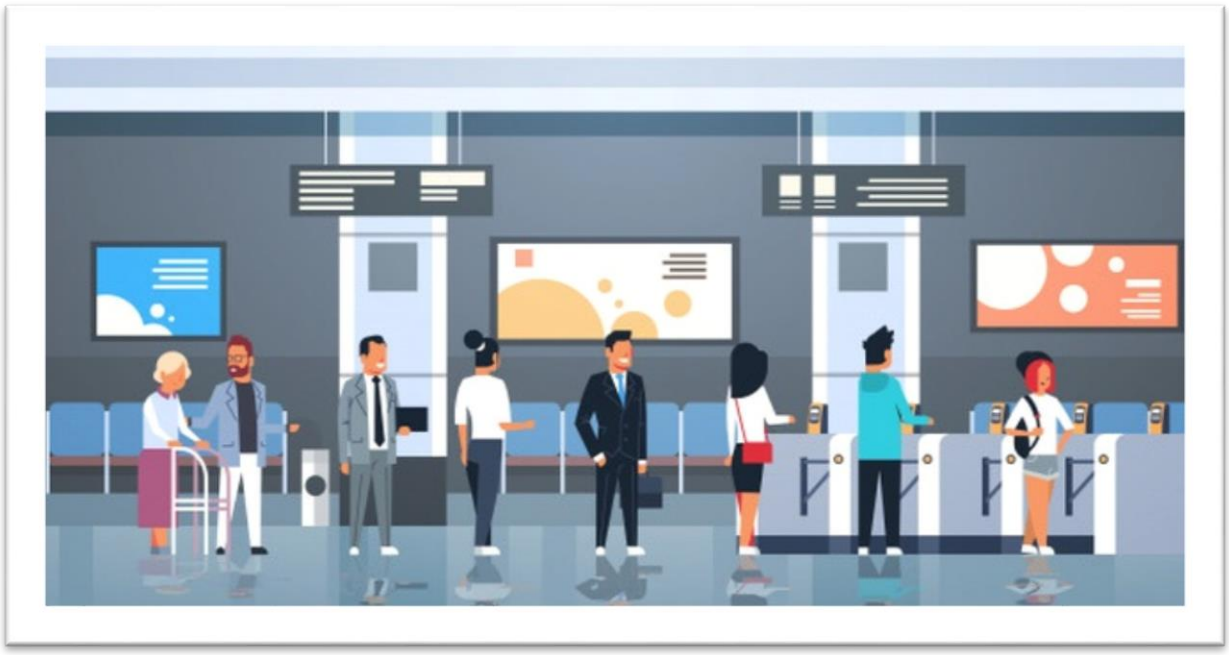


Automated Railway Ticketing System

DESIGN MANUAL

Table of Contents

General Information

