"IoT Based Smart Parking"

A Major Project Dissertation

Submitted in partial fulfillment of the requirement for the award of Degree of
Bachelor of Technology in Electronics and Communication Engineering

Submitted to

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, (M.P.)

Submitted By:

Akash Bagwan	**Enrollment No – 0808EC201005**
Jay Barode	**Enrollment No – 0808EC201012**
Surendra Prajapat	**Enrollment No – 0808EC201025**

Under the Supervision of:

Prof. Rupesh Dubey
Head Of Department
(Department of EC)

Department of Electronics & Communication Engineering
IPS Academy, Institute of Engineering and Science Indore [MP]
(A UGC Autonomous Institute, Affiliated to RGPV)

Session: 2023-24

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

IPS ACADEMY, INSTITUTE OF ENGINEERING AND SCIENCE INDORE (M.P.)

(A UGC Autonomous Institute, Affiliated to RGPV)

CERTIFICATE

This is to certify that the work embodies in this Major Project Dissertation entitled **"IoT Based Smart Parking"** being submitted by "**Mr. Akash Bagwan" (Roll No.:0808EC201005),** "**Mr. Jay Barode"(Roll No.:0808EC201012),** "**Mr. Surendra Prajapat" (Roll No.: 0808EC201025)** for partial fulfillment of the requirement for the award of "**Bachelor of Technology** in **Electronics and Communication Engineering"** to Rajiv Gandhi Proudyogiki Vishwavidyalaya, (M.P.), during the academic year 2023-24 is a record of bonafide piece of work, carried out by them under our supervision and guidance in the **"Department of Electronics and Communication Engg.", IPS Academy, IES, Indore (M.P.).**

APPROVED & SUPERVISED BY:

Guide - Prof. Rupesh Dubey
Head of Department
(Department of EC)

FORWARDED BY :

(Prof. Rupesh Dubey)
Head of Department
(Department of EC)

(Dr. Archana Keerti Chowdhary)
Principal

DECLARATION

We **Akash Bagwan (0808EC201005), Jay Barode (0808EC201012), Surendra Prajapat (0808EC201025)**, the students of **Bachelor of Technology** in Electronics and Communication Engineering, **session: 2023-24, IPS Academy, Institute of Engineering & Science Indore (M.P.)** hereby declare that the work presented in this Major Project dissertation entitled **"IoT Based Smart Parking"** is the outcome of our own work, is bonafide and correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics.

(Akash Bagwan)
Enrollment No.: 0808EC201005

(Jay Barode)
Enrollment No.: 0808EC201012

(Surendra Prajapat)
Enrollment No.: 0808EC201025

Date:

CONTENTS

Acknowledgement

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got this all the completion of our project work. Whatever we have done is only due to such guidance and assistance and we would not forget to thank them.

We owe our profound to our project guide Mr. Rupesh Dubey Head Of the Department of Electronics and Communication who took keen interest on our project work and guided, us all along, till the completion of our project work by providing all the necessary information, constant encouragement, and sympathetic attitude. The completion of this dissertation would not have been possible without such guidance and support.

We are thankful and fortunate enough to get constant encouragement and guidance from all teaching staffs of department of Electronics and Communication which helped us in successfully completing our project work.

Abstract

This project is based on the parking management operation using a system based on microcontroller and internet . This system uses the IR sensors to detect the presence of vehicle in parking lot, and accordingly allow the vehicle to enter the parking lot. The IR sensors sends the required data to the ESP-32 which then sends the signal to servo motor to open or close the entry gate accordingly. A typical microcontroller circuit with IR sensor and servo motors and also with built-in Wi-Fi will efficiently and perfectly do such operations.

LIST OF FIGURES

Chapter 1

Introduction

CHAPTER 1

Introduction

1.1 Introduction :-

An IoT Based Smart car parking system is a type of intelligent transportation system (ITS) that uses technology to optimize parking availability and efficiency. It typically consists of a network of sensors, cameras, database and software that can detect and track vehicles in a parking lot. This data is used to provide real-time information to drivers about available parking spaces, as well as to guide them to the nearest open spot on their device.

The goal of Smart Parking System(SPS) is to reduce congestion, enhance safety, and provide convenience to drivers.

Some of the key features of SPSs:

1. Real-Time Parking Availability:

SPSs utilize sensors and cameras to monitor parking spaces in real-time, providing drivers with accurate information about vacant spots. This eliminates the need for drivers to circle parking lots endlessly, reducing congestion and frustration.

2. Parking Guidance Systems:

SPSs often incorporate parking guidance systems that guide drivers directly to available parking spots. These systems can use digital signage, mobile apps, or in-vehicle navigation systems to provide directions.

3. Mobile App Integration:

Mobile apps are becoming an integral part of SPSs, allowing drivers to reserve parking spots, pay for parking, and receive real-time updates on parking availability. These apps also provide users with a convenient way to manage their parking history and preferences.

4. Contactless Payments:

SPSs often integrate with mobile payment platforms, enabling contactless and secure parking fee transactions. This eliminates the need for drivers to carry cash or wait in line at parking meters.

5. Security and Monitoring:

SPSs often incorporate surveillance cameras and monitoring capabilities to enhance security in parking facilities. This can help deter theft and vandalism and provide real-time footage in case of incidents.

6. Reservation and Pre-Booking: Advanced SPSs allow drivers to reserve parking spots in advance, ensuring they have a designated spot upon arrival. This is particularly useful for busy parking lots or events.

1.2 Literature Review :-

The advent of IoT-based car parking systems marks a significant advancement in urban mobility management, promising to revolutionize the way we navigate city streets. These systems leverage interconnected sensors, data analytics, and mobile applications to address the perennial challenge of finding parking spaces in congested urban areas. By detecting the availability of parking spots in real-time and providing users with up-to-date information via mobile apps or digital signage, IoT-based solutions offer unparalleled convenience and efficiency. Moreover, the data collected by these systems can inform urban planners in optimizing traffic flow, reducing congestion, and mitigating environmental impact. Despite their promise, challenges such as infrastructure costs, cybersecurity concerns, and interoperability issues remain to be addressed. Nevertheless, case studies from cities like Barcelona and Singapore demonstrate the transformative potential of IoT-based car parking systems in creating smarter, more sustainable urban environments. As cities continue to grapple with the complexities of urbanization, these systems offer a beacon of hope for a future where parking is no longer a source of frustration but an integrated component of seamless urban living.

Chapter 2
Project Objective

Chapter 2
Project Objective

2.1 Project Objective :-

The objective of our final year project is to develop an IoT-based smart car parking system that efficiently manages parking spaces and provides real-time information to users regarding parking slot availability. The system aims to enhance the overall parking experience for both drivers and parking management authorities by leveraging IoT technology and data analytics. The key objectives are as follows:

1. **Real-time Parking Slot Availability:** Develop a system that can accurately monitor and update the status of parking slots in real-time. Utilize IoT sensors and devices installed in each parking space to detect the presence of vehicles and transmit this information to a central server.

2. **User-Friendly Interface:** Create user-friendly interfaces for both drivers and parking management authorities. Develop a mobile application for drivers that displays real-time information about available parking slots, location guidance,

3. **Efficient Parking Management:** Implement intelligent algorithms to optimize parking space allocation and improve the utilization of parking resources.

4. **Integration with Entry Gate Display:** Integrate the system with display screens at the entry gate of the parking facility to provide real-time parking availability updates to incoming vehicles. Display clear and intuitive information to guide drivers to available parking spaces, reducing congestion at entry points and improving traffic flow within the parking facility.

5. **Energy Efficiency:** Optimize the energy consumption of IoT devices and sensors to prolong battery life and reduce maintenance requirements. Implement power-saving features such as sleep modes, intelligent scheduling, and low-power communication protocols to minimize energy usage without compromising system performance.

8. **Documentation and Knowledge Sharing:** Document the design, implementation, and testing processes comprehensively to facilitate knowledge sharing and future development efforts. Provide user manuals, technical specifications, and troubleshooting guides to assist stakeholders in understanding and utilizing the system effectively.

By achieving these objectives, our project aims to demonstrate the feasibility and benefits of IoT-based smart car parking systems in improving parking management efficiency, enhancing user experience, and addressing the challenges associated with urban parking congestion.

CHAPTER 3

Block Diagram and Its Description

CHAPTER 3

Block Diagram and Its Description

3.1 Block Diagram :-

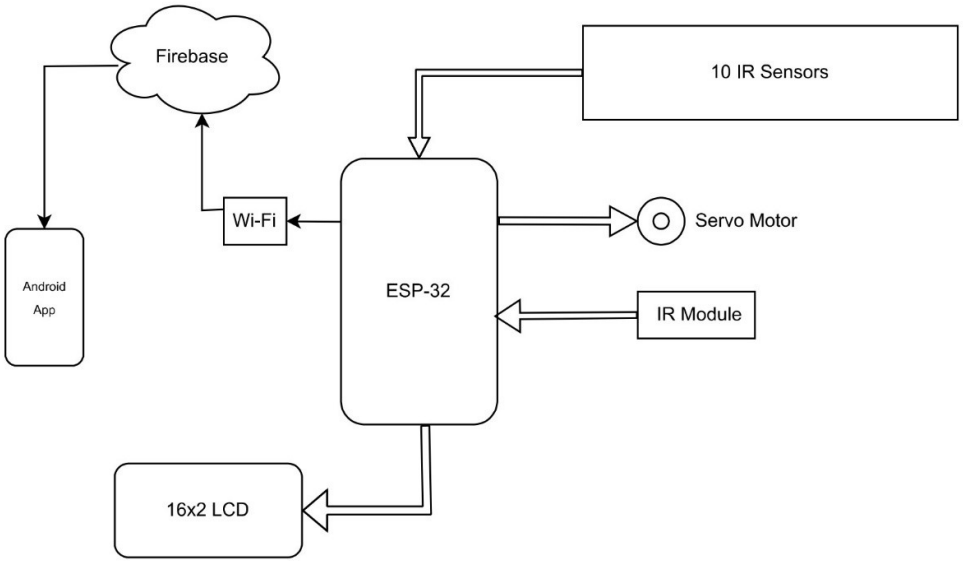


Fig. 3.1 Block Diagram

3.2 Description of Block Diagram :-

The block diagram of IoT Based Smart Parking using ESP-32, IR sensors, LCD Display and a servo motor shows how the different components work together to detect the presence of vehicles in parking spaces and control the movement of a barrier to allow or deny access and update the data on cloud database.

1. The IR sensors are placed at the entrance and exit of the parking space.
2. When a vehicle enters the parking space, the IR sensor at the entrance detects its presence and sends a signal to the ESP-32.
3. ESP-32 then checks if there is any available parking space. If there is, it sends a signal to the servo motor to open the barrier and allow the vehicle to enter and decrease the count of available slots by one and display it on LCD.
4. Once the vehicle has entered the parking space and parked, the IR sensor at the entrance detects its presence and sends a signal to the ESP-32.

