Now, for each slot, the following procedure is looped throughout the time.

When the detected distance between the object and the sensor is less than 50 cm, it is understood that a vehicle is parked in the slot, and hence the ESP32 Board checks if the slot was previously occupied or not. If the slot was previously not occupied, the corresponding LED is switched OFF, and the number of free slots is reduced by one. The corresponding individual slot variable (S) is updated to 1 to depict that the slot is occupied. A timer is started to measure the amount of time the slot is occupied by the particular user. All the relevant details are also updated on the LCD screen for user convenience. The corresponding slot occupancy variable (U) is updated from 0 to 1 in the Firebase to depict that the slot is occupied.

If the slot was previously occupied, the board keeps updating the corresponding slot occupancy variable (U) to 1 in the Firebase to avoid any discrepancies.

When the detected distance between the object and the sensor is greater than 50 cm, it is understood that there is no vehicle parked in the slot, and hence the ESP32 Board checks if the slot was previously empty or not. If the slot was previously occupied, the corresponding LED is switched ON, and the number of free slots is increased by one. The corresponding individual slot variable (S) is set to 0 to depict that the slot is free. The timer is turned off, and the amount of time that the slot was occupied is sent to IBM IoT Watson Studio. All the relevant details are also updated on the LCD screen for user convenience. The corresponding slot occupancy variable (U) is updated from 1 to 0 in the Firebase to depict that the slot is free.

If the slot was previously free, no changes are carried out.

The number of free slots is updated to IBM Iot Watson every 10 seconds.

4.3 Software Experimental Analysis

The unique and classy color palette and Graphic designs help make a long-lasting impact on the users. The brand logo of our proposed system 'Park It' is displayed on the app startup. Next, the user is directed to the login page. Here returning users can enter their Username and Password. The entered password is checked if it matches the username registered in our Firebase Database. If it matches, the user is directed to the page where the features of the app are displayed. If the details do not match, a warning is given stating the same and the user is denied further access to the app until the right password is entered.

New users can click on the Sign-Up button, where their details will be collected and they can set up their username and password. Only if the user types their password twice and they both match, will they be allowed to proceed further with vehicle registration. The Password will be stored to the Corresponding Username in the Firebase Database.

Under the Vehicle Registration page, the user will be asked about their vehicle type, which must be chosen from the drop-down menu, and their license plate number. For users who use the Open Camera app as their default Camera app, they can opt to click a picture of their license plate and the embedded Optical Character Recognition system will automatically extract the characters, hence reducing the amount of time taken during registration.

After the Login or Registration Procedure, the users are directed to the features page, where the users can choose to either Find available parking areas near them, Reserve Parking slots or get to know about the slot availability and traffic through our Parking Analytics by clicking a corresponding button.

To find parking areas near the current location, the user must navigate to that page and then choose their preferred location. Their current location is depicted by a green marker, whereas the available parking areas are depicted by multiple red markers. The user will get navigation guides to reach the chosen parking area.

The user can also get an audio guide for the same by clicking on the provided button. This is done using the Text to Speech feature embedded in the app.

To Reserve Parking slots, the user must navigate to the corresponding page from the features page. The available slots will be green in color and the already occupied or reserved slots will be red in color. Users can click on the button present above a free slot to reserve the slot. They can long press the same button to undo the selection. If a user tries to long press an already occupied slot, the slot will remain Red in color, meaning that slot cannot be reserved. This is done because the ESP32 board keeps updating the slot variable (U) to 1 (occupied) in the Firebase Database. Once the user selects a slot, the slot variable (U) for the corresponding slot is automatically updated to 1 signifying that it is occupied. Once the User confirms their reservation, they are redirected to the cost calculator page, where they can select the time till which they want their slot to be reserved. According to this duration, the fee is calculated internally. The user can then confirm the reservation by generating a QR code for validation procedures. The user will then be directed to the Slot Reservation confirmation page.

Parking analytics is a crucial tool for users and building management systems to understand the number of free slots available at specific times, the congestion probability, how to distribute the traffic, etc. Hence, we also provide the users with in-depth parking analytics performed using various Machine Learning Techniques. It consists of a graph showing the number of vehicles parked in the parking lot throughout the week for 5 weeks and a brief description regarding the same. Using this, the users can appropriately adjust their trips and reserve parking slots beforehand.

We have used the Node-Red platform to visualize real-time data and to help analyze the same. The data is then visualized through the Node-Red dashboard

using the dashboard nodes and chart node. The flow and the dashboard are as follows:

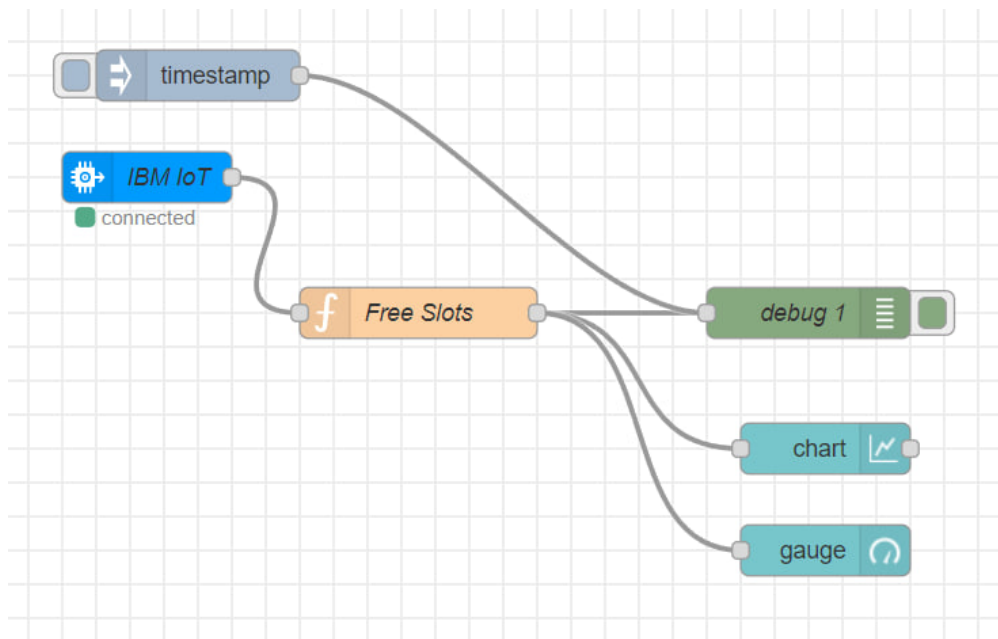


Figure 8. Node-Red Flow

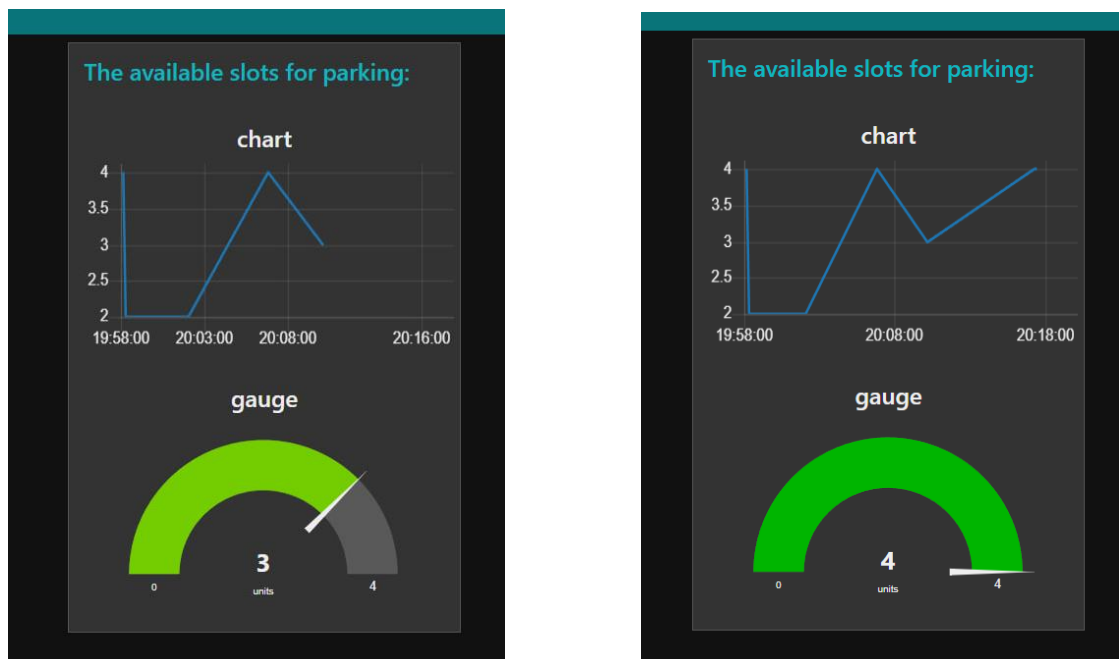


Figure 9. This shows the available number of slots for parking

CHAPTER 5

FLOWCHART

5.1 Overview

Owing to the high complexity of the proposed system, the control flow has been divided into two sub flows for ease of understanding and implementation. These sub flows are:

- Hardware Control Flow
- User Interface Control flow

5.2 Hardware Control Flow

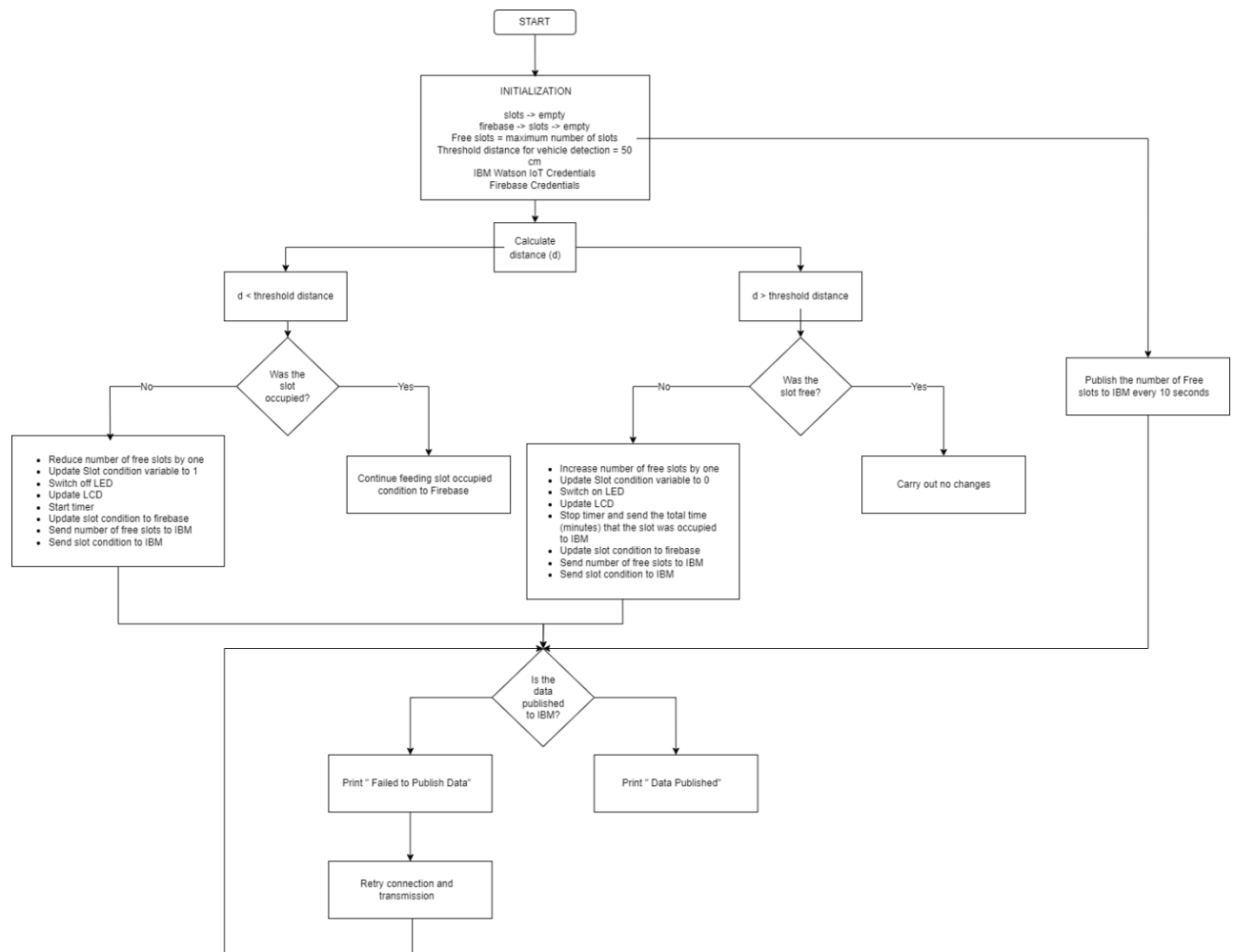


Figure 10. Hardware Control Flow

The above diagram depicts the flow of control to different components of the circuit and the data transfer according to the outcome of the previous step. The components controlled by the above flowchart are:

- Hardware Circuit
- IBM Watson IoT
- Firebase Database
- Node Red

5.3 User Interface Control Flow

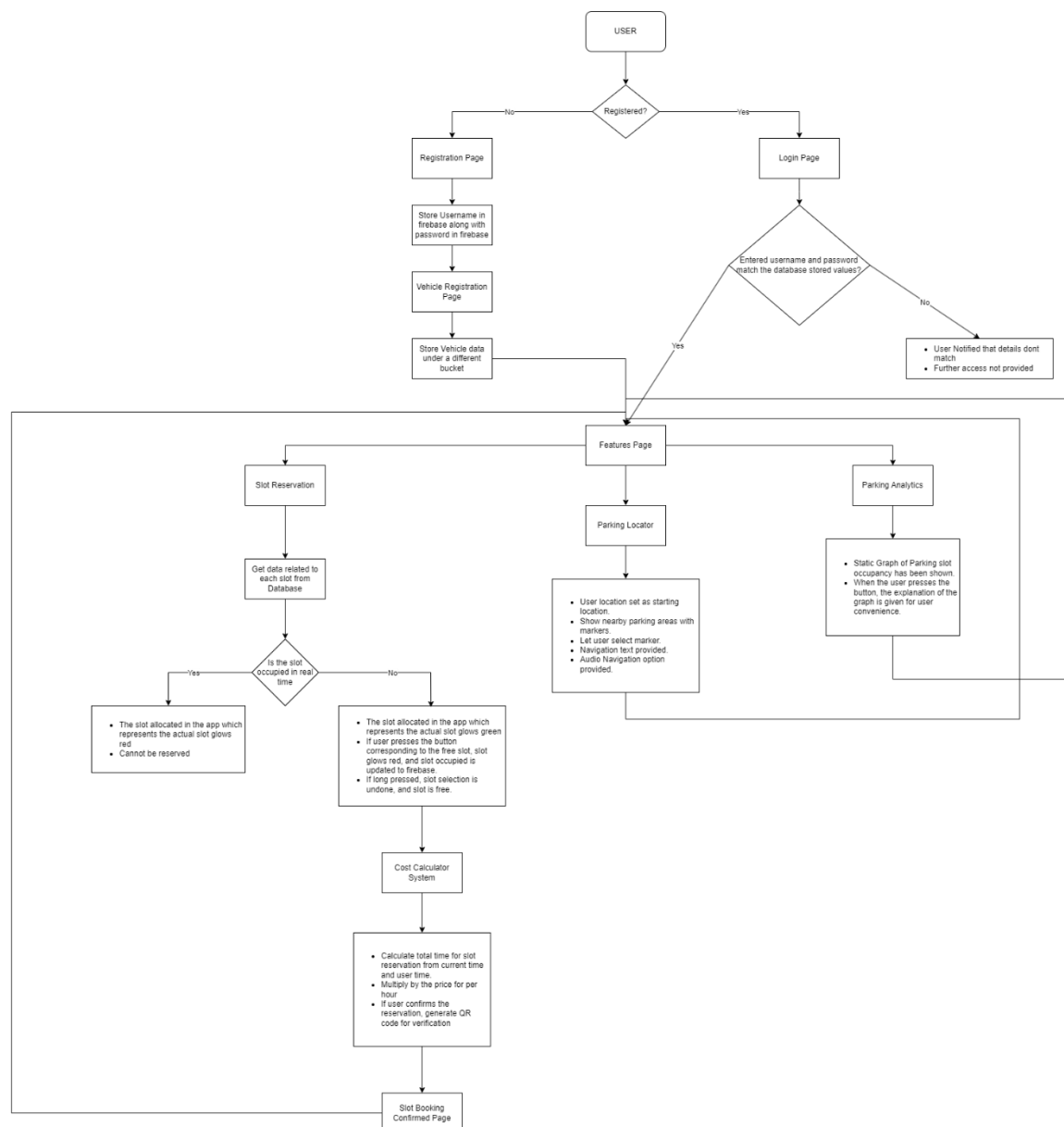


Figure 11. User Interface Control flow

The above diagram depicts the flow of control through the app as the user navigates through it. The flow is designed for smooth navigation and user friendliness as well as keeping in mind the secure storage and retrieval of the User data. The flow maintains real time connectivity to the Firebase database and keeps updating its information. The components controlled by the above flowchart are:

- Park It Mobile app
- Firebase Database

CHAPTER 6

RESULTS

Hardware Circuit Diagram (WOKWI)

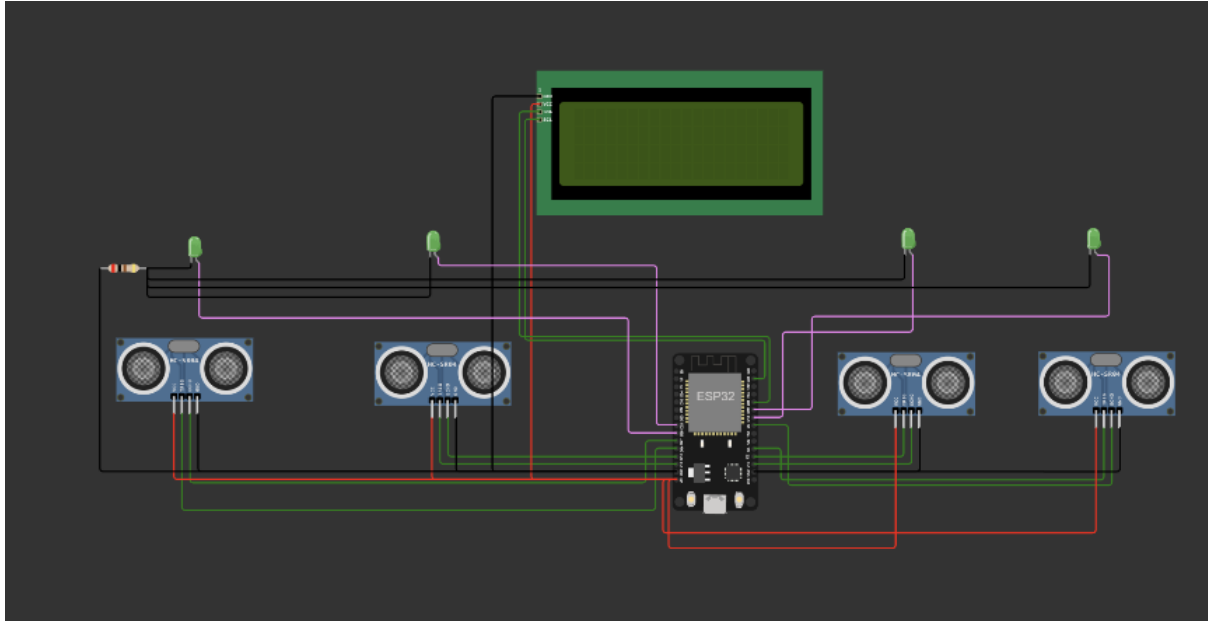


Figure 12. Circuit Diagram

Initial Circuit Conditions (WOKWI)

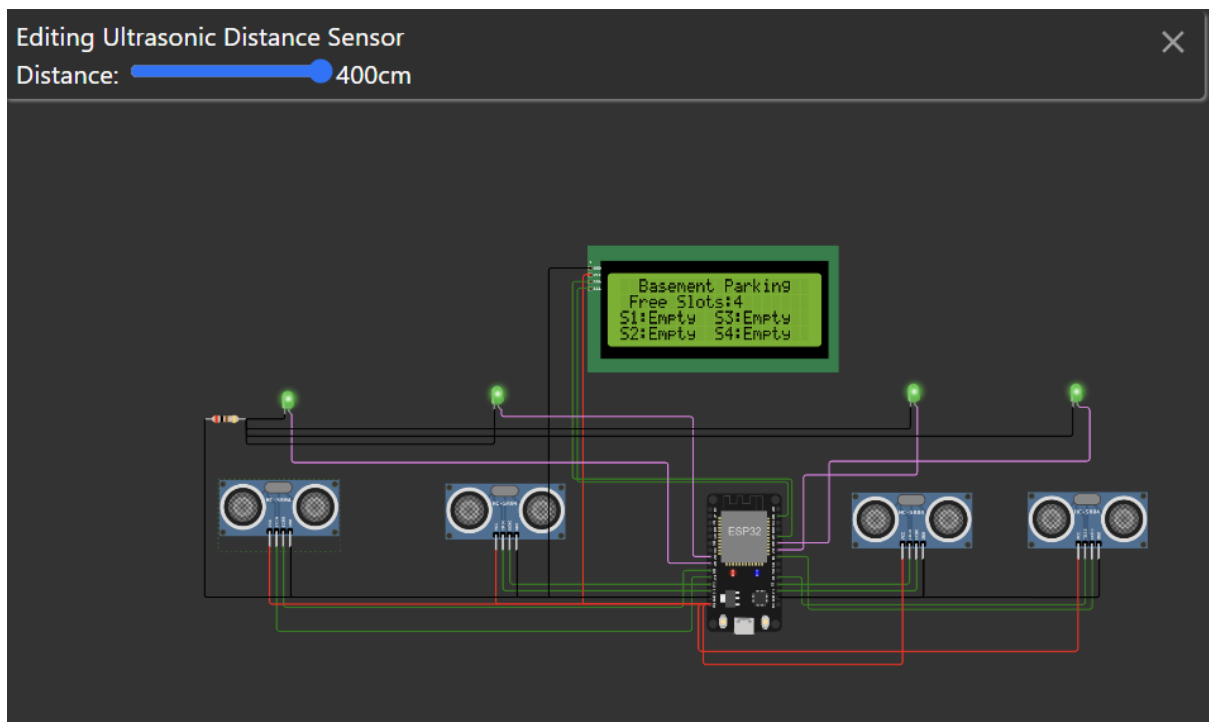


Figure 13. Initial Circuit with all slots free and LEDs glowing

Initial Database showing Slot Occupancy Variables (Firebase)