HARDWARE OVERVIEW

Physical Structural Design

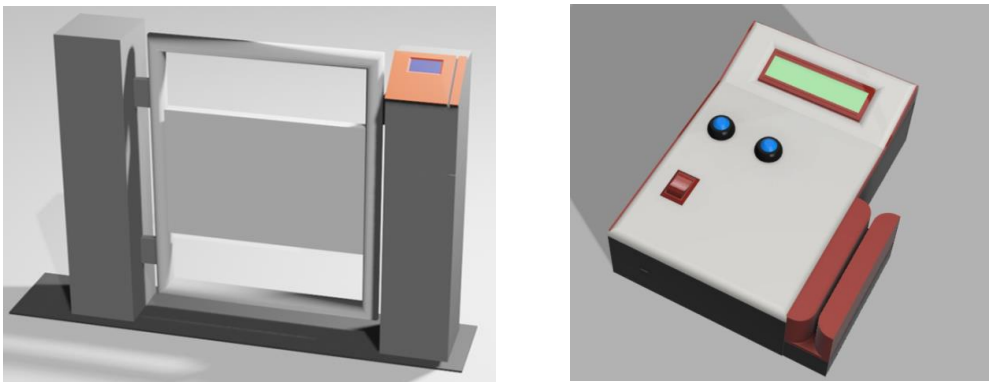


Figure 1.0

SYSTEM OVERVIEW