5. ESP-32 then updates the record of available parking spaces and send it to the database over the internet.
6. When the vehicle exits the parking space, the IR sensor at the exit detects its presence and sends a signal to the ESP-32.
7. ESP-32 then sends a signal to the servo motor to open the barrier and increase the count of available slots by one and update the database also.
8. This process is repeated for all vehicles that enter and exit the parking space.

Explanation of how each component works:

- **IR sensors:** IR sensors are used to detect the presence of vehicles in parking spaces. They work by emitting an infrared beam and detecting when the beam is reflected back. When a vehicle is present, the IR sensor will detect the reflected beam and send a signal to the Arduino Uno.
- **ESP-32:** ESP-32 is a development board with builtin wi-fi and bluetooth that is used to process the signals from the IR sensors and control the servo motor and LCD display and send data to database over the internet . ESP-32 is programmed with a sketch that tells it how to behave.
- **Servo motor:** The servo motor is used to control the movement of the barrier. When the Arduino Uno sends a signal to the servo motor, the servo motor will rotate to the specified position. This will open or close the barrier.
- **LCD display:** The LCD display is used to show the status of the parking system, such as the number of available parking spaces. ESP-32 can send data to the LCD display to update the information being displayed.

CHAPTER 4
Hardware Modeling

CHAPTER 4

Hardware Modeling

4.1 Components Used :-

- ESP-32
- 16x2 LCD
- I2C Module
- IR Sensor
- Servo SG90
- RGB LED's
- BC337 Transistor
- Resistors
- Toggle Switch
- Zero PCB
- Power Adaptor(5v 2A)

4.2 Description of Components :-

4.2.1 ESP-32 :-

The ESP-32 is a powerful and versatile microcontroller developed by Espressif Systems, a semiconductor company based in Shanghai, China. It is widely recognized for its robust features, low power consumption, and integration of Wi-Fi and Bluetooth connectivity, making it a popular choice for Internet of Things (IoT) applications.

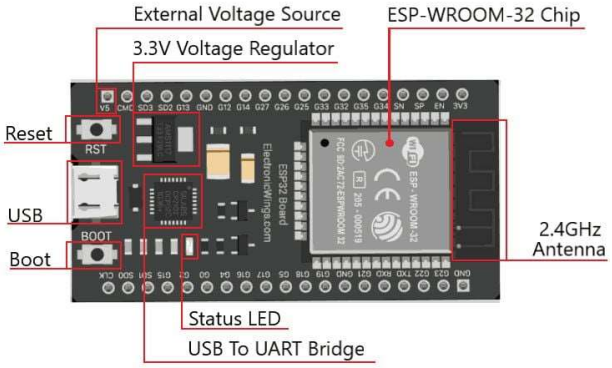


Fig. 4.1 ESP-32

Working of ESP-32:

- The ESP-32 microcontroller operates based on the Xtensa LX6 dual-core processor architecture.
- It integrates Wi-Fi and Bluetooth connectivity, allowing it to connect to wireless networks and communicate with other devices.
- The ESP-32 can be programmed using the Arduino IDE, MicroPython, or the Espressif IDF (IoT Development Framework).

- It features a rich set of peripherals, including GPIO (General Purpose Input/Output) pins, SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), UART (Universal Asynchronous Receiver-Transmitter), ADC (Analog-to-Digital Converter), DAC (Digital-to-Analog Converter), and more.
- The ESP-32 supports deep sleep modes, enabling low-power operation for battery-powered IoT devices.

Pin Description of ESP-32:

- **VIN (Input Voltage):** This pin is used to supply power to the ESP-32 module. It typically accepts a voltage range of 3.3V to 5V.
- **3V3 (3.3V Output):** This pin provides a regulated 3.3V output voltage. It can be used to power external components that require 3.3V.

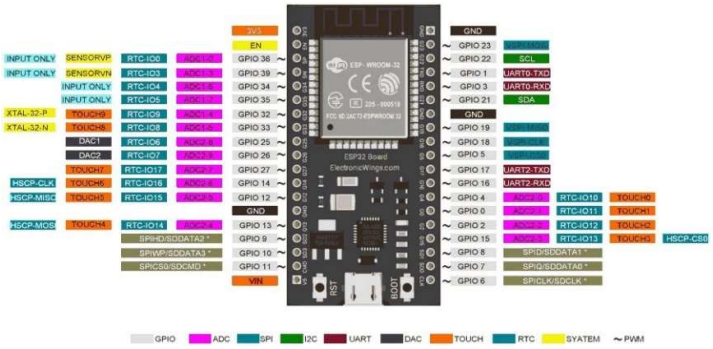


Fig. 4.2 ESP-32 Pinout

- **GND (Ground):** These pins are connected to the ground of the power supply and are used as the reference voltage for the circuit.
- **GPIO (General Purpose Input/Output):** These pins can be configured as either digital input or output. They are commonly used for interfacing with external sensors, actuators, and peripherals.
- **ADC (Analog-to-Digital Converter):** These pins are used to read analog voltage levels from sensors. They convert analog signals into digital values that can be processed by the ESP-32.
- **DAC (Digital-to-Analog Converter):** These pins are used to output analog voltage levels. They convert digital values into analog signals, which can be used for tasks such as audio output or controlling analog devices.
- **UART (Universal Asynchronous Receiver-Transmitter):** These pins are used for serial communication with other devices. They allow the ESP-32 to transmit and receive data asynchronously.
- **SPI (Serial Peripheral Interface):** These pins are used for high-speed serial communication with peripherals such as sensors, displays, and SD cards. The SPI interface typically consists of four pins: MOSI (Master Out Slave In), MISO (Master in Slave Out), SCK (Serial Clock), and SS (Slave Select).
- **I2C (Inter-Integrated Circuit):** These pins are used for two-wire serial communication with devices such as sensors, EEPROMs, and real-time clocks. The I2C interface typically consists of two pins: SDA (Serial Data) and SCL (Serial Clock).
- **Boot Mode Pins:** These pins determine the boot mode of the ESP-32 during startup. They control whether the ESP-32 boots into normal mode, bootloader mode, or firmware update mode.

- **Reset Pin:** This pin is used to reset the ESP-32 microcontroller. When pulled low, it resets the device, restarting the program execution.
- **EN (Enable) Pin:** This pin is used to enable or disable the ESP-32's power supply. When pulled high, it activates the ESP-32 and allows it to operate.

Parameters of ESP-32:

- Operating Voltage: Typically operates at 3.3V, but can tolerate up to 3.6V.
- Clock Speed: Dual-core processor running at up to 240MHz.
- Wi-Fi: IEEE 802.11 b/g/n/e/i support with integrated TCP/IP protocol stack.
- Bluetooth: Bluetooth v4.2 BR/EDR and BLE (Bluetooth Low Energy) support.
- Memory: Typically equipped with 520KB SRAM and up to 16MB flash memory.
- GPIO Pins: Typically features around 36 GPIO pins, which can be used for various digital and analog input/output purposes.
- ADC/DAC: Integrated ADC with up to 18 channels and DAC with up to 2 channels.
- Operating Temperature: The ESP-32 is designed to operate reliably within a certain temperature range, typically around -40°C to 125°C.

Specification of ESP-32:

- Processor: Dual-core Tensilica LX6 MCU.
- Wi-Fi: IEEE 802.11 b/g/n/e/i support.
- Bluetooth: Bluetooth v4.2 BR/EDR and BLE support.
- RAM: Typically, 520KB SRAM.
- Flash Memory: Up to 16MB.
- Operating Voltage: 3.3V.
- Clock Speed: Up to 240MHz.
- GPIO Pins: Around 36 GPIO pins.
- ADC/DAC: Integrated ADC with up to 18 channels and DAC with up to 2 channels.
- Dimensions: Various modules and development boards are available, each with its own dimensions.

Application of ESP-32:

- IoT Devices: ESP-32 is widely used in IoT applications such as smart home devices, environmental monitoring systems, and wearable technology.
- Wireless Sensors: It can be used to develop wireless sensor nodes for data collection and transmission.
- Home Automation: ESP-32-based devices can control and monitor home appliances, lighting systems, and security cameras.
- Industrial Automation: ESP-32 can be used in industrial automation for monitoring and controlling machinery and equipment.

4.2.2 16x2 LCD Display :-

A 16×2 LCD display is a liquid crystal display that can show 16 characters in each of its two rows, providing a total of 32 characters of information and it is commonly used to Display the Slot availability in the parking based on different sensor and other things that use in Project.

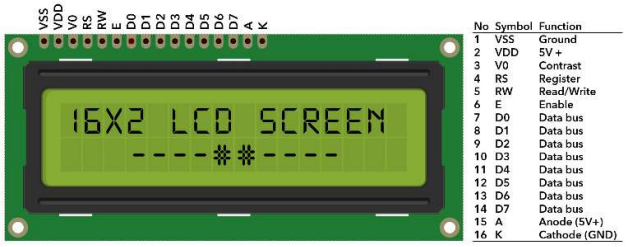


Fig 4. 3 16x2 LCD Display

Specifications of LCD 16x2:

- Character size: Typically, 5x8 pixels.
- Display type: Reflective or translative, with or without backlight.
- Viewing area: Approximately 64mm x 16mm.
- Interface: Parallel (e.g., 4-bit or 8-bit) or serial (e.g., I2C or SPI).

Pin Description of LCD 16x2:

- **GND (Ground):** This pin is connected to the ground of the power supply and serves as the reference voltage for the circuit.
- **VCC (+5V):** This pin is connected to the positive terminal of the power supply and provides the operating voltage for the LCD module.
- **VEE (Contrast Adjustment):** This pin is used to adjust the contrast of the display. By varying the voltage applied to this pin, the contrast of characters on the LCD can be adjusted for optimal visibility.
- **RS (Register Select):** This pin is used to select between data and command modes. When RS is high (logic 1), the data sent to the LCD is interpreted as a character to be displayed. When RS is low (logic 0), the data is treated as a command to be executed by the LCD controller.
- **RW (Read/Write):** This pin is used to control the direction of data transfer between the microcontroller and the LCD module. When RW is high (logic 1), the LCD is in read mode, allowing the microcontroller to read data from the LCD. When RW is low (logic 0), the LCD is in write mode, allowing the microcontroller to write data to the LCD.
- **E (Enable):** This pin is used to enable the LCD module. A high-to-low transition on this pin triggers the LCD controller to execute the data or command present on the data bus.

- **D0-D7 (Data Lines):** These pins are used to transfer data between the microcontroller and the LCD module. In 4-bit mode, only D4-D7 are used, while in 8-bit mode, all eight data lines (D0-D7) are utilized for communication.
- **LED+ (Backlight Anode):** This pin is connected to the positive terminal of the backlight LED(s) and is used to supply power to illuminate the LCD backlight.
- **LED- (Backlight Cathode):** This pin is connected to the negative terminal of the backlight LED(s) and is used as the ground for the backlight circuit.
- **NC (Not Connected):** These pins are not connected to anything and are typically left unused. They may be present for compatibility with different LCD modules or for future expansion.