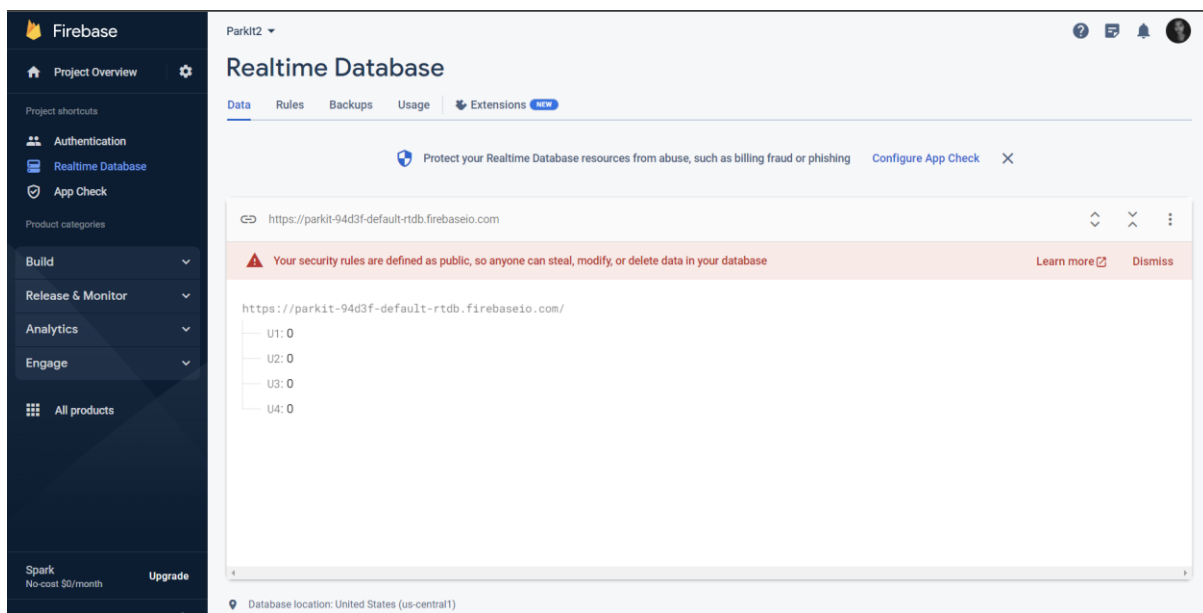


Figure 14. Initial Database with all slots free

Publishing of Free Slots to IBM Watson IoT every 10 Seconds (IBM)

Device ID

Status

Device Type

▼

112233

Connected

smartparking

Identity

Device Information

Recent Events

State

Logs

The recent events listed show the live stream of data that is coming and going from this device.

Event

Value

Format

Last Received

FreeSlots

{"free_slots":4}

json

a few seconds ago

FreeSlots

{"free_slots":4}

json

a few seconds ago

Figure 15. Free Slots data received in IBM Watson IoT

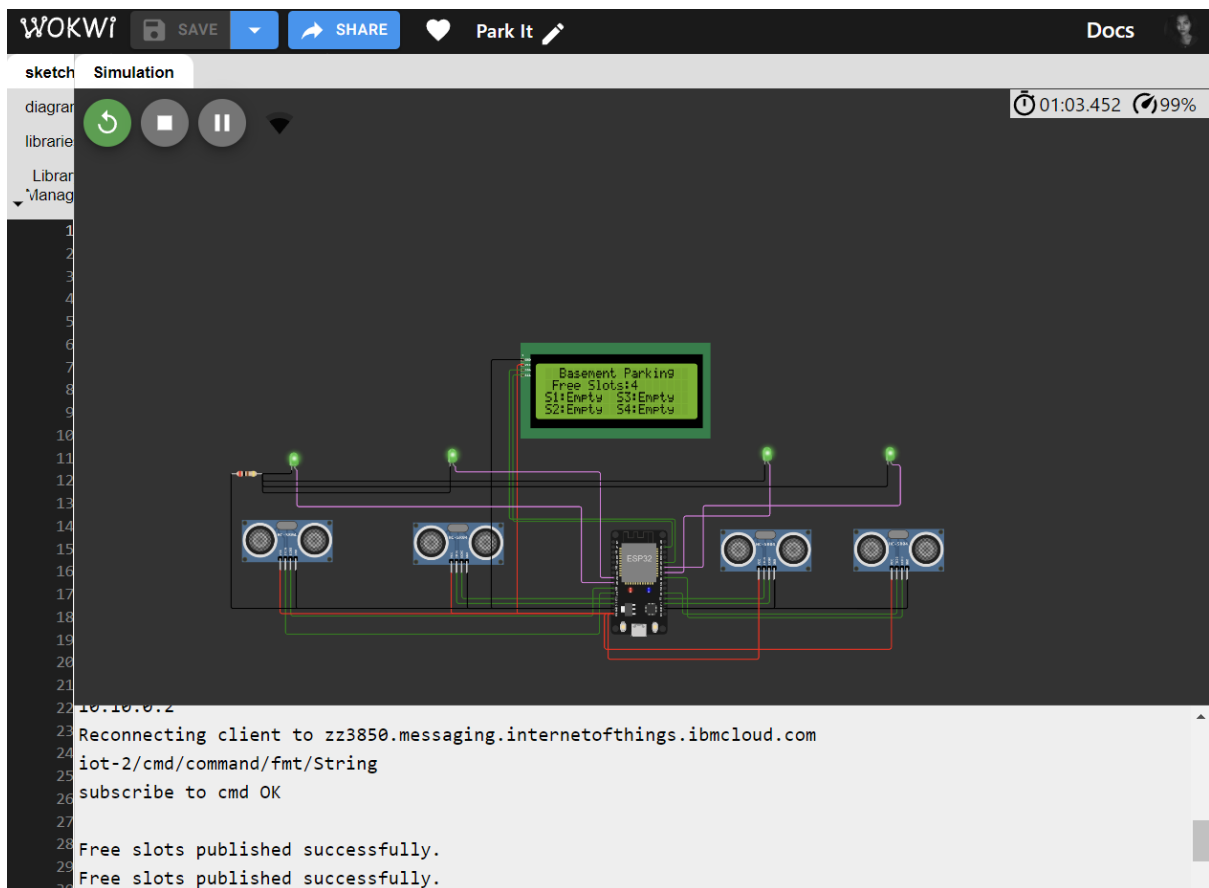


Figure 16. Free Slots data successfully published every 10 seconds

Let us assume that someone has parked their car in Slot 1 (Leftmost)

First slot distance less than 50 cm (threshold distance), so car is parked in slot 1

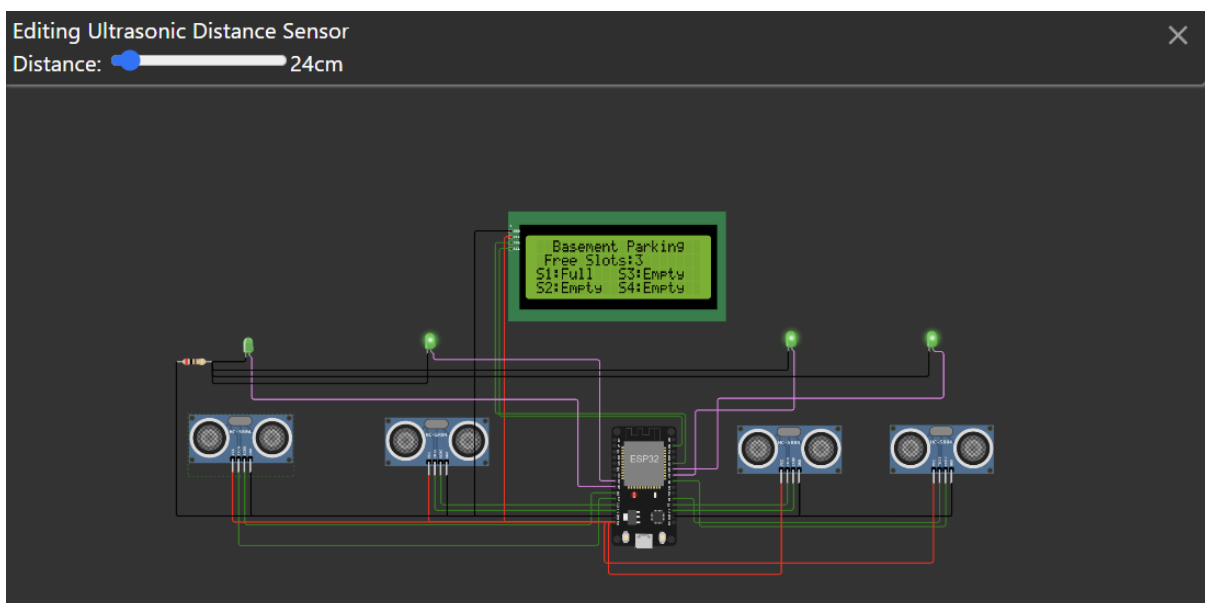


Figure 17. Slot 1 Occupied

LED turned off for Slot 1 as it is occupied

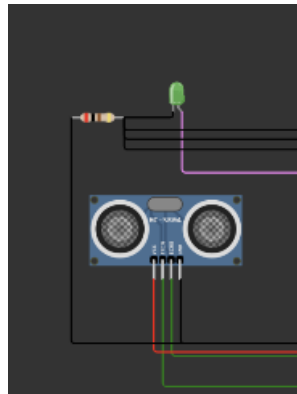


Figure 18. LED off

LCD updated to real-time values

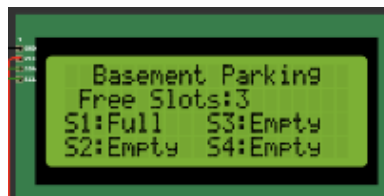


Figure 19. Updated LCD Screen

Publishing of updated Free Slots to IBM Watson IoT every 10 Seconds (IBM)