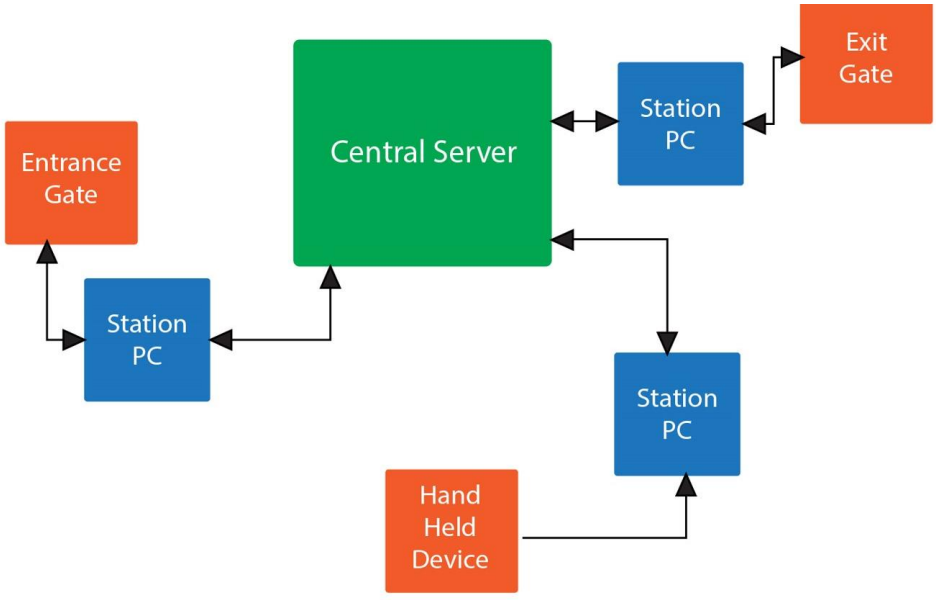


Figure 2.0

SOFTWARE OVERVIEW

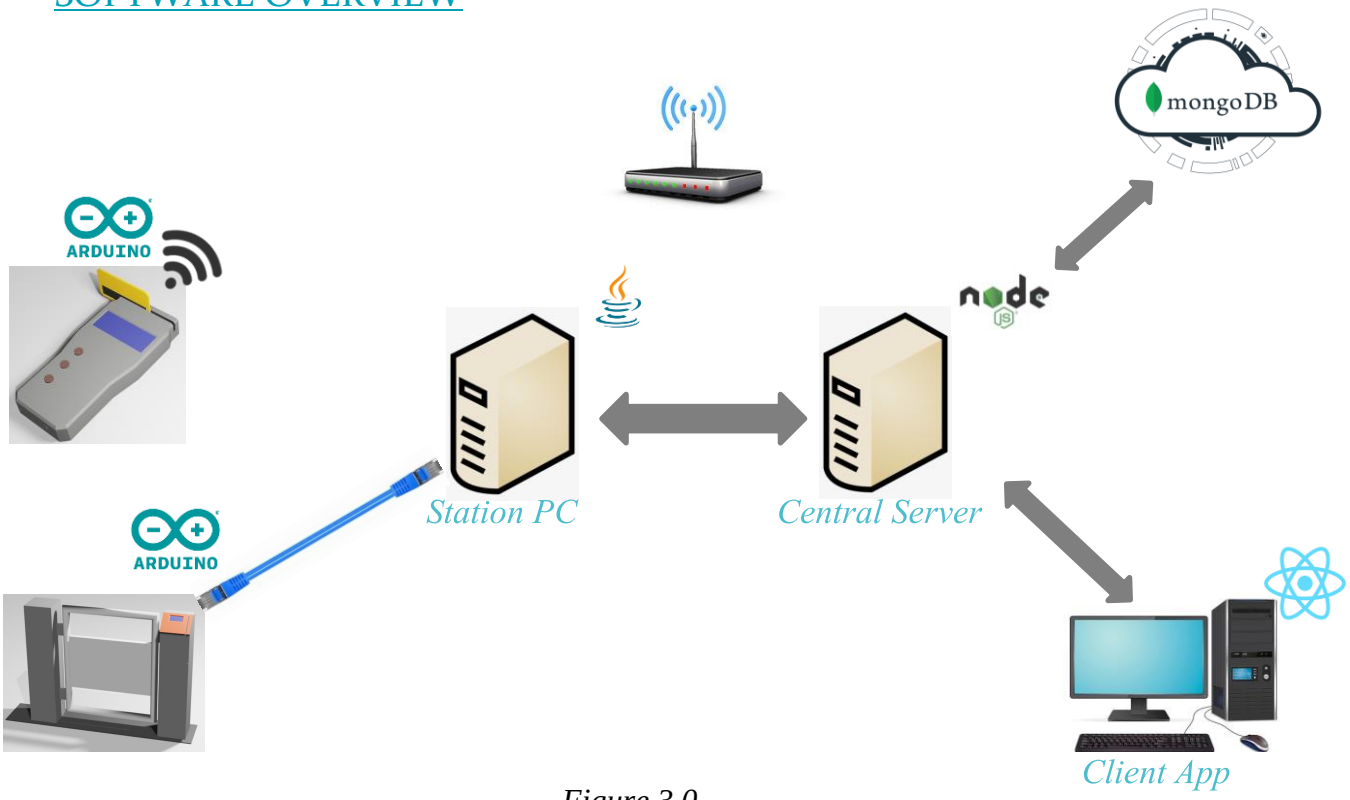


Figure 3.0

Hand-Held device