4.2.3 I2C Module of LCD :-

An I2C module is a small electronic device that allows you to connect multiple devices to a single microcontroller or microprocessor using only two wires. This is in contrast to traditional parallel communication methods, which require a separate wire for each data bit.

The I2C protocol is a two-wire serial bus that was developed by Philips Semiconductor in the 1980s. It is a widely used protocol that is supported by a wide variety of microcontrollers and microprocessors.

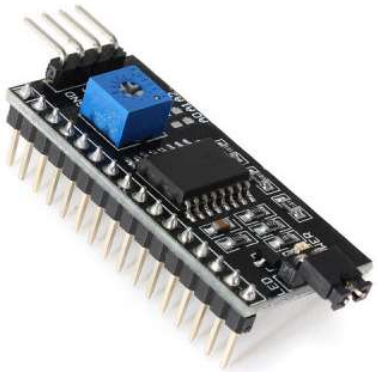
The I2C bus consists of two wires: a serial data (SDA) line and a serial clock (SCL) line. The SDA line is used to transmit data between the master and slave devices on the bus. The SCL line is used to synchronize the clock signals of the master and slave devices.

The I2C protocol uses a master-slave architecture. The master device is responsible for initiating and controlling communication on the bus. The slave devices are responsible for responding to commands from the master device.

The I2C protocol supports a variety of data transfer modes, including standard mode, fast mode, and high-speed mode. The standard mode supports data transfer rates of up to 100 kbps. The fast mode supports data transfer rates of up to 400 kbps. The high-speed mode supports data transfer rates of up to 3.4 Mbps.

I2C modules are commonly used to connect a variety of devices to a microcontroller or microprocessor, including:

- LCD displays
- Keyboards
- Sensors
- Actuators



- Memory devices

I2C modules are a convenient and easy-to-use way to connect multiple devices to a microcontroller or microprocessor. They are a popular choice for a wide variety of applications, including hobby electronics, robotics, and industrial automation.

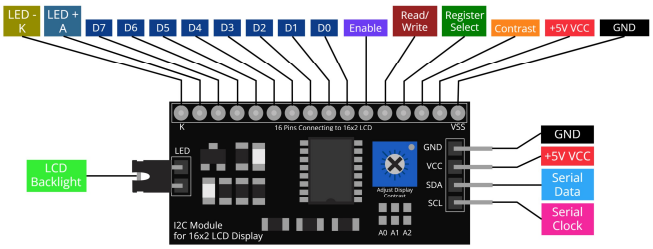


Fig 4.4 I2C Module

4.2.4 Infrared Sensor :-

An infrared sensor (IR sensor) is an electronic device that measures and detects infrared radiation in its surrounding environment. Infrared radiation is a type of electromagnetic radiation that is invisible to the human eye, but it can be detected as heat. IR sensors are used in a wide variety of applications, including motion detection, temperature measurement, and communication.

There are two main types of IR sensors:

- Active IR sensors: Active IR sensors emit their own infrared radiation and then detect the reflection of that radiation off of objects in their environment. This type of sensor is often used in motion detectors, as it can detect objects even if they are not emitting their own infrared radiation.
- Passive IR sensors: Passive IR sensors do not emit their own infrared radiation. Instead, they detect the infrared radiation that is emitted by objects in their environment. This type of sensor is often used in temperature measurement, as it can measure the temperature of an object without having to touch it.

IR sensors are a versatile and widely used type of sensor. They are relatively inexpensive, easy to use, and reliable. They are also very sensitive to infrared radiation, which makes them well-suited for a variety of applications.

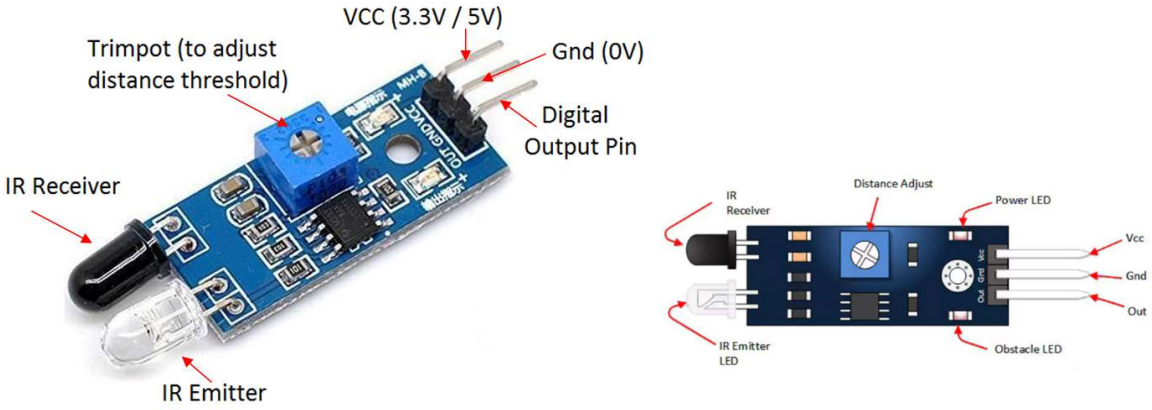


Fig 4.5 Infrared Sensor

4.2.5 Mini Servo Motor SG-90 :-

The SG-90 is a small, low-cost, high-torque servo motor commonly used in remote-controlled (RC) hobby applications, such as robotics, model making, and RC vehicles. It is a popular choice for beginners due to its ease of use and affordability.

The SG-90 has a 180-degree range of motion and can operate at speeds of up to 0.12 seconds per 60 degrees. It has a torque of 1.2 kg-cm at 4.8V and 1.6 kg-cm at 6.6V. It is powered by a 3-7.2V DC power supply and has a JR-style servo connector.

The SG-90 is a versatile servo motor that can be used for a variety of applications. It is commonly used in RC vehicles, such as airplanes, boats, and cars. It is also used in robotics, such as robotic arms and manipulators. Additionally, it is used in other hobbyist projects, such as animatronics and model trains.

Here are some of the specifications of the SG-90 servo motor:

- Operating voltage: 3-7.2V DC
- Torque: 1.2 kg-cm at 4.8V, 1.6 kg-cm at 6.6V
- Speed: 0.12 seconds per 60 degrees at 4.8V, 0.08 seconds per 60 degrees at 6V
- Operating temperature range: -30 to +60 degrees Celsius
- Dimensions: 22.5 x 11.5 x 23.5 mm
- Weight: 9 grams

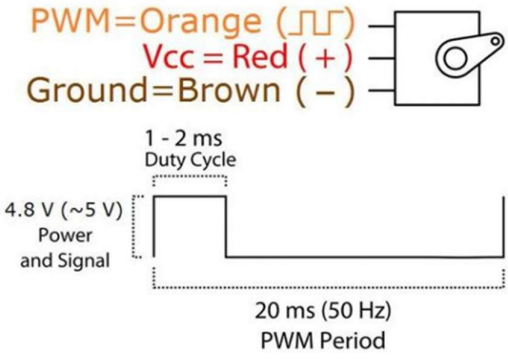


Fig 4.6 Servo Motor SG 90

4.2.6 RGB LED :-

An RGB LED is basically an LED package that can produce almost any color. It can be used in different applications such as outdoor decoration lighting, stage lighting designs, home decoration lighting, LED matrix display, and more.

RGB LEDs have three internal LEDs (Red, Green, and Blue) that can be combined to produce almost any color output. In order to produce different kinds of colors, we need

to set the intensity of each internal LED and combine the three color outputs. In this tutorial, we are going to use PWM to adjust the intensity of the red, green, and blue LEDs individually and the trick here is that our eyes will see the combination of the colors, instead of the individual colors because the LEDs are very close to each other inside.

Types and Structure:

As mentioned earlier, RGB LEDs have three LEDs inside them and usually, these three internal LEDs share either a common anode or a common cathode especially in a through-hole package. So basically, we can categorize RGB LEDs as either common anode or common cathode type just like in seven segment displays.

Common Anode RGB LED

In a common anode RGB LED, the anode of the internal LEDs are all connected to the external anode lead. To control each color, you need to apply a LOW signal or ground to the red, green, and blue leads and connect the anode lead to the positive terminal of the power supply.

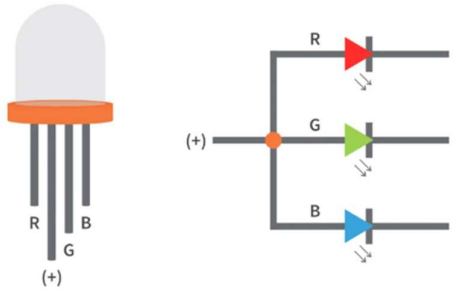


Fig. 4.7 RGB Led

4.2.7 BC337 Transistor :-

The BC337 is an NPN bipolar junction transistor (BJT) commonly used in electronic circuits. It is designed for general-purpose amplification and switching applications.

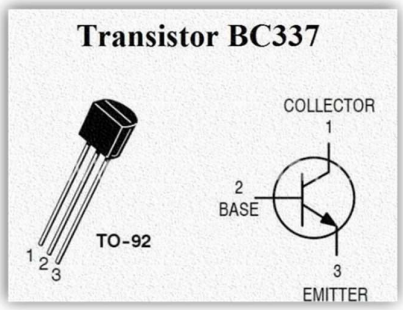


Fig. 4.8 BC337 Transistor

Working of BC337 Transistor:

- In an NPN transistor like the BC337, current flows from the emitter to the collector when a small current is applied to the base.
- By controlling the current at the base, the larger current flowing between the collector and emitter can be regulated.
- The transistor acts as an amplifier or a switch depending on how it's biased and the circuit configuration.

Parameters of BC337 Transistor:

- Maximum Collector-Emitter Voltage (Vce): Typically, around 45V.
- Maximum Collector Current (Ic): Typically, around 800mA.
- DC Current Gain (hfe): Typically ranges from 100 to 630.
- Transition Frequency (ft): Typically, around 100MHz.

- Power Dissipation (Pd): Depends on the package, typically ranges from 625mW to 1.25W.

- Specifications of BC337 Transistor:
- Package: Available in various packages including TO-92, SOT-23, and SOT-54.
- Pinout: Typically consists of three leads for the collector, base, and emitter connections.
- Operating Temperature: Typically ranges from -55°C to +150°C.

4.2.8 Resistors :-

Resistor is defined as a passive electrical component with two terminals that are used for either limiting or regulating the flow of electric current in electrical circuits.

The main purpose of resistor is to reduce the current flow and to lower the voltage in any particular portion of the circuit. It is made of copper wires which are coiled around a ceramic rod and the outer part of the resistor is coated with an insulating paint.

Here we required resistors of value 1k, 220ohm and 50 ohm.

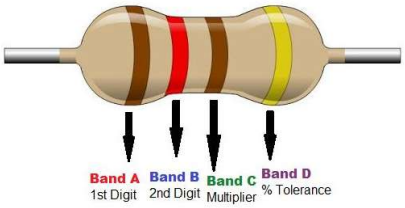


Fig. 4.9 Resistor

4.2.9 Toggle Switch :-

Toggle Switches have an operating lever that can be pushed up and down or left and right to switch an electrical circuit.