<input type="checkbox"/>	Device ID	Status	Device Type
▼ <input type="checkbox"/>	112233	● Connected	smartparking
Identity Device Information Recent Events State Logs			
The recent events listed show the live stream of data that is coming and going from this dev			
Event	Value	Format	
FreeSlots	{"free_slots":3}	json	

Figure 20. Free Slots data received in IBM Watson IoT

Database with updated Slot Occupancy Variable for slot 1 (U1) to 1 (occupied) (Firebase)

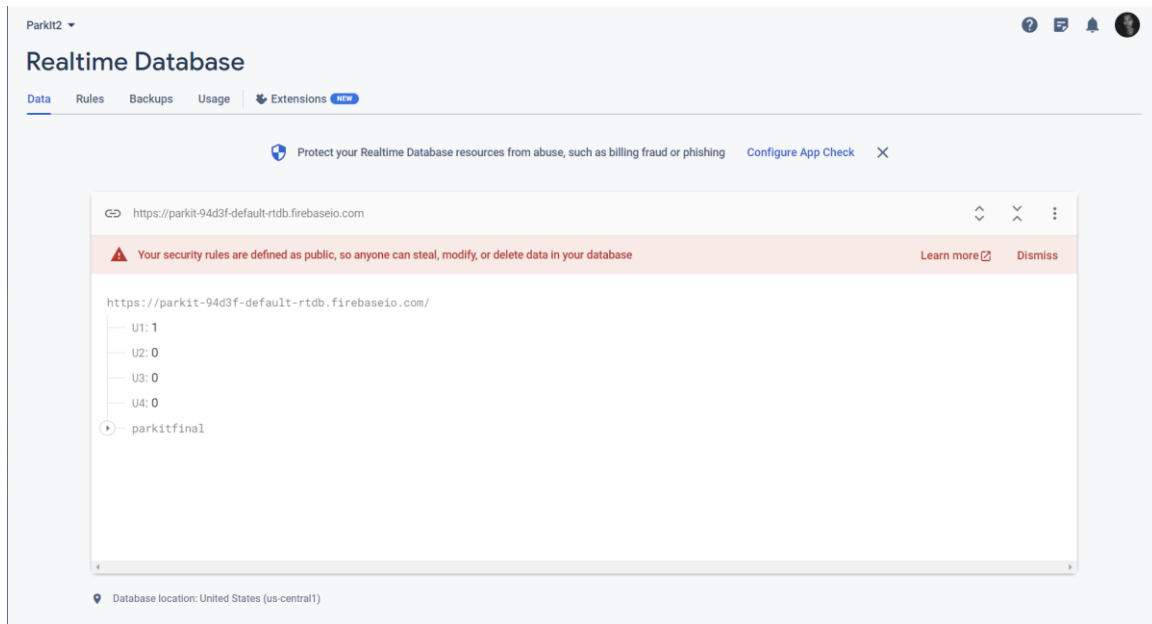


Figure 21. Updated Database with all Slot 1 Occupied (U1 = 1)

Slot 1 Occupied at 11:42 pm

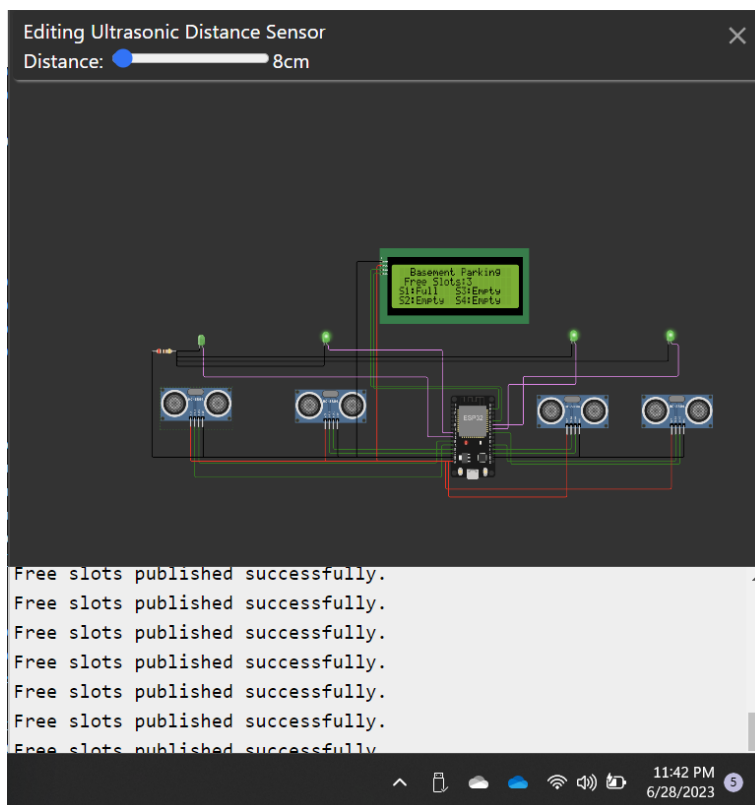


Figure 22. Slot occupied at 11:42

Slot 1 free at 11:43 pm

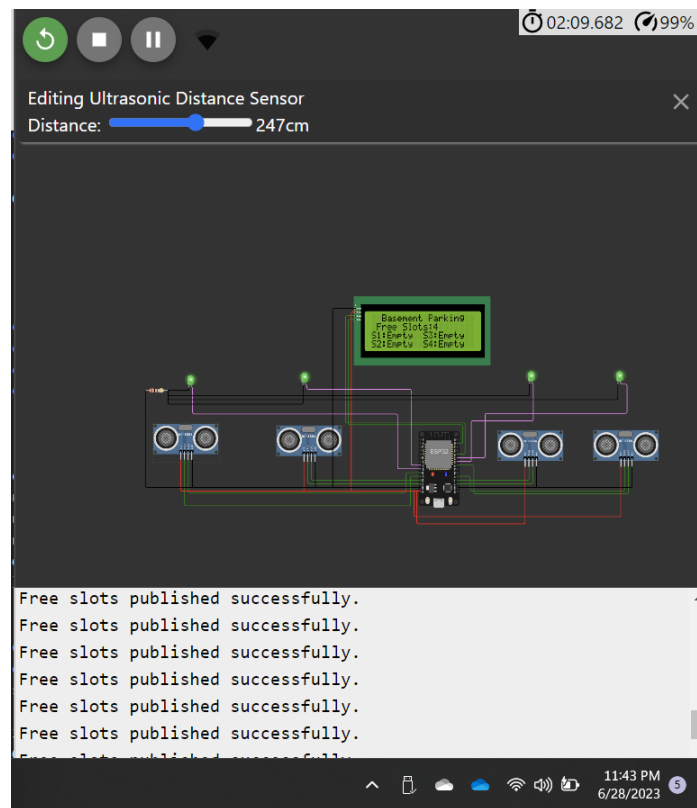


Figure 23. Slot free at 11:43

Duration for which Slot 1 was occupied

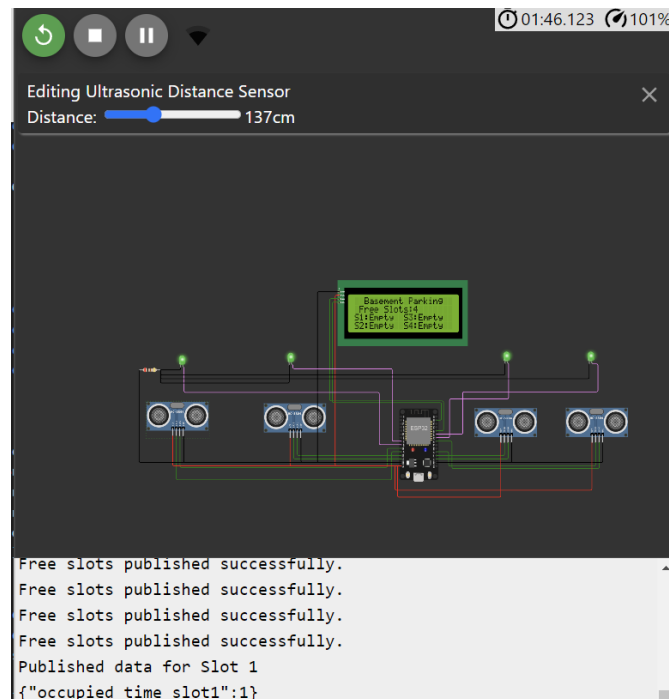


Figure 24. Slot 1 occupied for 1 minute

Duration sent to IBM Watson

<input type="checkbox"/>	Device ID	Status	Device Type
▼ <input type="checkbox"/>	112233	● Connected	smartparking
Identity Device Information <u>Recent Events</u> State Logs			
The recent events listed show the live stream of data that is coming and going from this device.			
Event	Value	Format	Last Rec
Data	{"occupied_time_slot1":1}	json	a few s
FreeSlots	{"free_slots":3}	json	a few s

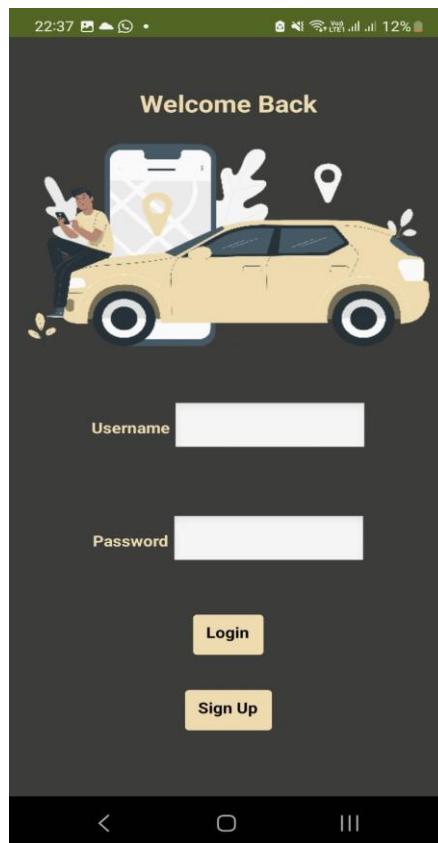
Figure 25. Slot 1 Occupied duration sent to IBM

Park It App Initiation (MIT APP INVENTOR)



Figure 26. App Front Page with logo and tagline

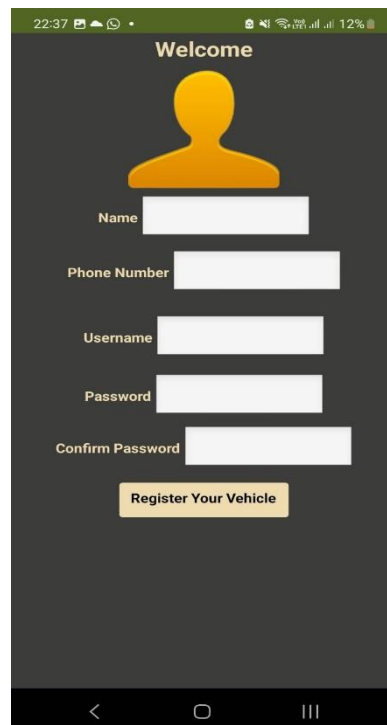
Log in Page for returning users



A mobile app login screen with a dark grey background. At the top, the status bar shows the time 22:37, battery level 12%, and various icons. Below the status bar, the text "Welcome Back" is displayed in a light yellow font. Underneath, there is an illustration of a yellow car with a person sitting in the driver's seat, holding a smartphone. The car is surrounded by white location pins and a white document icon. Below the illustration, there are two input fields: "Username" and "Password", both with light grey text labels and white input boxes. Below the "Password" field, there are two buttons: "Login" and "Sign Up", both with yellow text on a dark grey background. At the bottom of the screen, there is a navigation bar with three icons: a back arrow, a home circle, and a menu hamburger icon.