DESIGN OVERVIEW

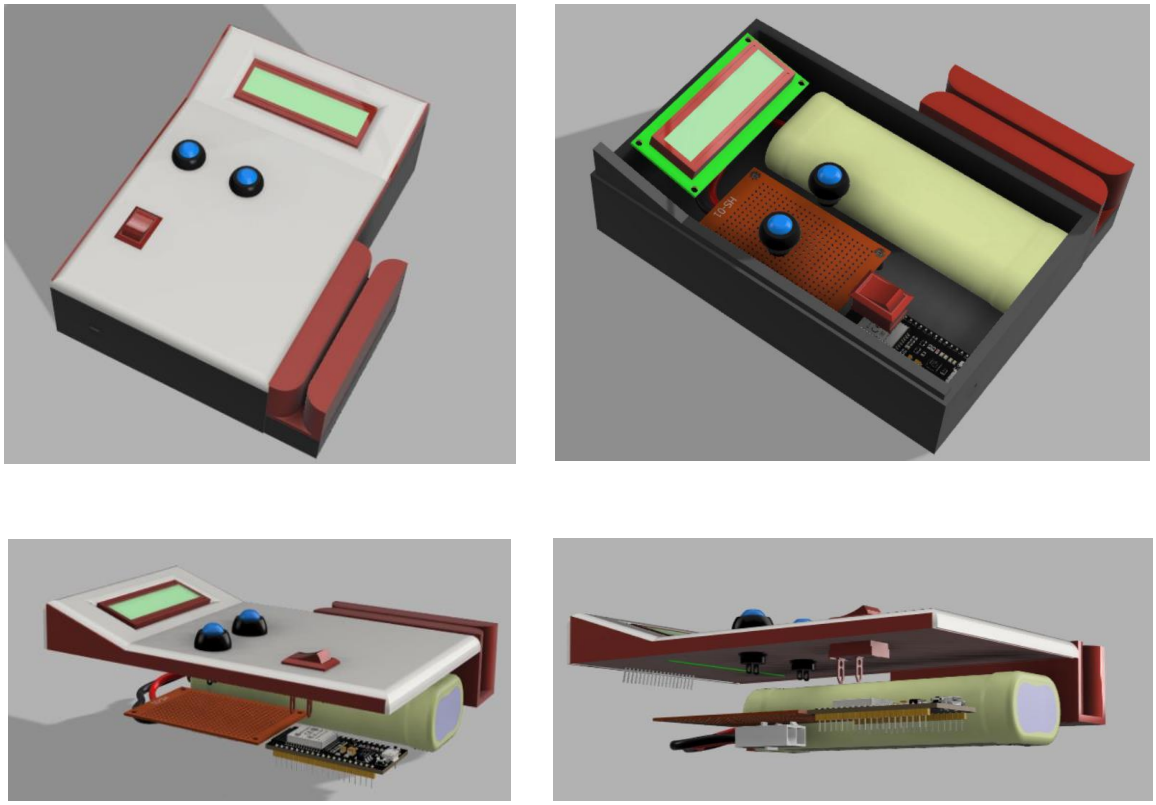
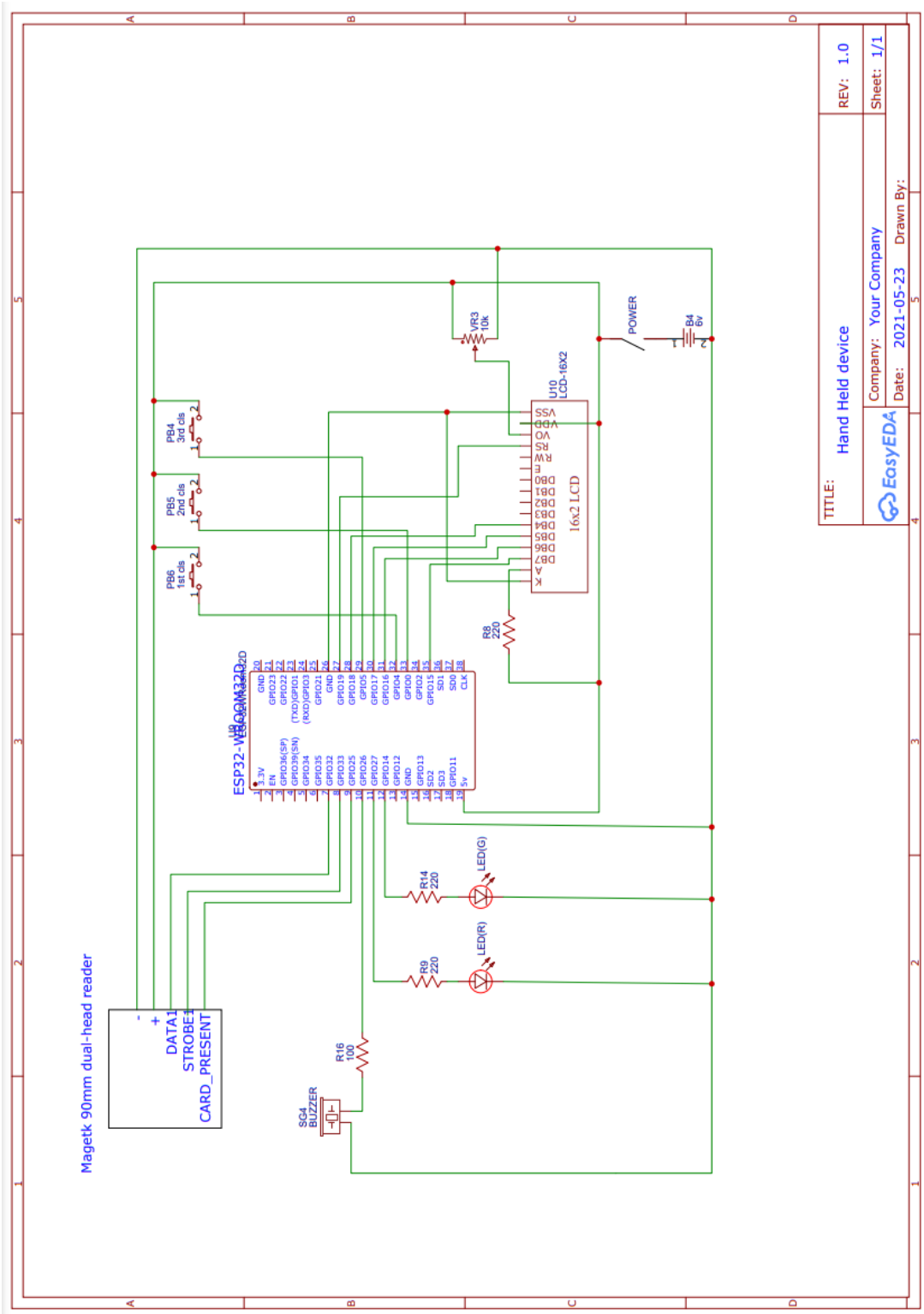
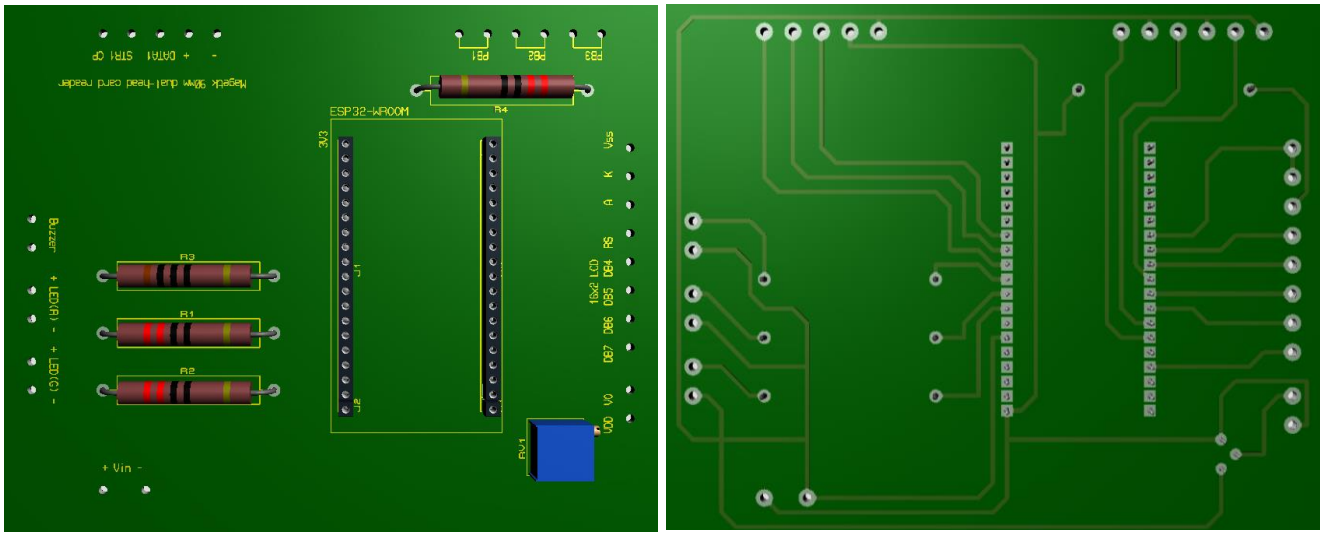