A “toggle” is a small wooden rod that is used as a clothing fastener in the place of buttons. The term “toggle switch” was created because of the resemblance of the switch’s lever to that of the toggle used in clothing.

Fig. 4.10 Toggle Switch

4.2.10 Zero PCB :-

Zero PCB is basically a general-purpose printed circuit board (PCB), also known as perfboard or DOT PCB. It is a thin rigid copper sheet with holes pre-drilled at standard intervals across a grid with 2.54mm (0.1-inch) spacing between holes. Each hole is encircled by a round or square copper pad so that component lead can be inserted into the hole and soldered around the pad without short-circuiting the nearby pads and other leads. For connecting the lead of component with another lead, solder these together or join these using a suitable conducting wire.

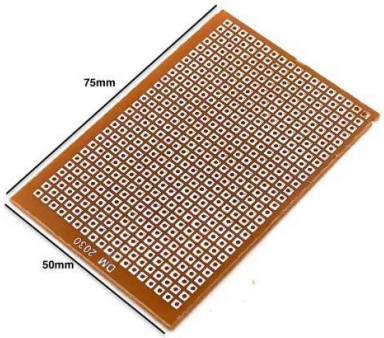


Fig. 4.11 Zero PCB

4.2.11 Power Adapter (5V, 2A) :-

A 5V, 2A power adapter is a device that converts AC (alternating current) power from a wall outlet to DC (direct current) power that can be used to power electronic devices. The "5V" part of the name refers to the voltage of the DC power that the adapter outputs, and the "2A" part refers to the maximum current that the adapter can supply.

5V, 2A power adapters are commonly used to power a wide variety of electronic devices, including smartphones, tablets, laptops, cameras, and game consoles. They are also used to power some electronic components, such as LED strips and Arduino boards.

Fig. 4.12 Power Adapter

CHAPTER 5

Circuit Diagram and Pin Connections

CHAPTER 5

Circuit Diagram and Pin Connections

5.1 Circuit Diagram :-

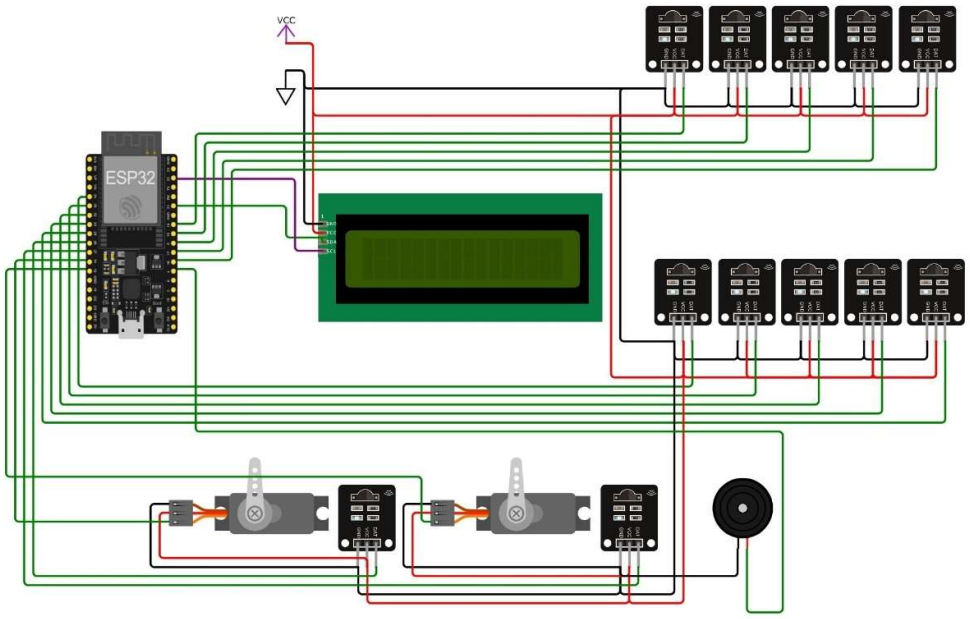


Fig. 5.1 Circuit Diagram

5.2 Pin Connections :-

- ESP-32 to IR Sensor :-

- ESP-32 Pin 34 – Out pin of IR 1
- ESP-32 Pin 35 – Out pin of IR 2
- ESP-32 Pin 32 – Out pin of IR 3
- ESP-32 Pin 33 – Out pin of IR 4
- ESP-32 Pin 25 – Out pin of IR 5

- ESP-32 Pin 19 – Out pin of IR 6
- ESP-32 Pin 18 – Out pin of IR 7
- ESP-32 Pin 05 – Out pin of IR 8
- ESP-32 Pin 17 – Out pin of IR 9
- ESP-32 Pin 16 – Out pin of IR 10

- ESP-32 Pin 26 – Out pin of IR Entry
- ESP-32 Pin 27 – Out pin of IR Exit

- **ESP-32 to Servo Motor :-**

- ESP-32 Pin 14 – Signal pin of IR Servo_Entry
- ESP-32 Pin 12 –Signal pin of IR Servo_Exit

- **ESP-32 to Buzzer :-**

- ESP-32 Pin 04 –Pin of Buzzer

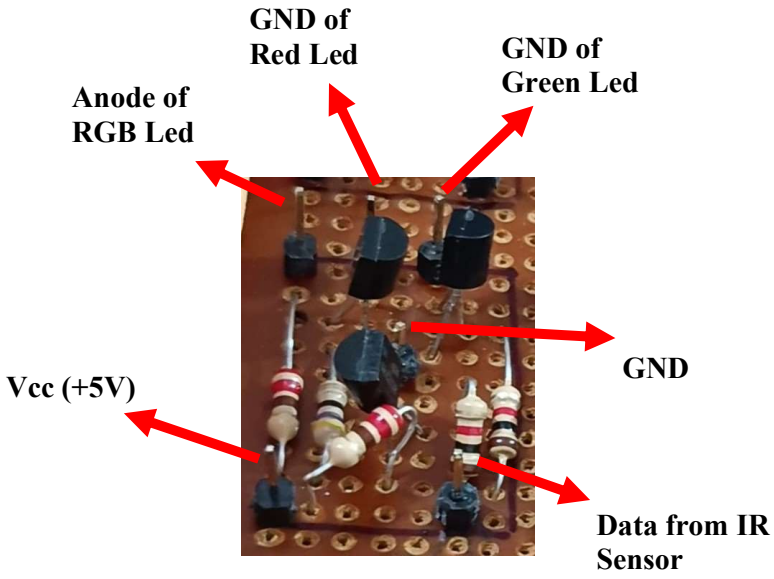
CHAPTER 6
PCB Layout

CHAPTER 6

PCB Layout

6.1 PCB Layout :-

Fig. 6.1 PCB Layout



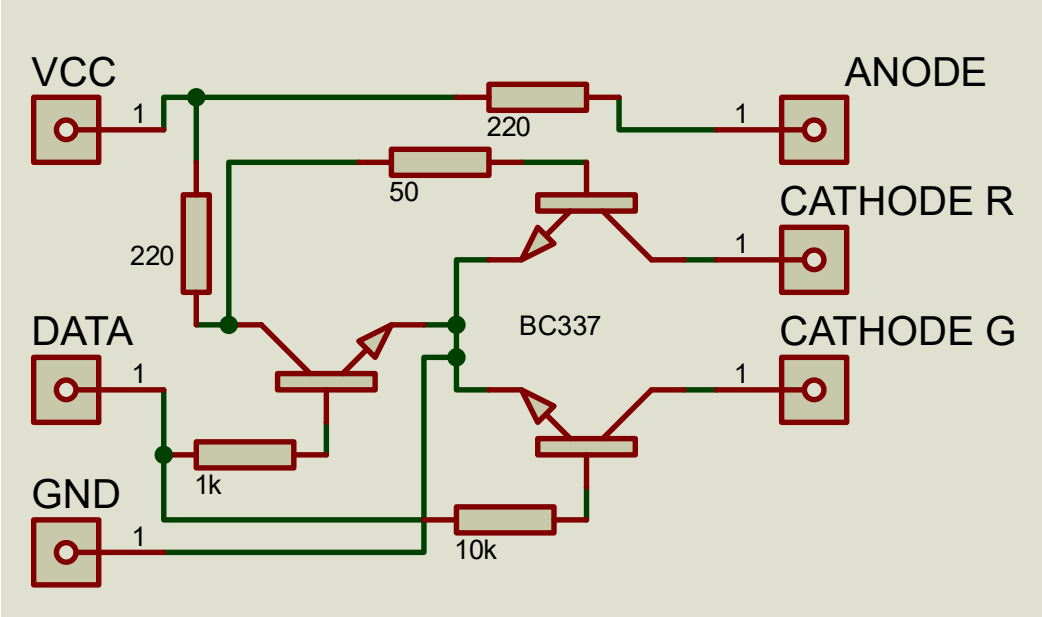


Fig. 6.2 PCB Circuit Design

The above circuit is designed for a special purpose, when the parking slot is empty/Available there be a green led turned on and when the parking slot is filled/Engaged green led turned off and red led will turn on. For this we use BC337 transistor for switching and designing one NOT Gate. The switching of Leds is done on the basis of data coming from IR Sensor used at slot. Hence for 10 slots we made 10 copies of this circuit module which takes input from those 10 IR sensor modules and work accordingly.

CHAPTER 7

Software Setup and Programming

CHAPTER 7

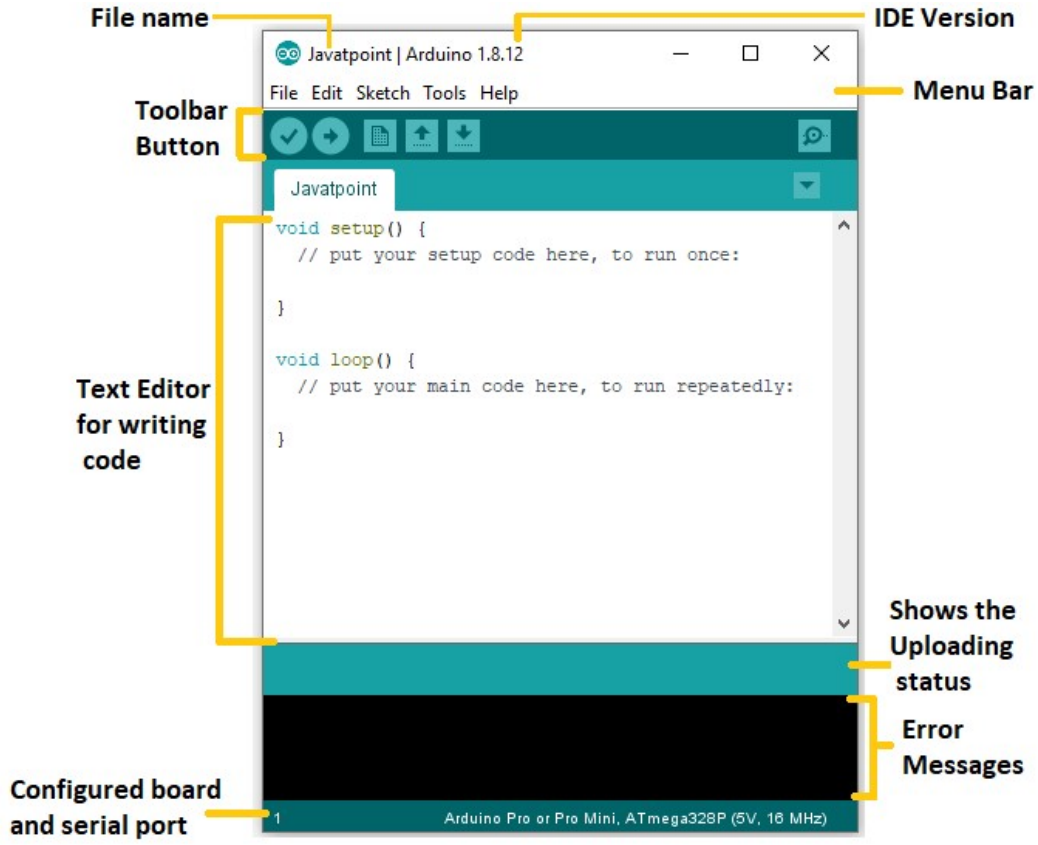
Software Setup and Programming

7.1 Arduino IDE :-

The Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as Windows, Mac OS X, and Linux. It supports the programming languages C and C++. Here, IDE stands for Integrated Development Environment.