Figure 27. Log in Page

Registration Page for New users



A mobile app registration screen with a dark grey background. At the top, the status bar shows the time 22:37, battery level 12%, and various icons. Below the status bar, the text "Welcome" is displayed in a light yellow font. Underneath, there is an illustration of a yellow person silhouette. Below the illustration, there are five input fields: "Name", "Phone Number", "Username", "Password", and "Confirm Password", all with light grey text labels and white input boxes. Below the "Confirm Password" field, there is a button labeled "Register Your Vehicle" with yellow text on a dark grey background. At the bottom of the screen, there is a navigation bar with three icons: a back arrow, a home circle, and a menu hamburger icon.

Figure 28. Registration Page

Vehicle Registration Page for New users with OCR (Capture Plate)

The left screenshot shows the 'Vehicle Registration' page. It features an illustration of a yellow car in a city. Below the illustration, the text 'Vehicle Registration' is displayed. A button labeled 'Proceed with Vehicle Registration' is present. Below this, there is a 'Vehicle Type' section with a 'Choose' button and a hint 'Hint for TextBox4'. A 'License Plate' section has a 'Plate Number' input field. At the bottom, there are 'Capture Plate' and 'Accept' buttons.

The right screenshot shows a list of vehicle types for selection. The list includes: Bike, Rickshaw, Car, Van, Bus, and Truck. A search bar at the top is labeled 'Search list...'.

Figure 29. Vehicle Registration Page

Firebase with data

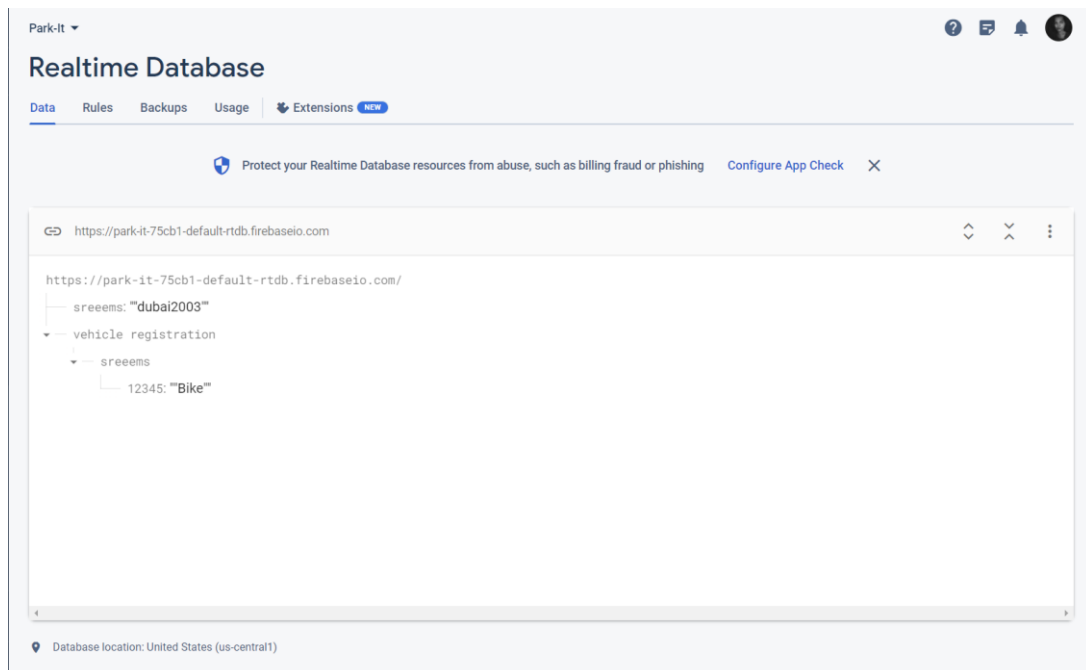


Figure 30. Firebase with username, password and vehicle details stored

Features Page

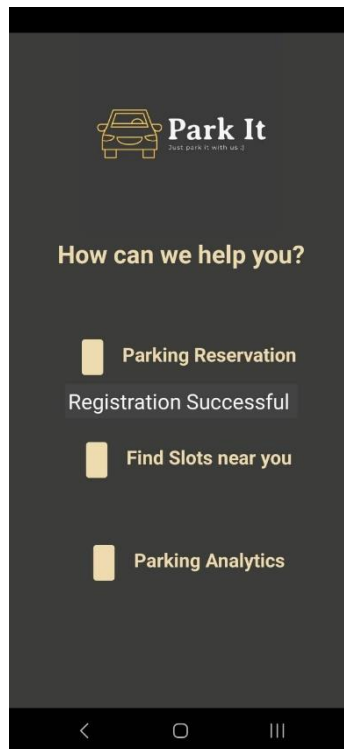


Figure 31. Features Page

Parking Reservation Page

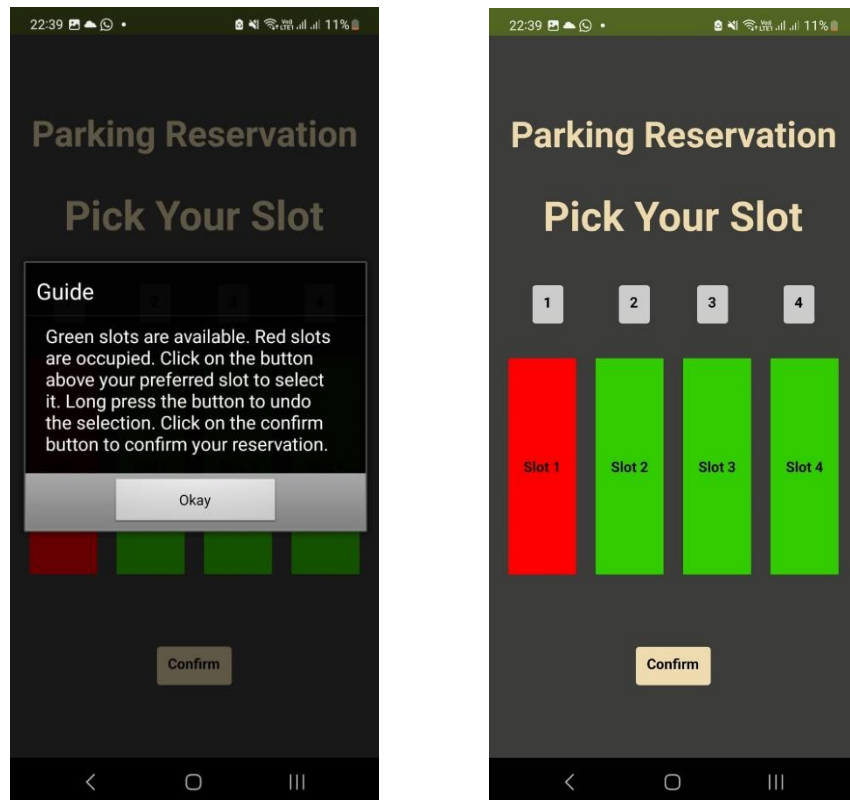


Figure 32. Parking Reservation Page

Slot 2 is reserved along with Firebase

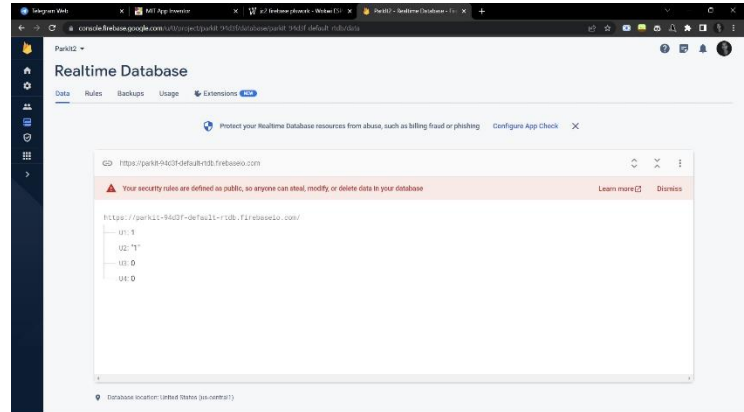
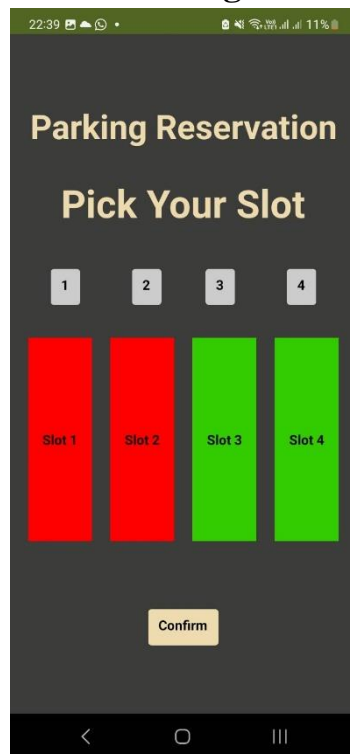


Figure 33. Slot 2 Reservation done by pressing button 2

Undoing Slot 2 Reservation along with Firebase

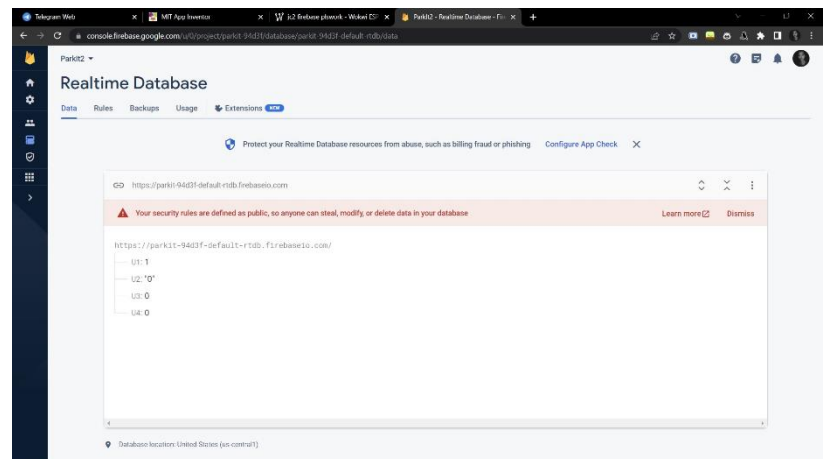
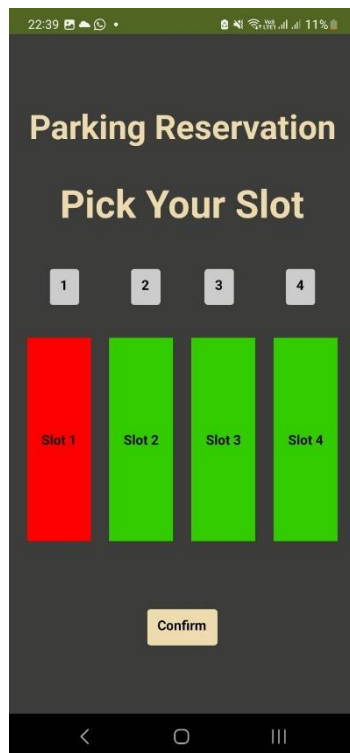