Figure 4.0

CIRCUIT DIAGRAM OF THE HAND HELD DEVICE IS AS FOLLOWS



PCB DESIGN OF THE HAND HELD DEVICE IS AS FOLLOWS



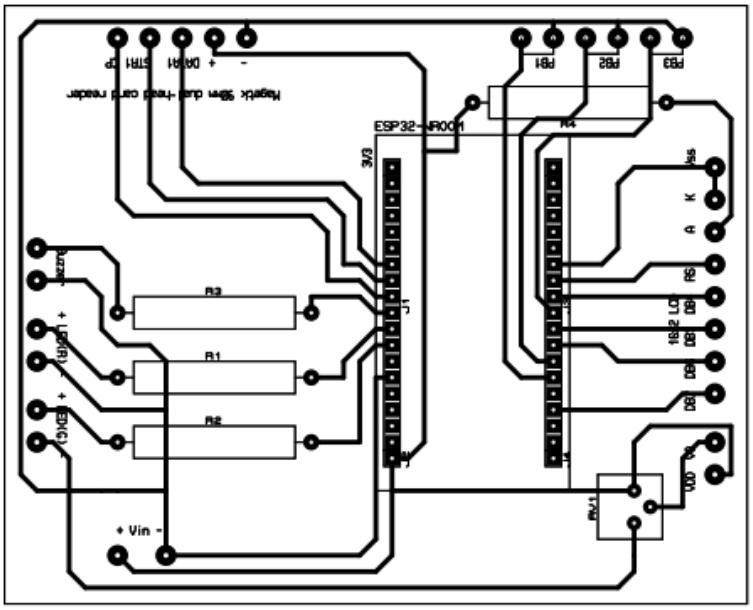


Figure: PCB Designs

ESP32 DEVELOPMENT BOARD

ESP32 is a microcontroller development board that is a product of Espressif, which is developed for the applications of IoT and wearable electronics which contains inbuilt WiFi and Bluetooth communications modules with UART, SPI, I2C, I2S and CAN bus communication interfaces.