The program or code written in the Arduino IDE is often called as sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.'

The Arduino IDE will appear as:



The editor contains the four main areas:

1. A **Toolbar with buttons** for common functions and a series of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

2. The **message area**, gives feedback while saving and exporting and also displays errors.

3. The **text editor** for writing your code.

4. The **text console** displays text output by the Arduino Software (IDE), including complete error messages and other information.

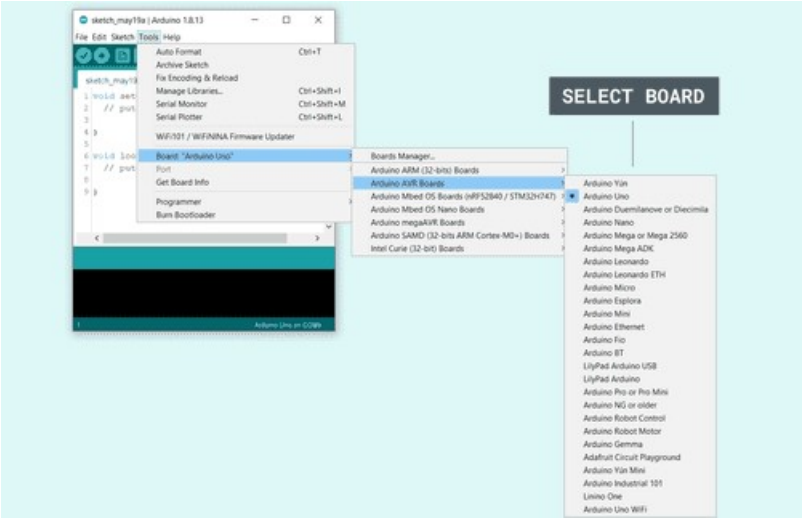
The bottom right-hand corner of the window displays the configured board and serial port.

Now that you are all set up, **let's try to make your board blink!**

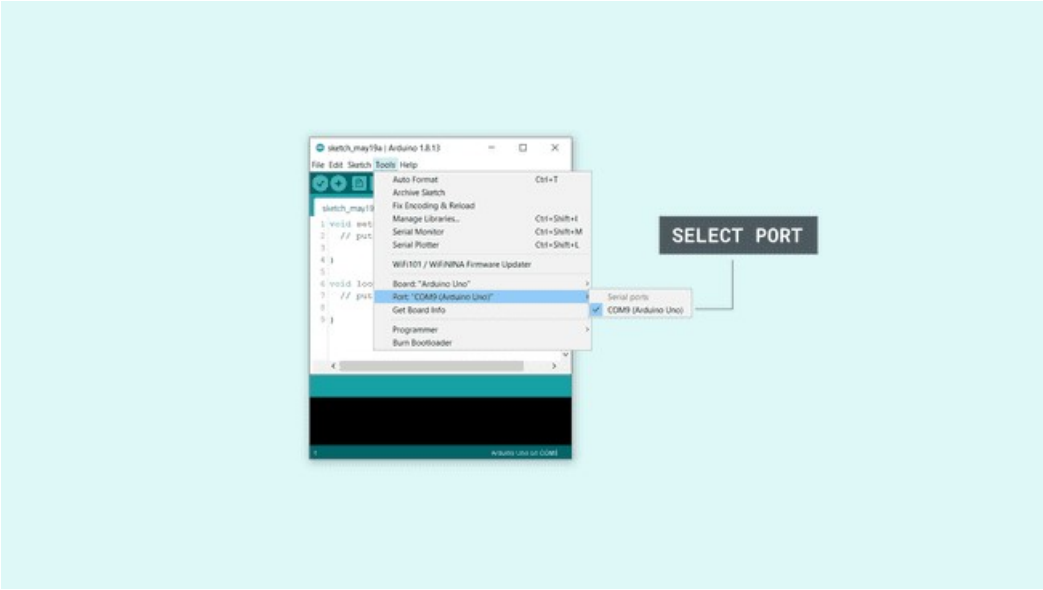
5. **Connect your Arduino** or Genuino board to your computer.

6. Now, you need to **select the right core & board**. This is done by navigating to **Tools > Board > Arduino AVR Boards > Board**. Make sure you select the board that you are using. If you cannot find your board, you can add it from **Tools > Board > Boards Manager**.

Selecting a board

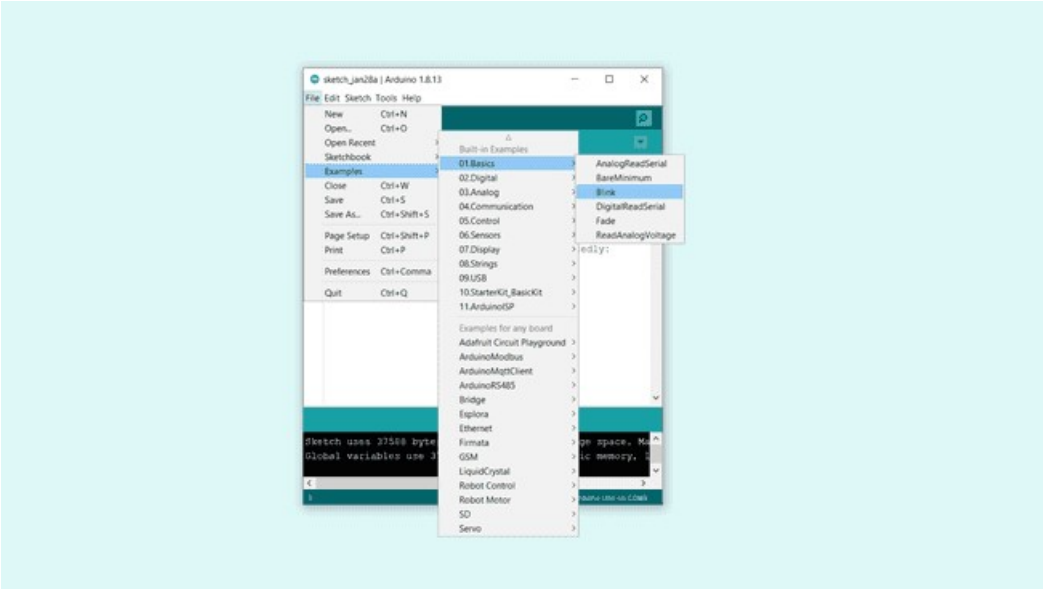


7. Now, let's make sure that your board is found by the computer, by **selecting the port**. This is simply done by navigating to **Tools > Port**, where you select your board from the list.



Selecting the port

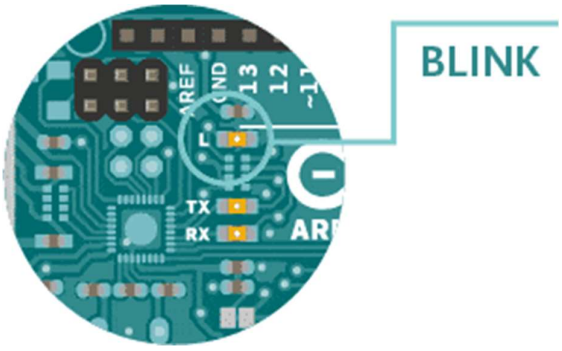
8. Let's **try an example**: navigate to **File > Examples > 01.Basics > Blink.**



Opening an example

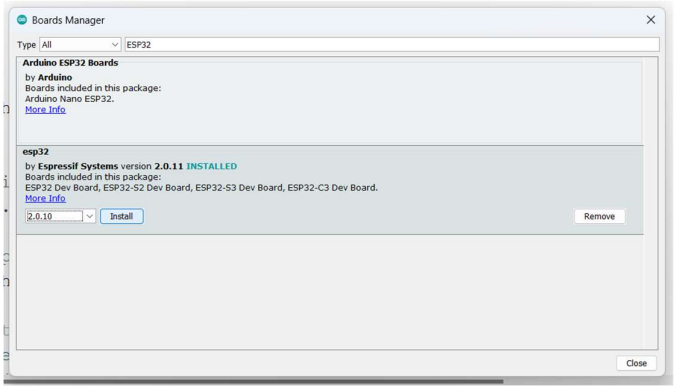
9. To **upload it to your board**, simply click on the arrow in the top left corner. This process takes a few seconds, and it is important to not disconnect the board during this process. If the upload is successful, the message "Done uploading" will appear in the bottom output area.

10. Once the upload is complete, you should then see on your board the yellow LED with an L next to it start blinking. You can **adjust the speed of blinking** by changing the delay number in the parenthesis to 100, and upload the Blink sketch again. Now the LED should blink much faster.

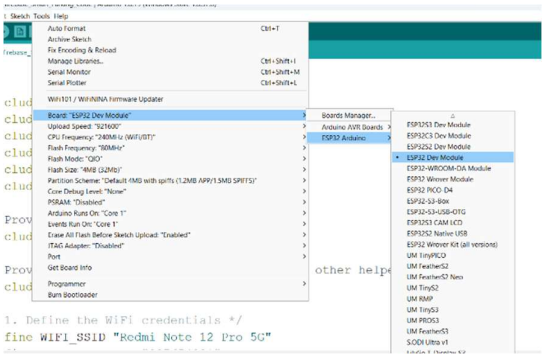


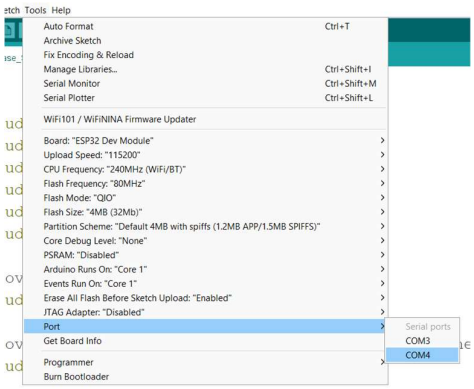
7.2 Setup and Required Libraries :-

For this project we have to first set up Arduino ide for Board ESP-32. For this go to Board Manager and then search for ESP-32 and then install board as shown in figure.