Figure 34. Slot 2 Reservation undone by long pressing button 2

Undoing Slot 1 Reservation

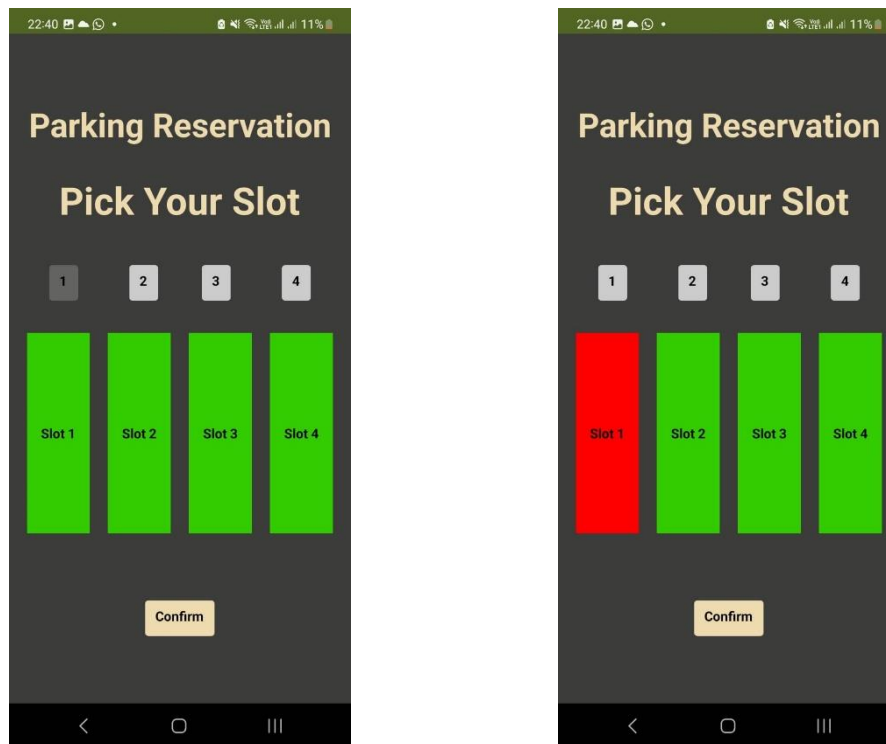


Figure 35. Slot 1 Reservation cannot be undone by long pressing button 1

Cost Calculator and QR Page

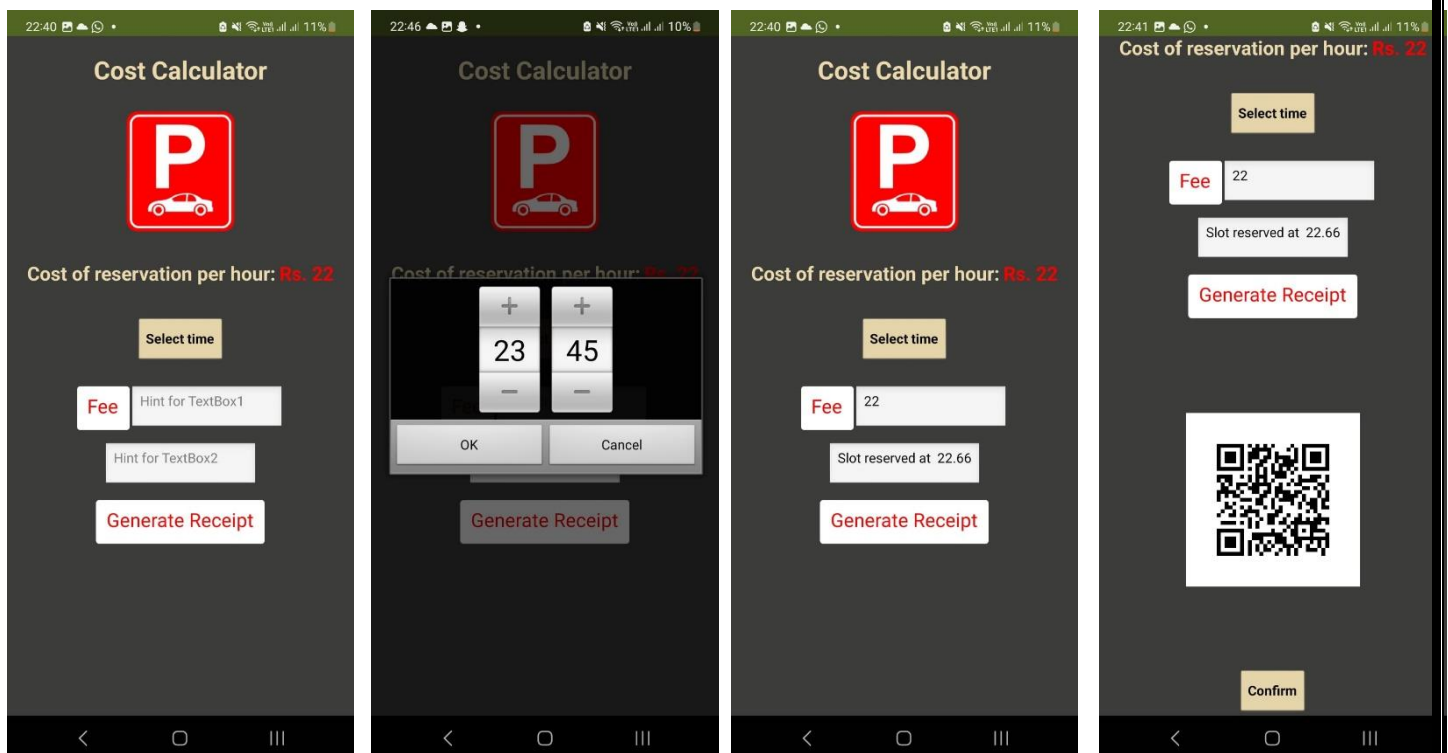


Figure 36. Cost Calculator and QR code generator

Slot booked page

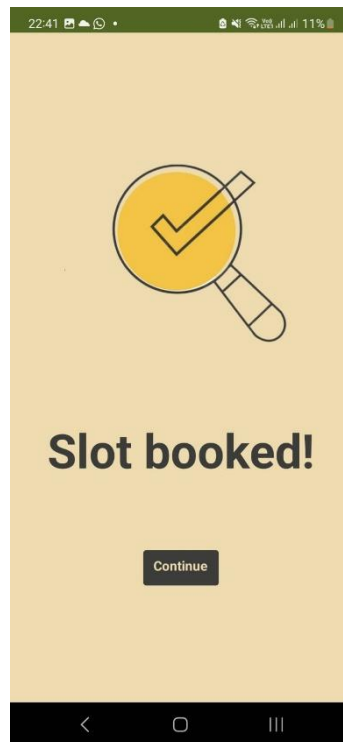


Figure 37. Slot Reservation Confirmation Page

Parking Slots near you

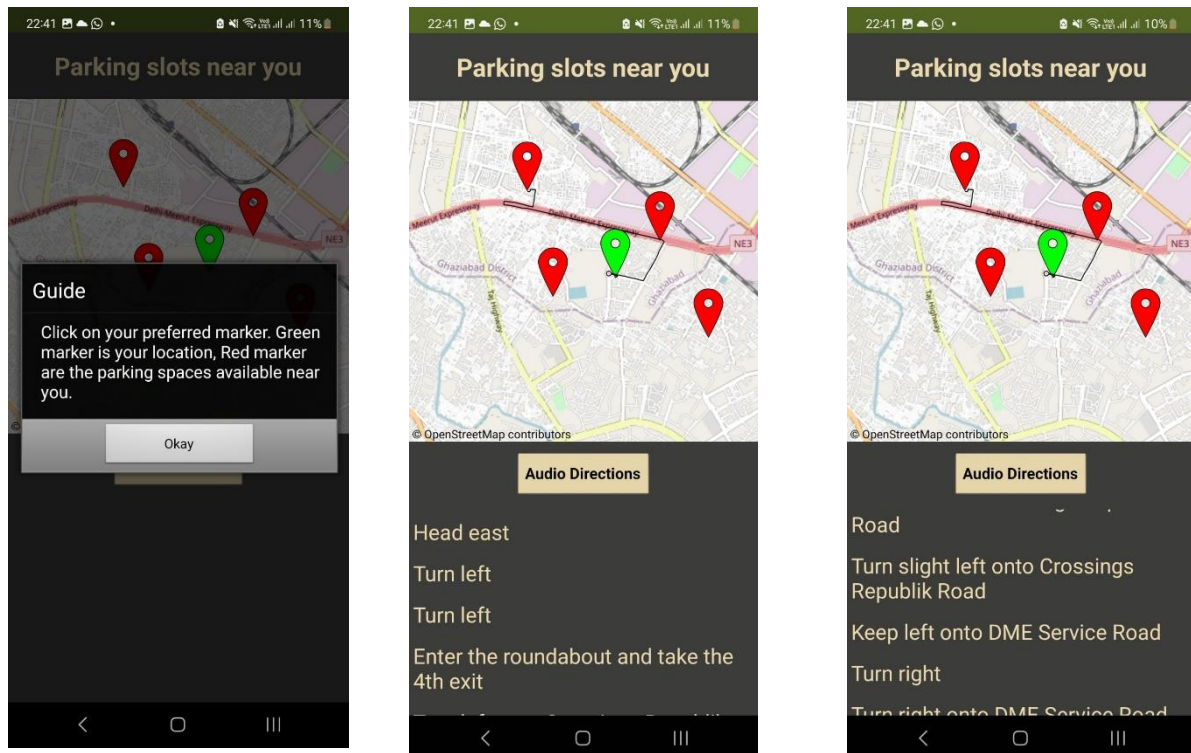


Figure 38. Parking Slots available near user with navigation

Parking Analytics

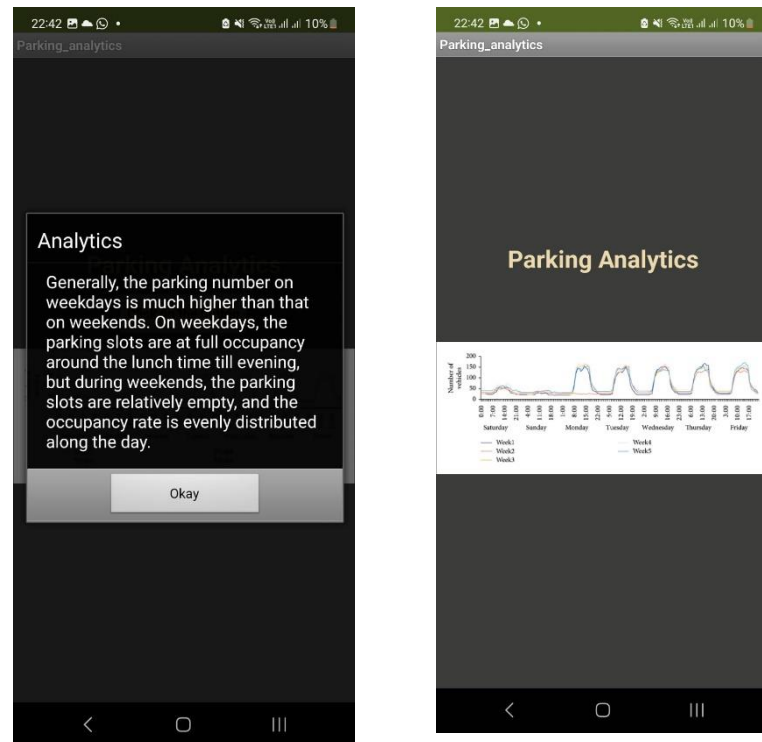


Figure 39. Parking Analytics for user convinience

CHAPTER 7

ADVANTAGES AND DISADVANTAGES

7.1 Advantages

The advantages that our system brings to the people are as follows:

- **Enhanced Efficiency:** Our Smart parking system utilizes advanced technologies such as sensors, cameras, and data analytics to efficiently manage parking spaces. It enables real-time monitoring of parking occupancy, allowing drivers to quickly find available spaces, reducing the time spent searching for parking.
- **Improved User Experience:** With a smart parking system, drivers can access real-time information about available parking spaces through mobile apps.
- **Optimal Space Utilization:** This Smart parking system can optimize the allocation of parking spaces, ensuring that each space is used efficiently. By accurately monitoring occupancy, the system can identify vacant spaces and guide drivers to those areas.
- **Real-time Monitoring and Management:** Smart parking systems provide real-time data on parking occupancy, which allows operators to monitor usage patterns, identify trends, and make data-driven decisions.
- **Data-driven Insights:** The collection of parking data in a smart parking system can provide valuable insights for urban planning and future infrastructure development. Analyzing parking patterns, demand trends, and user behavior can help cities make informed decisions regarding parking policies, infrastructure expansion, and transportation planning.

7.2 Disadvantages

Following mentioned are some problems people could face using the parking system

- **Technical Challenges:** Smart parking systems rely on technology, and like any technology (wokwi), they can experience technical issues. Malfunctions, connectivity problems, or software bugs can disrupt the system's effectiveness and cause inconvenience to users.
- **Reliance on Connectivity:** Smart parking systems depend on stable and reliable connectivity to function effectively. If there are connectivity issues or network outages, the system's performance may be affected, and real-time data updates may not be available.
- **Limited Accessibility:** Users/People who are not yet comfortable with the emerging technologies or unable to keep up with the technological advancements may face issues or may not be able to understand the system's working.
- **Adoption Challenges:** Introducing a smart parking system may face resistance or challenges in terms of user adoption. Some drivers may be resistant to change or unfamiliar with using mobile apps or digital platforms to find parking spaces.
- **High Initial Investment:** Implementing a smart parking system typically requires significant upfront investment in infrastructure, including sensors, communication networks, and software. The cost may be a barrier for smaller parking facilities or organizations with lower budget or limited budget.
- **Extension Difficulties relating to MIT App Inventor:** Few of the extensions used in the app built in this proposed system are not supported in higher android versions (OCR) and apk (QR Generator).

CHAPTER 8

APPLICATIONS

The following are some of the real-life applications of the smart parking system

- **Urban Parking Management:** Smart Parking systems can be integrated with parking meters and payment systems to enable seamless payment and improve overall parking management. This will help reduce the congestion on roads by guiding the drivers to find the parking spot easily.
- **Commercial Parking Facilities:** Smart parking systems can be implemented in commercial parking lots, shopping centers, airports, and stadiums. They help optimize space utilization, improve customer experience, and provide real-time occupancy data to parking operators.
- **Parking Guidance Systems:** As our application demonstrates, these systems can help you choose as well as reserve your parking spaces in advance. This helps in the reduction of time to search and allows you to park at your pre-booked/reserved slot.
- **Parking Reservation Systems:** Smart parking systems can enable drivers to reserve parking spaces in advance through mobile apps or online platforms. This feature is particularly useful for high-demand areas, events, or time-limited parking. Reservation systems ensure parking availability and enhance convenience for users.
- **Parking Analytics and Planning:** The data collected by smart parking systems can be analyzed to gain insights into parking patterns, occupancy trends, and user behavior. This information is valuable for urban planners and parking authorities in making informed decisions regarding parking policies, infrastructure expansion, and transportation planning.

CHAPTER 9

CONCLUSION

Smart parking systems have several benefits and have the potential to significantly improve parking management, user experience, and urban mobility. These systems use innovative technology, including sensors, data analytics, and communication, to optimize space utilization, direct cars to available parking spots, and alleviate traffic congestion.