Product Documentation: https://docs.espressif.com/projects/esp-idf/en/latest/i

FEATURES

1. CPU and Memory

- Xtensa® single-/dual-core 32-bit LX6 microprocessor(s), up to 600 MIPS
- 448 KB ROM
- 520 KB SRAM
- 16 KB SRAM in RTC
- QSPI supports multiple flash/SRAM chips

2. Clocks and Timers

- Internal 8 MHz oscillator with calibration
- Internal RC oscillator with calibration
- External 2 MHz ~ 60 MHz crystal oscillator (40 MHz only for Wi-Fi/BT functionality)
- External 32 kHz crystal oscillator for RTC with calibration
- Two timer groups, including 2 x 64-bit timers and 1 x main watchdog in each group
- One RTC timer
- RTC watchdog

3. Advanced Peripheral Interfaces

- 34 × programmable GPIOs
- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit DAC
- 10 × touch sensors
- 4 × SPI
- 2 × I2S
- 2 × I2C
- 3 × UART
- 1 host (SD/eMMC/SDIO)
- 1 slave (SDIO/SPI)

- Ethernet MAC interface with dedicated DMA and IEEE 1588 support
- CAN 2.0
- IR (TX/RX)

● Motor PWM
● LED PWM up to 16 channels
● Hall sensor

4. WiFi

● Support Station-only mode, AP-only mode, Station/AP-coexistence mode
● Support IEEE-802.11B, IEEE-802.11G, IEEE802.11N and APIs to configure the protocol mode
● Support WPA/WPA2/WPA2-Enterprise and WPS

5. Bluetooth

● Compliant with Bluetooth v4.2 BR/EDR and BLE specifications
● Class-1, class-2 and class-3 transmitter without external power amplifier
● Multi-connections in Classic BT and BLE