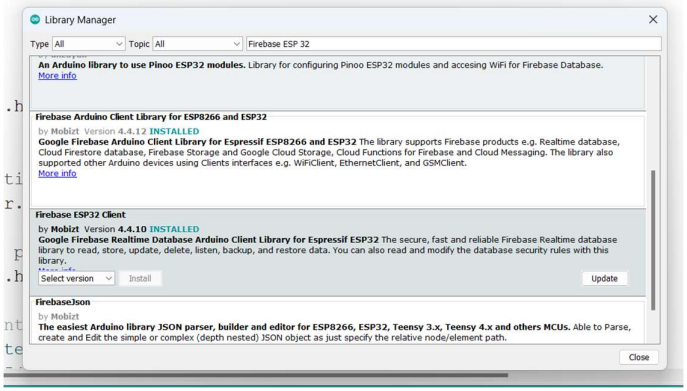


After installing board , go to tools and then select desired board and make sure the setting would be same as in figure.





Now add required libraries for code, specially Firebase ESP32 Client for making connection between firebase and ESP-32.



After successful setup upload the code to ESP-32. Make sure that you have added required credentials to code.

7.3 Programming :-

```
#include <Arduino.h>
#include <WiFi.h>
#include <FirebaseESP-32.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <ESP-32Servo.h>

#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>

/* 1. Define the WiFi credentials */
#define WIFI_SSID "WiFi Name"
#define WIFI_PASSWORD "WiFi Password"

/* 2. Define the API Key */
#define API_KEY "Your web api key"

/* 3. Define the RTDB URL */
#define DATABASE_URL "your database link"

/* 4. Define the user Email and password that alreadey registerd or
added in your project */
#define USER_EMAIL " user Email "
#define USER_PASSWORD " password "

// Define Firebase Data object
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

unsigned long sendDataPrevMillis = 0;
unsigned long count = 0;
#define slot_1 34
#define slot_2 35
#define slot_3 32
#define slot_4 33
#define slot_5 25
```

```
#define slot_6 16
#define slot_7 17
#define slot_8 5
#define slot_9 18
#define slot_10 19

#define ir_enter 26
#define ir_exit 27
#define servo_enter 14
#define servo_exit 12
#define buzzer 4
/* ---------------------------------------------------- */
String line1 = " Welcome! "; // stationary
String line2 = " Its an IoT Based Smart Parking. Developed by: Akash
Bagwan,Jay Barode,Surendra Prajapat. ";

int screenWidth = 16;
int screenHeight = 2;

int stringStart, stringEnd = 0;
int scrollCursor = screenWidth;

LiquidCrystal_I2C LCD = LiquidCrystal_I2C(0x27, 16, 2);
Servo myservo_enter;
Servo myservo_exit;

int pos_enter = 0;
int pos_exit = 0;

int S_1 = 1;
int S_2 = 1;
int S_3 = 1;
int S_4 = 1;
int S_5 = 1;
int S_6 = 1;
int S_7 = 1;
int S_8 = 1;
int S_9 = 1;
int S_10 = 1;
int S_total = 0;
```

```
int counter = 0;
/* ------------------------------------------------- */
void setup()
{ Serial.begin(115200);
  LCD.begin();
  LCD.backlight();

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  LCD.print("Connecting....");

  unsigned long wifiConnectStartTime = millis();
  while (WiFi.status() != WL_CONNECTED && millis() -
wifiConnectStartTime < 10000) {
    Serial.print(".");
    delay(300);
  }

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nFailed to connect to Wi-Fi. Skipping Firebase
setup.");
    LCD.setCursor(0, 1);
    LCD.print("Failed");
    delay(800);
  }
  else {
  Serial.println();
  Serial.print("Connected with IP: ");
  LCD.setCursor(0, 1);
  LCD.print("WiFi-Connected");
  Serial.println(WiFi.localIP());
  Serial.println();
  delay(700);
  Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);
  LCD.clear();
  LCD.setCursor(0, 0);
  LCD.print("Getting Firebase");
  LCD.setCursor(0, 1);
  LCD.print("...");
  /* Assign the api key (required) */
```

```
  config.api_key = API_KEY;

  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  config.database_url = DATABASE_URL;
  config.token_status_callback = tokenStatusCallback;
  Firebase.reconnectNetwork(true);
  Firebase.begin(&config, &auth);
  }
  pinMode(slot_1, INPUT);
  pinMode(slot_2, INPUT);
  pinMode(slot_3, INPUT);
  pinMode(slot_4, INPUT);
  pinMode(slot_5, INPUT);
  pinMode(slot_6, INPUT);
  pinMode(slot_7, INPUT);
  pinMode(slot_8, INPUT);
  pinMode(slot_9, INPUT);
  pinMode(slot_10, INPUT);
  pinMode(ir_enter, INPUT);
  pinMode(ir_exit, INPUT);
  pinMode(buzzer, OUTPUT);

  myservo_enter.attach(servo_enter);
  myservo_exit.attach(servo_exit);
  myservo_enter.write(9);
  myservo_exit.write(9);

  LCD.clear();
  scroll(line1,line2);
  ReadSensorData1();
  if (WiFi.status() != WL_CONNECTED) {Update_Slots(); counter =
S_total; ReadSensorData3();}
  else{Update_Slots(); counter = S_total; ReadSensorData2();}
 }
```

```
void loop()
{  if (Firebase.ready() && (millis() - sendDataPrevMillis > 5000 ||
sendDataPrevMillis == 0))
    { sendDataPrevMillis = millis();
      ReadSensorData2();
    } else{
      ReadSensorData3();
      }
}
/* It will update counter */
void ReadSensorData1(){

    S_1 = digitalRead(slot_1);
    String status1;
    if(S_1==0){status1= "Engaged";}else{status1 = "Available";}

    S_2 = digitalRead(slot_2);
    String status2;
    if(S_2==0){status2 = "Engaged";}else{status2 = "Available";}

    S_3 = digitalRead(slot_3);
    String status3;
    if(S_3==0){status3 = "Engaged";}else{status3 = "Available";}

    S_4 = digitalRead(slot_4);
    String status4;
    if(S_4==0){status4 = "Engaged";}else{status4 = "Available";}

    S_5 = digitalRead(slot_5);
    String status5;
    if(S_5==0){status5 = "Engaged";}else{status5 = "Available";}

    S_6 = digitalRead(slot_6);
    String status6;
    if(S_6==0){status6 = "Engaged";}else{status6 = "Available";}

    S_7 = digitalRead(slot_7);
    String status7;
    if(S_7==0){status7 = "Engaged";}else{status7 = "Available";}
```

```
    S_8 = digitalRead(slot_8);
    String status8;
    if(S_8==0){status8 = "Engaged";}else{status8 = "Available";}

    S_9 = digitalRead(slot_9);
    String status9;
    if(S_9==0){status9 = "Engaged";}else{status9 = "Available";}

    S_10 = digitalRead(slot_10);
    String status10;
    if(S_10==0){status10 = "Engaged";} else{status10 = "Available";}
 }
/* ------------------------------------------------------ */
/* It Will Update data inn Firebase */
void ReadSensorData2(){

    S_1 = digitalRead(slot_1);
    String status1;
    if(S_1==0){
      status1 = "Engaged";
    }else{
      status1 = "Available";
    }
    Firebase.setString(fbdo,F("Sensors/Slot 1"),status1);
    LCD.setCursor(1,1);
    LCD.print("S_1: "+String(status1)+"    ");
    Update_Slots();
    ServoStatus();

    S_2 = digitalRead(slot_2);
    String status2;
    if(S_2==0){
      status2 = "Engaged";
    }else{
      status2 = "Available";
    }
    Firebase.setString(fbdo,F("Sensors/Slot 2"),status2);
    LCD.setCursor(1,1);
    LCD.print("S_2: "+String(status2)+"    ");
    Update_Slots();
```

```
ServoStatus();

S_3 = digitalRead(slot_3);
String status3;
if(S_3==0){
  status3 = "Engaged";
}else{
  status3 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 3"),status3);
LCD.setCursor(1,1);
LCD.print("S_3: "+String(status3)+"    ");
Update_Slots();
ServoStatus();

S_4 = digitalRead(slot_4);
String status4;
if(S_4==0){
  status4 = "Engaged";
}else{
  status4 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 4"),status4);
LCD.setCursor(1,1);
LCD.print("S_4: "+String(status4)+"    ");
Update_Slots();
ServoStatus();

S_5 = digitalRead(slot_5);
String status5;
if(S_5==0){
  status5 = "Engaged";
}else{
  status5 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 5"),status5);
LCD.setCursor(1,1);
LCD.print("S_5: "+String(status5)+"    ");
Update_Slots();
ServoStatus();
```

```
S_6 = digitalRead(slot_6);
String status6;
if(S_6==0){
  status6 = "Engaged";
}else{
  status6 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 6"),status6);
LCD.setCursor(1,1);
LCD.print("S_6: "+String(status6)+"    ");
Update_Slots();
ServoStatus();

S_7 = digitalRead(slot_7);
String status7;
if(S_7==0){
  status7 = "Engaged";
}else{
  status7 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 7"),status7);
LCD.setCursor(1,1);
LCD.print("S_7: "+String(status7)+"    ");
Update_Slots();
ServoStatus();

S_8 = digitalRead(slot_8);
String status8;
if(S_8==0){
  status8 = "Engaged";
}else{
  status8 = "Available";
}
Firebase.setString(fbdo,F("Sensors/Slot 8"),status8);
LCD.setCursor(1,1);
LCD.print("S_8: "+String(status8)+"    ");
Update_Slots();
ServoStatus();
```

```
    S_9 = digitalRead(slot_9);
    String status9;
    if(S_9==0){
      status9 = "Engaged";
    }else{
      status9 = "Available";
    }
    Firebase.setString(fbdo,F("Sensors/Slot 9"),status9);
    LCD.setCursor(1,1);
    LCD.print("S_9: "+String(status9)+"    ");
    Update_Slots();
    ServoStatus();

    S_10 = digitalRead(slot_10);
    String status10;
    if(S_10==0){
      status10 = "Engaged";
    }else{
      status10 = "Available";
    }
    Firebase.setString(fbdo,F("Sensors/Slot 10"),status10);
    LCD.setCursor(1,1);
    LCD.print("S_10: "+String(status10)+"    ");
    Update_Slots();
    ServoStatus();
}
void Update_Slots(){
    S_total = S_1 + S_2 + S_3 + S_4 + S_5 + S_6 + S_7 + S_8 + S_9 +
S_10 ;
    Firebase.setString(fbdo,F("Sensors/Total_Slots"),S_total);
    LCD.setCursor(0,0);
    LCD.print("Total_Slots:"+String(counter)+"  ");
}
void ServoStatus(){

      if(digitalRead(ir_enter)==0 && (S_total > 0 && counter > 0)){
      counter = counter - 1;
      myservo_enter.write(99);
      LCD.setCursor(1,1);
      LCD.print("Opening Gate...");
```

```
      digitalWrite(buzzer,HIGH);
      delay(3000);
      digitalWrite(buzzer,LOW);
      myservo_enter.write(9);
      }

      if(digitalRead(ir_exit)==0){
      counter = counter + 1;
      myservo_exit.write(99);
      LCD.setCursor(1,1);
      LCD.print("Opening Gate...");
      digitalWrite(buzzer,HIGH);
      delay(3000);
      digitalWrite(buzzer,LOW);
      myservo_exit.write(9);
      }
      if(counter==0){LCD.setCursor(1,1);
      LCD.print("Parking is Full");}
}
void scroll(String line1, String line2){
  for(int i = 1 ; i <=line2.length();i++){
  LCD.setCursor(0, 0); // Seting the cursor on first row
  LCD.print(line1); // To print line1 message
  LCD.setCursor(scrollCursor, 1); // Seting the cursor on first row
and (scrolling from left end to right)
  LCD.print(line2.substring(stringStart,stringEnd)); // To print line1
first character "T"
  delay(230);

  LCD.clear(); // clear message

  if(stringStart == 0 && scrollCursor > 0){
    scrollCursor--; // Moving cursor from 16 to 0
    stringEnd++; // Character T, H, I, S ...
                 // it will print out character from 0 to 15 the whole
length of the screen
  }
  else if (stringStart == stringEnd){ // start all over again
    stringStart = stringEnd = 0;
    scrollCursor = screenWidth;
```

```
  }
  else if (stringEnd == line1.length() && scrollCursor == 0) { // if
reach to the end c haracter
    stringStart++;
  }
  else { // it will print out character from (1 to 16) to end
character (this case it's !))
    stringStart++;
    stringEnd++;
  }
  }
}
/* It will print info on LCd in case of Offline System i.e not
connected to wi-fi and firebase*/
void ReadSensorData3(){

    S_1 = digitalRead(slot_1);
    String status1;
    if(S_1==0){
      status1 = "Engaged";
    }else{
      status1 = "Available";
    }
    LCD.setCursor(1,1);
    LCD.print("S_1: "+String(status1)+"    ");
    Update_Slots();
    ServoStatus();
    delay(900);

    S_2 = digitalRead(slot_2);
    String status2;
    if(S_2==0){
      status2 = "Engaged";
    }else{
      status2 = "Available";
    }
    LCD.setCursor(1,1);
    LCD.print("S_2: "+String(status2)+"    ");
    Update_Slots();
    ServoStatus();
```

```
delay(900);

S_3 = digitalRead(slot_3);
String status3;
if(S_3==0){
  status3 = "Engaged";
}else{
  status3 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_3: "+String(status3)+"    ");
Update_Slots();
ServoStatus();
delay(900);

S_4 = digitalRead(slot_4);
String status4;
if(S_4==0){
  status4 = "Engaged";
}else{
  status4 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_4: "+String(status4)+"    ");
Update_Slots();
ServoStatus();
delay(900);

S_5 = digitalRead(slot_5);
String status5;
if(S_5==0){
  status5 = "Engaged";
}else{
  status5 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_5: "+String(status5)+"    ");
Update_Slots();
ServoStatus();
delay(900);
```

```
S_6 = digitalRead(slot_6);
String status6;
if(S_6==0){
  status6 = "Engaged";
}else{
  status6 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_6: "+String(status6)+"    ");
Update_Slots();
ServoStatus();
delay(900);

S_7 = digitalRead(slot_7);
String status7;
if(S_7==0){
  status7 = "Engaged";
}else{
  status7 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_7: "+String(status7)+"    ");
Update_Slots();
ServoStatus();
delay(900);

S_8 = digitalRead(slot_8);
String status8;
if(S_8==0){
  status8 = "Engaged";
}else{
  status8 = "Available";
}
LCD.setCursor(1,1);
LCD.print("S_8: "+String(status8)+"    ");
Update_Slots();
ServoStatus();
delay(900);
```

```
    S_9 = digitalRead(slot_9);
    String status9;
    if(S_9==0){
      status9 = "Engaged";
    }else{
      status9 = "Available";
    }
    LCD.setCursor(1,1);
    LCD.print("S_9: "+String(status9)+"    ");
    Update_Slots();
    ServoStatus();
    delay(900);

    S_10 = digitalRead(slot_10);
    String status10;
    if(S_10==0){
      status10 = "Engaged";
    }else{
      status10 = "Available";
    }
    LCD.setCursor(1,1);
    LCD.print("S_10: "+String(status10)+"    ");
    Update_Slots();
    ServoStatus();
    delay(900);
}
```