Although, it is critical to evaluate the possible drawbacks, which include a large initial investment, ongoing maintenance costs, privacy problems, and technological challenges. It is critical for the effective deployment and operation of smart parking systems to address these problems through careful design, frequent maintenance, data security measures, and user education.

The future scope of smart parking systems includes integration with connected and autonomous vehicles, AI and machine learning integration, and sustainability initiatives. Apart from that, smart parking systems are expected to contribute to data-driven decision-making, predictive analytics, and seamless integration with another smart city environment.

Overall, smart parking solutions can alter parking management, optimize resource utilization, improve user experiences, and help cities become smarter. Smart parking solutions are set to revolutionize the way we park our vehicles and traverse urban areas by combining technology and data.

CHAPTER 10

FUTURE SCOPE

The current system offers many services but still there is scope for advancement and improvement of the product. Following are some of the features that could be integrated into the system:

- **Integration with Connected and Autonomous Vehicles (CAVs):** As CAVs become more prevalent, smart parking systems can integrate with these vehicles to provide seamless parking experiences.
- **Artificial Intelligence and Machine Learning Integration:** These technologies can predict parking demand, dynamically adjust pricing, and offer personalized parking recommendations.
- **Data Sharing:** Future smart parking systems may facilitate data sharing among various stakeholders, such as urban planners, transportation agencies, and third-party service providers.
- **Predictive Analytics and Management:** This can help parking operators anticipate peak periods, adjust pricing strategies, and allocate parking spaces efficiently.
- **Integration with Smart City:** Smart parking systems will increasingly integrate with broader smart city initiatives, including traffic management systems, public transportation networks, and Internet of Things infrastructure. This will enable seamless coordination among various urban services, providing holistic mobility solutions.

BIBLIOGRAPHY

Aydin, Ilhan, Mehmet Karakose, and Ebru Karakose. "A navigation and reservation based smart parking platform using genetic optimization for smart cities." 2017 5th International Istanbul Smart Grid and Cities Congress and Fair (ICSG). IEEE, 2017.

Pham, Thanh Nam, et al. "A cloud-based smart-parking system based on Internet-of-Things technologies." IEEE access 3 (2015): 1581-1591.

Abdulkader, Omar, et al. "A novel and secure smart parking management system (SPMS) based on integration of WSN, RFID, and IoT." 2018 15th Learning and Technology Conference (L&T). IEEE, 2018.

IBM Watson IoT Platform [zz3850.internetofthings.ibmcloud.com](https://www.ibmcloud.com/zz3850.internetofthings.ibmcloud.com)

console.firebase.google.com

wokwi.com

ai2.appinventor.mit.edu

<https://github.com/MillsCS215AppInventorProj/chartmaker>

<https://www.hindawi.com/journals/jat/2020/5624586/tab1/>

<https://community.appinventor.mit.edu/t/failed-resolution-of-lcom-google-zxing-encodehinttype/29535/4>

<https://manage.wix.com/logo/maker/esh/?companyName=Park-It&tagLine=Parking%20made%20easy&selectedWebsiteId=1>

<https://openrouteservice.org/dev/#/home>

<https://community.appinventor.mit.edu/t/firebase-error-in-mit-app-inventor-app/40777/3>

<https://community.appinventor.mit.edu/t/firebase-is-not-working/34484>

<http://127.0.0.1:1880/#flow/34634e99.c4a09a>

APPENDIX

Source Code:

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#include <WiFi.h>//library for wifi

#include <PubSubClient.h>//library for MQTT

#include <FirebaseESP32.h>


LiquidCrystal_I2C lcd(0x27, 20, 4);


//void callback(char* subscribetopic, byte* payload, unsigned int payloadLength);


//-----credentials of IBM Accounts-----


#define ORG "zz3850"//IBM ORGANITION ID

#define DEVICE_TYPE "smartparking"//Device type mentioned in ibm watson IOT Platform

#define DEVICE_ID "112233"//Device ID mentioned in ibm watson IOT Platform

#define TOKEN "11223344" //Token


//----- Customise the above values -----

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";// Server Name

char publishTopic[] = "iot-2/evt/Data/fmt/json";// topic name and type of event perform and
format in which data to be send

char subscribetopic[] = "iot-2/cmd/command/fmt/String";// cmd REPRESENT command type
AND COMMAND IS TEST OF FORMAT STRING

char authMethod[] = "use-token-auth";// authentication method

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;//client id


//-----FIREBASE-----
```

```

#define FIREBASE_HOST "parkit-94d3f-default-rtdb.firebaseio.com"

#define FIREBASE_AUTH "TUlzvbaXNOAxyUhTE2ol40HLCVwIE5JmyN7FRC2l"

FirebaseData firebaseData;


WiFiClient wifiClient; // creating the instance for wificlient

PubSubClient client(server, 1883, wifiClient); //calling the predefined client id by passing
parameter like server id,portand wificredential


const int numofsensors=4;

const int trigpin3=2;

const int echopin3=15;

const int trigpin4=4;

const int echopin4=5;

const int trigpin1=14;

const int echopin1=27;

const int trigpin2=13;

const int echopin2=12;

const float thresholddist=50.0; //car rear sensor beeps when half a meter away from obstacle/wall

int parkingslots=4;

int s1=0;

int s2=0;

int s3=0;

int s4=0;

unsigned long slotOccupiedTime[numofsensors];

const int led1 = 26;

const int led2 = 25;

const int led3 = 18;

const int led4 = 19;

int freeSlots = 0;

String topic;

String payload;

```

```
unsigned long lastPublishTime = 0;

const unsigned long publishInterval = 10000;

void setup() {
    Serial.begin(9600);