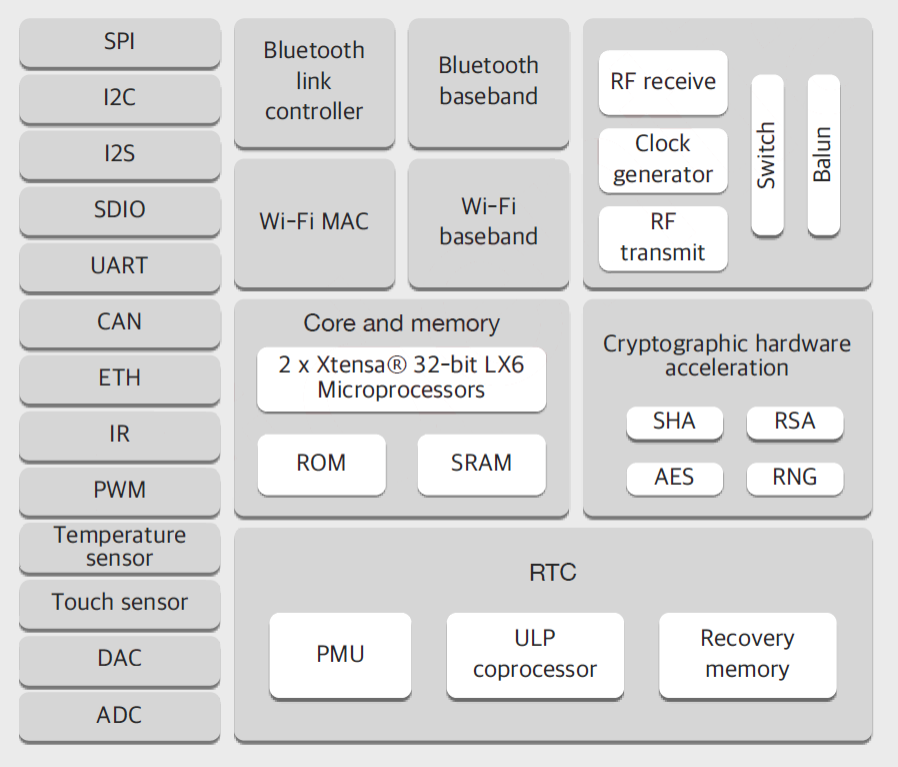


Figure: Functional Block Diagram

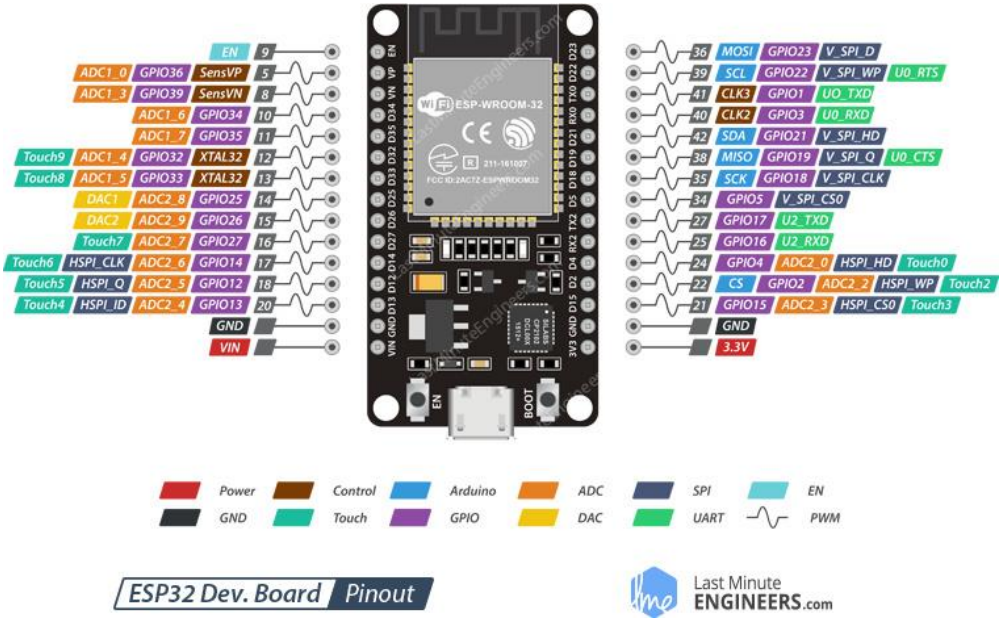


Figure: ESP32 Dev Board Pinout

16X2 LCD DISPLAY

Product Documentation: https://components101.com/displays/16x2-lcd-pinout-datasheet

Figure: 16x2 lcd display pin diagram

Pin No:	*Pin Name:*	*Description*
1	Vss (Ground)	Ground pin connected to system ground
2	Vdd (+5 Volt)	Powers the LCD with +5V (4.7V – 5.3V)
3	VE (Contrast V)	Decides the contrast level of display. Grounded to get maximum contrast.
4	Register Select	Connected to Microcontroller to shift between command/data register
5	Read/Write	Used to read or write data. Normally grounded to write data to LCD
6	Enable	Connected to Microcontroller Pin and toggled between 1 and 0 for data acknowledgement
7	Data Pin 0	Data pins 0 to 7 forms a 8-bit data line. They can be connected to Microcontroller to send 8-bit data. These LCD's can also operate on 4-bit mode in such case Data pin 4,5,6 and 7 will be left free.
8	Data Pin 1	
9	Data Pin 2	
10	Data Pin 3	
11	Data Pin 4	
12	Data Pin 5	
13	Data Pin 6	
14	Data Pin 7	
15	LED Positive	Backlight LED pin positive terminal

16	LED Negative	Backlight LED pin negative terminal

Features

- Operating Voltage is 4.7V to 5.3V
- Current consumption is 1mA without backlight
- Alphanumeric LCD display module, meaning can display alphabets and numbers
- Consists of two rows and each row can print 16 characters.
- Each character is built by a 5×8 pixel box
- Can work on both 8-bit and 4-bit mode
- It can also display any custom generated characters
- Available in Green and Blue Backlight

MAGETK 90MM DUAL-HEAD READER

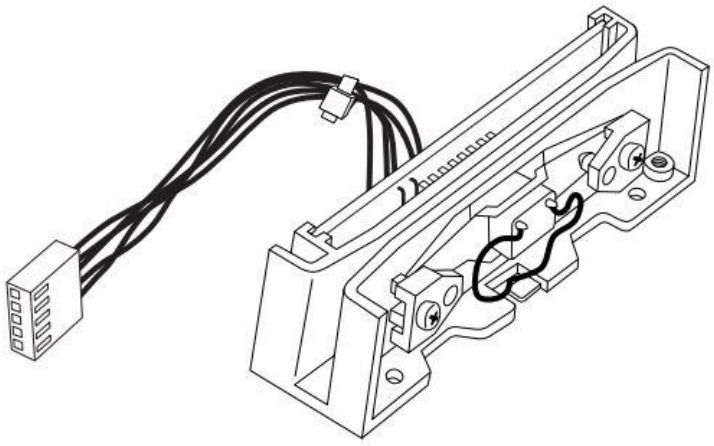


Figure: 90-millimeter Compatible Swipe Reader

Product Documentation:
https://www.magtek.com/content/documentationfiles/d99831083.pdf