CHAPTER 8

Firebase Database

CHAPTER 8

Firebase Database

8.1 Introduction to Database :-

Here we are using Firebase as Database. The Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON. The data is synchronized in real-time to every connected client. All of our clients share one Realtime Database instances and automatically receive updates with the newest data, when we build cross-platform applications with our iOS, and JavaScript SDKs.

The Firebase Realtime Database is a NoSQL database from which we can store and sync the data between our users in real-time. It is a big JSON object which the developers can manage in real-time. By using a single API, the Firebase database provides the application with the current value of the data and updates to that data. Real-time syncing makes it easy for our users to access their data from any device, be it web or mobile.

The Realtime database helps our users collaborate with one another. It ships with mobile and web SDKs, which allow us to build our app without the need for servers. When our users go offline, the Real-time Database SDKs use local cache on the device for serving and storing changes. The local data is automatically synchronized, when the device comes online.

Key capabilities :

A Real-time database is capable of providing all offline and online services. These capabilities include accessibility from the client device, scaling across multiple databases, and many more.

Real-time :

The Firebase Real-time database uses data synchronization instead of using HTTP requests. Any connected device receives the updates within milliseconds. It doesn't think about network code and provides collaborative and immersive experiences.

Offline :-

The Firebase Database SDK persists our data to disk, and for this reason, Firebase apps remain responsive even when offline. The client device receives the missed changes, once connectivity is re-established.

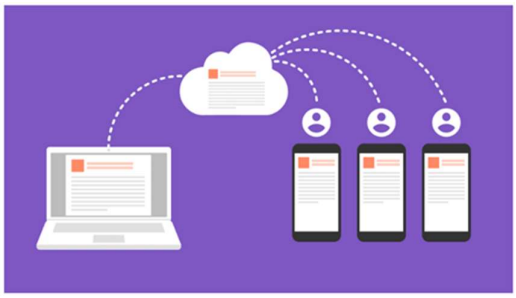
Accessible from client devices :-

There is no need for an application server to access the Firebase Real-time database. We can access it directly from a mobile device or web browser. Data validation and security are available through the Firebase Real-time Database Security Rules, expression-based rules executed when data is read or written.

Working of Firebase Realtime database: -

The Firebase Realtime Database lets you build rich, collaborative applications by allowing secure access to the database directly from client-side code. Data is persisted locally, and even while offline, realtime events continue to fire, giving the end user a responsive experience. When the device regains connection, the Realtime Database synchronizes the local data changes with the remote updates that occurred while the client was offline, merging any conflicts automatically.

The Realtime Database provides a flexible, expression-based rules language, called Firebase Realtime Database Security Rules, to define how your data should be structured and when data can be read from or written to. When integrated with Firebase Authentication, developers can define who has access to what data, and how they can access it.



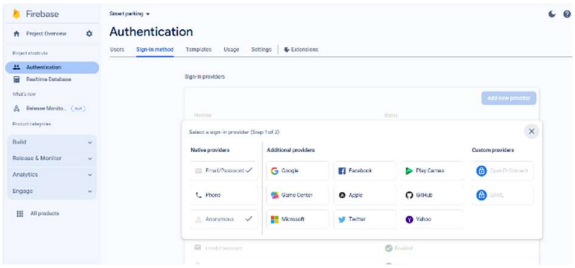
The Realtime Database is a NoSQL database and as such has different optimizations and capabilities compared to a relational database. The Realtime Database API is designed to only allow operations that can be executed quickly. This lets you build a great realtime experience that can serve millions of users without compromising on responsiveness. Because of this, it is important to think about how users need to access your data and then structure it accordingly.

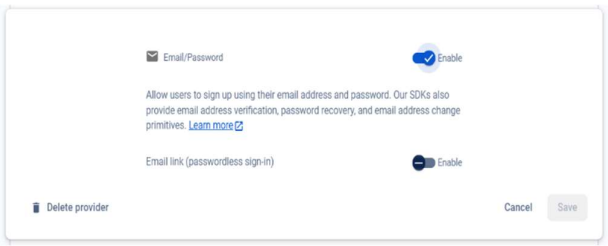
8.2 Setting Up Database :-

1. For this go to firebase console and create a new project and then go to build and create RTDB with schema as shown in figure.

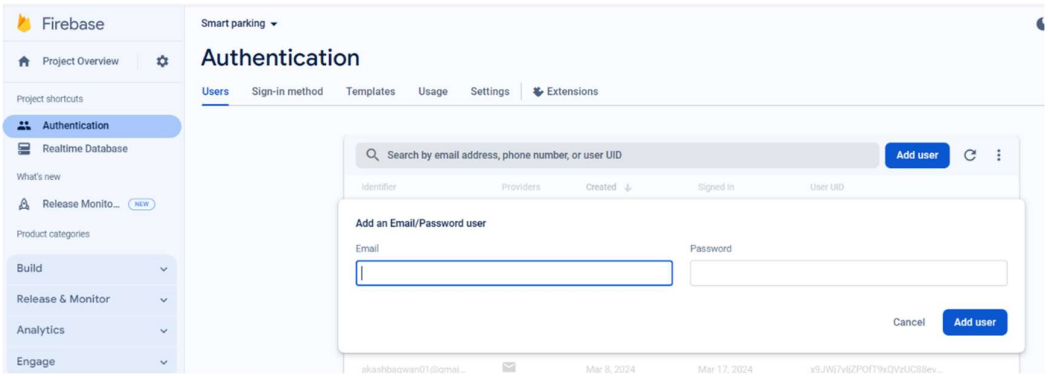
Fig. 8.1 Firebase Database

2. After this go to authentication and then sign in method and add new provider and select Email/Password and then save it.





3. Now go to Users and add new user and give email address and password and click Add new user. These credentials will be used in ESP-32 code .



Now you have done.

CHAPTER 9

Android Application

CHAPTER 9

Android Application

9.1 Introduction to Android Studio :-

Android Studio is the official integrated development environment (IDE) for Android app development. Here's an overview:

IDE Overview:

- Android Studio provides a comprehensive environment for developing Android applications.
- It is based on the IntelliJ IDEA IDE, offering powerful features and tools specifically tailored for Android development.