    Wire.begin(SDA, SCL);

    lcd.begin(20, 4);
    lcd.backlight();
    wificonnect();
    mqttconnect();

    pinMode(trigpin1, OUTPUT);
    pinMode(trigpin2, OUTPUT);
    pinMode(trigpin3, OUTPUT);
    pinMode(trigpin4, OUTPUT);

    pinMode(led1,OUTPUT);
    pinMode(led2,OUTPUT);
    pinMode(led3,OUTPUT);
    pinMode(led4,OUTPUT);

    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);

    updateLCD();

    Firebase.begin(FIREBASE_HOST,FIREBASE_AUTH);
    Firebase.reconnectWiFi(true);
}
```

```

    Firebase.setReadTimeout(firebaseData,1000 *60);

    Firebase.setwriteSizeLimit(firebaseData,"tiny");


    Firebase.setInt(firebaseData,"/U1",0);
    Firebase.setInt(firebaseData,"/U2",0);
    Firebase.setInt(firebaseData,"/U3",0);
    Firebase.setInt(firebaseData,"/U4",0);
}

void loop() {

    unsigned long currentMillis = millis();
    if (currentMillis - lastPublishTime >= publishInterval) {
        publishFreeSlots();
        lastPublishTime = currentMillis;
    }


    // Sensor 1


    digitalWrite(trigpin1, LOW);
    digitalWrite(trigpin1, HIGH);
    delay(10);
    digitalWrite(trigpin1, LOW);
    long duration1 = pulseIn(echopin1, HIGH);
    float dist1 = (duration1 * 0.0343) / 2;

    if (dist1 < thresholddist && s1 == 0) {
        parkingslots--;
        s1 = 1;
        digitalWrite(led1, LOW);
    }
}

```

```

    updateLCD();

    slotOccupiedTime[0] = millis();

    Firebase.setInt(firebaseData, "/U1", 1);
}

else if (dist1 < thresholddist && s1 == 1){

    Firebase.setInt(firebaseData, "/U1", 1);
}

else if (dist1 >= thresholddist && s1 == 1) {

    parkingslots++;

    s1 = 0;

    digitalWrite(led1, HIGH);

    updateLCD();

    Firebase.setInt(firebaseData, "/U1", 0);

    unsigned long elapsed = millis() - slotOccupiedTime[0]; // Calculate elapsed time

    // Convert elapsed time to minutes or hours, depending on your requirement

    int minutes = elapsed / 60000; // 60000 milliseconds = 1 minute

    //int hours = minutes / 60;

    // Send occupied time to IBM Watson IoT Platform

    String topic = "iot-2/evt/Data/fmt/json";

    String payload = "{\"occupied_time_slot1\": " + String(minutes) + "}";

    //if (hours > 0) {

    //    payload += "\"" + String(hours) + " hours ";

    //}

    //payload += String(minutes % 60) + " minutes}";

    if (client.publish(topic.c_str(), (char*)payload.c_str())) {

        Serial.println("Published data for Slot 1");

        Serial.println(payload);

    }

    else {

        Serial.println("Failed to publish data for Slot 1");

    }
}

```

```

}

// Sensor 2
digitalWrite(trigpin2, LOW);
digitalWrite(trigpin2, HIGH);
delay(10);
digitalWrite(trigpin2, LOW);
long duration2 = pulseIn(echopin2, HIGH);
float dist2 = (duration2 * 0.0343) / 2;

if (dist2 < thresholddist && s2 == 0) {
    parkingslots--;
    s2 = 1;
    digitalWrite(led2, LOW);
    updateLCD();
    slotOccupiedTime[1] = millis();
    Firebase.setInt(firebaseData, "/U2", 1);
}
else if (dist2 < thresholddist && s2 == 1){
    Firebase.setInt(firebaseData, "/U2", 1);
}
else if (dist2 >= thresholddist && s2 == 1) {
    parkingslots++;
    s2 = 0;
    digitalWrite(led2, HIGH);
    updateLCD();
    Firebase.setInt(firebaseData, "/U2", 0);
    unsigned long elapsed = millis() - slotOccupiedTime[1]; // Calculate elapsed time
    // Convert elapsed time to minutes or hours, depending on your requirement
    int minutes = elapsed / 60000; // 60000 milliseconds = 1 minute
    //int hours = minutes / 60;

```

```

// Send occupied time to IBM Watson IoT Platform

String topic = "iot-2/evt/Data/fmt/json";

String payload = "{\"occupied_time_slot2\":\" + String(minutes) + "\"}";

//if (hours > 0) {

//  payload += "\"" + String(hours) + " hours ";

//}

//payload += String(minutes % 60) + " minutes}";

if (client.publish(topic.c_str(), (char*)payload.c_str())) {

  Serial.println("Published data for Slot 2");

  Serial.println(payload);

}

else {

  Serial.println("Failed to publish data for Slot 2");

}

}

// Sensor 3

digitalWrite(trigpin3, LOW);

digitalWrite(trigpin3, HIGH);

delay(10);

digitalWrite(trigpin3, LOW);

long duration3 = pulseIn(echopin3, HIGH);

float dist3 = (duration3 * 0.0343) / 2;

if (dist3 < thresholddist && s3 == 0) {

  parkingslots--;

  s3 = 1;

  digitalWrite(led3, LOW);

  updateLCD();

  slotOccupiedTime[2] = millis();

  Firebase.setInt(firebaseData,"/U3",1);

```



```

}

else if (dist3 < thresholddist && s3 == 1){

  Firebase.setInt(firebaseData,"/U3",1);

}

else if (dist3 >= thresholddist && s3 == 1) {

  parkingslots++;

  s3 = 0;

  digitalWrite(led3, HIGH);

  updateLCD();

  Firebase.setInt(firebaseData,"/U3",0);

  unsigned long elapsed = millis() - slotOccupiedTime[2]; // Calculate elapsed time

  // Convert elapsed time to minutes or hours, depending on your requirement

  int minutes = elapsed / 60000; // 60000 milliseconds = 1 minute

  //int hours = minutes / 60;

  // Send occupied time to IBM Watson IoT Platform

  String topic = "iot-2/evt/Data/fmt/json";

  String payload = "{\"occupied_time_slot3\": " + String(minutes) + "}";

  //if (hours > 0) {

  //  payload += "\"" + String(hours) + " hours ";

  //}

  //payload += String(minutes % 60) + " minutes}";

  if (client.publish(topic.c_str(), (char*)payload.c_str())) {

    Serial.println("Published data for Slot 3");

    Serial.println(payload);

  }

  else {

    Serial.println("Failed to publish data for Slot 3");

  }

}

// Sensor 4

```

```

digitalWrite(trigpin4, LOW);
digitalWrite(trigpin4, HIGH);
delay(10);
digitalWrite(trigpin4, LOW);
long duration4 = pulseIn(echopin4, HIGH);
float dist4 = (duration4 * 0.0343) / 2;

if (dist4 < thresholddist && s4 == 0) {
    parkingslots--;
    s4 = 1;
    digitalWrite(led4, LOW);
    updateLCD();
    slotOccupiedTime[3] = millis();
    Firebase.setInt(firebaseData, "/U4", 1);
}
else if (dist4 < thresholddist && s4 == 1){
    Firebase.setInt(firebaseData, "/U4", 1);
}
else if (dist4 >= thresholddist && s4 == 1) {
    parkingslots++;
    s4 = 0;
    digitalWrite(led4, HIGH);
    updateLCD();
    Firebase.setInt(firebaseData, "/U4", 0);
    unsigned long elapsed = millis() - slotOccupiedTime[3]; // Calculate elapsed time
    // Convert elapsed time to minutes or hours, depending on your requirement
    int minutes = elapsed / 60000; // 60000 milliseconds = 1 minute
    //int hours = minutes / 60;
    // Send occupied time to IBM Watson IoT Platform
    String topic = "iot-2/evt/Data/fmt/json";
    String payload = "{\"occupied_time_slot4\":\"" + String(minutes) + "\"}";
}

```

```

        //if (hours > 0) {
        // payload += "\"" + String(hours) + " hours ";
        //}

        //payload += String(minutes % 60) + " minutes}";
        if (client.publish(topic.c_str(), (char*)payload.c_str())) {
            Serial.println("Published data for Slot 4");
            Serial.println(payload);
        }
        else {
            Serial.println("Failed to publish data for Slot 4");
        };
    }
}

void mqttconnect() {
    if (!client.connected()) {
        Serial.print("Reconnecting client to ");
        Serial.println(server);
        while (!client.connect(clientId, authMethod, token)) {
            Serial.print(".");
            delay(500);
        }

        initManagedDevice();
        Serial.println();
    }
}

void wificonnect() //function defination for wificonnect
{
    Serial.println();
}

```

```

Serial.print("Connecting to ");

WiFi.begin("Wokwi-GUEST", "", 6); //passing the wifi credentials to establish the connection
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void initManagedDevice() {
    if (client.subscribe(subscribetopic)) {
        Serial.println((subscribetopic));
        Serial.println("subscribe to cmd OK");
    } else {
        Serial.println("subscribe to cmd FAILED");
    }
}

void publishFreeSlots() {
    topic = "iot-2/evt/FreeSlots/fmt/json";
    payload = "{\"free_slots\": " + String(freeSlots) + "}";

    if (client.publish(topic.c_str(), payload.c_str())) {
        Serial.println("Free slots published successfully.");
    } else {
        Serial.println("Failed to publish free slots data.");
    }
}

```

```
}
```

```
//LCD UPDATE FUNC
```

```
void updateLCD() {  
    freeSlots = parkingslots;  
    lcd.clear();  
    lcd.setCursor(2, 0);  
    lcd.print("Basement Parking");  
    lcd.setCursor(1, 1);  
    lcd.print("Free Slots:");  
    lcd.setCursor(12, 1);  
    lcd.print(freeSlots);  
    if(s1&& s2==1){  
        lcd.setCursor(14,1);  
        lcd.print("-->");  
    }  
    else if (s3&& s4==1){  
        lcd.setCursor(14,1);  
        lcd.print("<--");  
    }  
    lcd.setCursor(0, 2);  
    lcd.print("S1:");  
    lcd.setCursor(3, 2);  
    if (s1 == 0) {  
        lcd.print("Empty");  
    } else {  
        lcd.print("Full");  
    }  
    lcd.setCursor(0, 3);  
    lcd.print("S2:");  
    lcd.setCursor(3, 3);
```

```
    if (s2 == 0) {  
        lcd.print("Empty");  
    } else {  
        lcd.print("Full");  
    }  
    lcd.setCursor(10, 2);  
    lcd.print("S3:");  
    lcd.setCursor(13, 2);  
    if (s3 == 0) {  
        lcd.print("Empty");  
    } else {  
        lcd.print("Full");  
    }  
    lcd.setCursor(10, 3);  
    lcd.print("S4:");  
    lcd.setCursor(13, 3);  
    if (s4 == 0) {  
        lcd.print("Empty");  
    } else {  
        lcd.print("Full");  
    }  
}
```