Features:

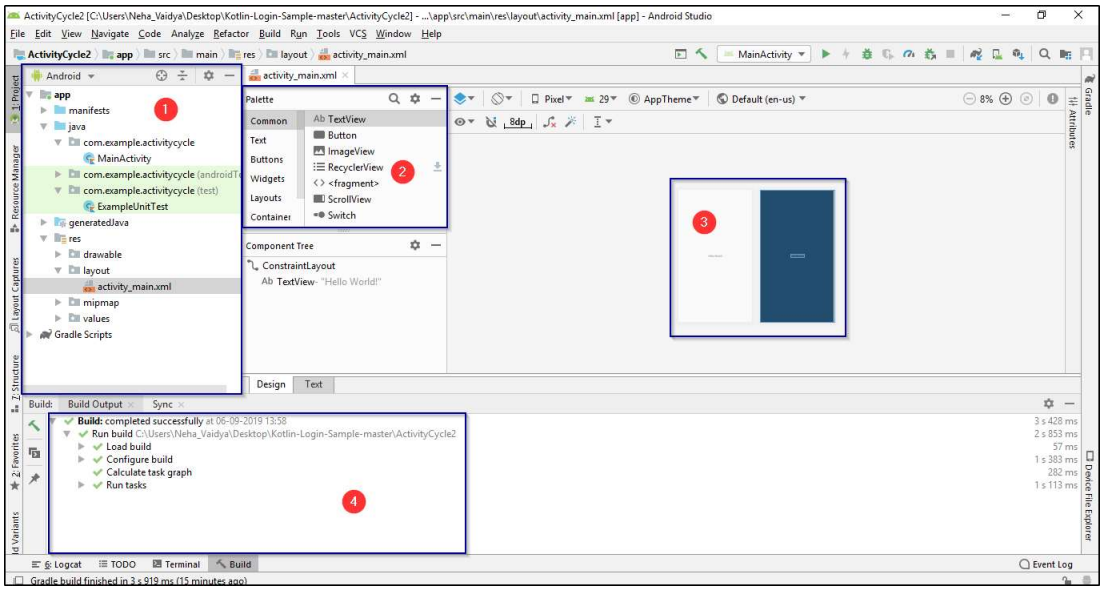
- Code Editor: Android Studio includes a sophisticated code editor with features like syntax highlighting, code completion, and refactoring tools.
- Layout Editor: Developers can design app layouts visually using the Layout Editor, which supports drag-and-drop UI design.
- Debugging Tools: Android Studio provides advanced debugging tools for identifying and fixing issues in code, including breakpoints, watches, and real-time error checking.
- Performance Profiling: Developers can analyze app performance and identify bottlenecks using built-in profiling tools.
- Version Control: Android Studio integrates with version control systems like Git, allowing developers to manage their codebase efficiently.
- Testing Tools: The IDE supports various testing frameworks for unit testing, integration testing, and UI testing.
- Build System: Android Studio uses the Gradle build system, which offers flexibility and customization options for building Android apps.
- Emulator: It includes a built-in Android Emulator for testing apps on virtual devices with different configurations.
- Device Deployment: Developers can deploy apps directly to physical devices for testing and debugging.

Supported Languages:

- Android Studio supports programming in Kotlin, Java, and C/C++ for developing Android applications.
- Kotlin has gained popularity as an official language for Android development due to its concise syntax and advanced features.

Integration with Google Services:

- Android Studio seamlessly integrates with various Google services and APIs, such as Google Maps, Firebase, AdMob, and Google Play Services.
- Developers can easily add features like maps, analytics, authentication, and cloud messaging to their apps using these integrations.



1. This section represents the project structure of an Android Application which comprise of the layout, result, and Gradle scripts.
2. This window is a Palette which comprises of a component that is essential for building an application. You can add a button, layout, image as per the requirement on to your app window.
3. This is a section where you can actually build your Android application using the Palette components. All that you need to do is – just drag and drop the components.
4. This is a console in Android Studio which displays the result and the configuration tasks.

9.2 Development of Application :-

For the development of IoT android app everyone is free to design his own user interface. The important thig is the connection of that app with firebase database.

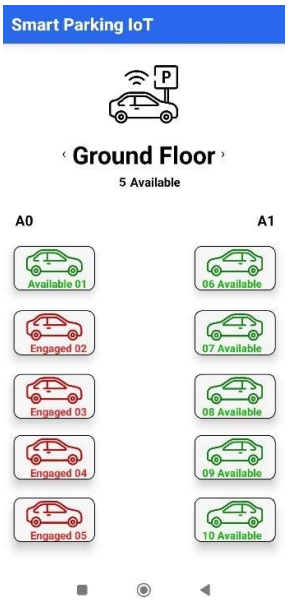
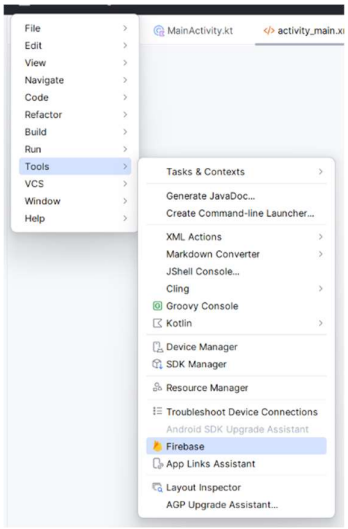
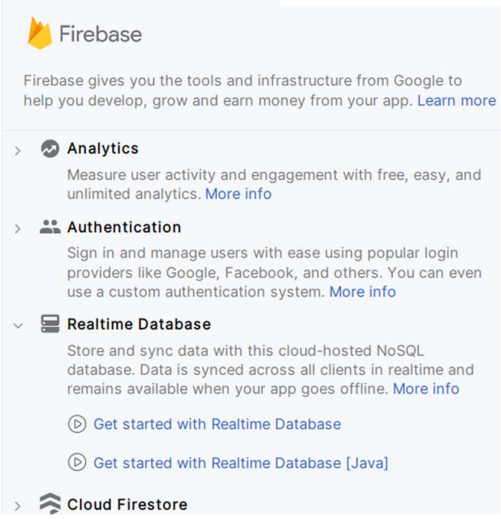
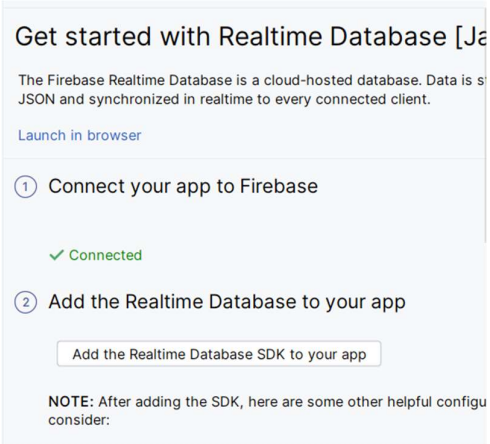


Fig. 9.1 Android App

1. For the connection, in android studio just go to tools and then click to Firebase.
2. After that go to Real Time Database and then go with option "Get started with Realtime Database [JAVA]".
3. Then click on 1st option "Connect your app to Firebase".
4. Then 2nd "Add the Realtime Database to your app" and now its done.







CHAPTER 10

Result

CHAPTER 10

Result

10.1 Result :-

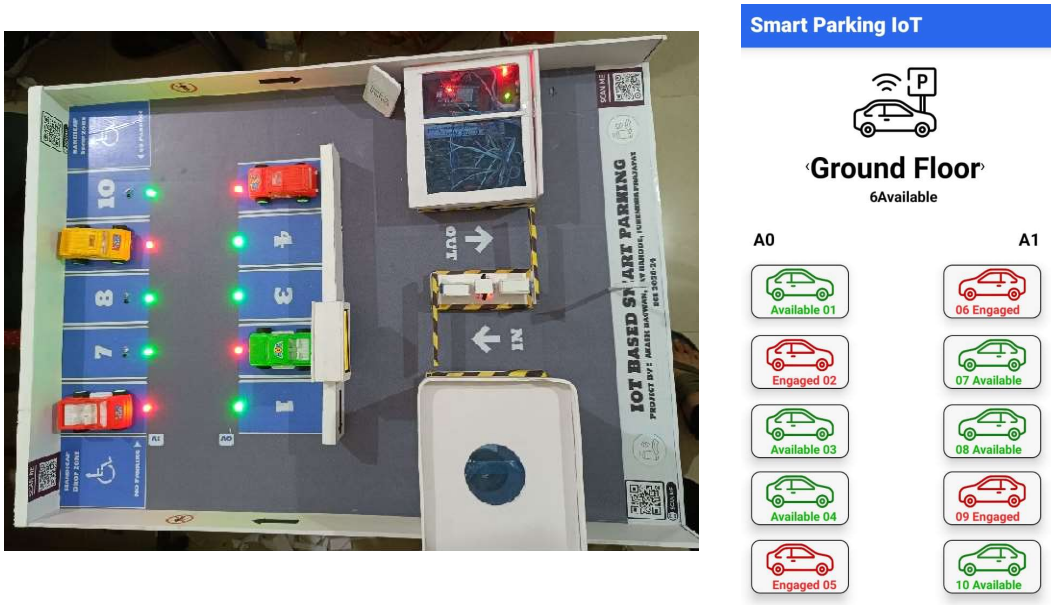


Fig. 10.1 Result

The above picture shows the result that we got after successful completion of project. This shows that when a particular slot is Engaged in parking lot, same slot displayed as Engaged in android application and Red Led glows when slot is Engaged by default when slot is Available Green Led will glow.

CHAPTER 11
Conclusion

CHAPTER 11
Conclusion

11.1 Conclusion :-

IoT Based Smart parking have emerged as a promising solution to address the growing challenges associated with urban parking management. By leveraging advanced technologies and data-driven insights, these systems offer a range of potential benefits, including increased efficiency, improved convenience, enhanced accessibility, reduced congestion, and increased revenue. However, careful consideration of the initial investment costs, reliance on technology, maintenance costs, data privacy concerns, and potential for misuse is necessary before implementing smart parking systems in any given context. Despite these limitations, smart parking systems hold the potential to revolutionize the way we park our cars, making parking more efficient, convenient, and sustainable for all.

CHAPTER 12
References

CHAPTER 12

References

12.1 References :-

1. Aggarwal, S. (2011). Smart Parking System using Sensors and Networking Technologies. International Journal of Electronics and Computer Science Engineering (IJECSE), 1(2), 1-8.
2. Kumar, S. (2023). Smart Car Parking System. International Journal for Research in Applied Science and Engineering Technology (IJRASTE), 11(5), 707-711.
3. Idris, M. Y., Noor, N. M., & Bakri, M. A. (2009). Smart Car Parking System Using Wireless Sensor Networks (WSNs) and RFID Technology. International Journal of Computer Science and Network Security, 9(6), 155-160.

Articles ….

4. "Harnessing Technology to Solve Urban Parking Woes: The Rise of Smart Parking Systems"
5. "A City Transformed: The Impact of Smart Parking Systems on Urban Traffic and Congestion

Text Books …..

6. Programming Arduino: Getting Started With Sketches (second edition)
7. Encyclopedia of Electronic Components Volume 3
8. Servo Motors and Industrial Control Theor