Specification

IEC: Meets or Exceeds Requirements for:	IEC 1000-4-2 ESD (Electro Static Discharge) IEC 1000-4-3 Radiated EMC Field (2X requirement) IEC 1000-4-4 Electrical Fast Transient Burst requirement (transmission on I/O cable)
Flammability	Meets UL94V-0
Recording Method	Two-Frequency Coherent Phase (F2F)
Speed	Card speed through the unit may vary from: 2 to 125 in/s at 75 bpi (5.1 to 318 cm/s at 29.5 b/cm) 2 to 60 in/s at 210 bpi (5.1 to 152.4 cm/s at 82.7 b/cm)
Power Requirements	Single Track: 2.7 to 5.5VDC at 1mA, typical Dual Track: 2.7 to 5.5VDC at 2mA, typical 3 Tracks: 2.7 to 5.5VDC at 3mA, typical
Output Signal Levels	V_{ol} = 0.4V at 2mA V_{oh} = V_{cc} -0.5V at -2mA
Operating Temperature	-30°C to 70° C
Operating Humidity	10% to 90% relative humidity
Life	300,000 passes Single Track 1,000,000 passes Multi-Track
MTBF Electronics	125,000 h
Dimensions	Length: 3.54" (90.0mm) Height: 0.95" (24.13mm) Width: 0.88" (22.4mm)
Cable Length:	Single Track: 6" (150mm) Dual Track: 4" (101.6mm) 3 Track 5" (127mm)
Connector	See Section 2, Connectors
Colors available	Black, Standard

NIMH BATTERY PACK

Figure: Battery pack with charger

Battery Specifications:

- Voltage: 7.2V
- Capcity: 3300 mAh
- Chemistry: NiMH (Nickel Metal Hydride)
- Size: 6 x Sub-C Cells
- Configuration: 2 x 3 - Flat Stick Pack
- Connector: Standard Tamiya
- Designed for: RC Racing Vehicles.
- Fits: Associated HPI, LOSI, Traxxas, Tamiya, Kyosho and all other standard 7.2V cars
- Brand: Power Portable
- Included Qty: 1

Dimensions and Weight:

- Length: 5.35" (135 mm)
- Width: 1.81" (45 mm)
- Height: 0.91"(25 mm)
- Weight: 12.31 oz (349 g)

Wall Charger Specifications:

- Input: AC 110 V
- Output: 12V - 300mA

Charging Time:

7.2v 700 mAh Ni-Cd Pack - 2.5 Hours

BUTTONS

Figure: Start Stop push button

- Product Name : Push Button Switch : Fit Model : PPBB-30N;Material : Plastic, Electric Parts
- Ei : 600V;Ith : 10A;Light Rating Voltage : 220V - 240V AC
- Panel Cutout Diameter : 3cm / 1.2";Fit Panel Thickness(Adjustable) : Max 1.6cm /0.63"
- Total Size (Approx.) : 5.5 x 3.2 x 8cm / 2.2" x 1.3" x 3.1"(L*W*H);Color : As Picture Shown
- Weight : 89g;Package Content : 1 power Push Button Switch

Figure: Momentary push button

- Rated voltage & current: AC250V/3A AC125V/6A.Model:R13-507,
- Pin Quantity:2, Action:Momentary Lockless. Contact Type:SPST. Contact:1 NO(Normally Open).
- Switch height:24mm/0.95″.Mounting Thread Diameter:15.5mm / 0.61".Pre-soldered Wires length:12cm/4.7"
- Material:Plastic, Metal.The brass terminal has good conductivity and the plastic shell is environment-friendly and durable.

Gate Machine

DESIGN OVERVIEW

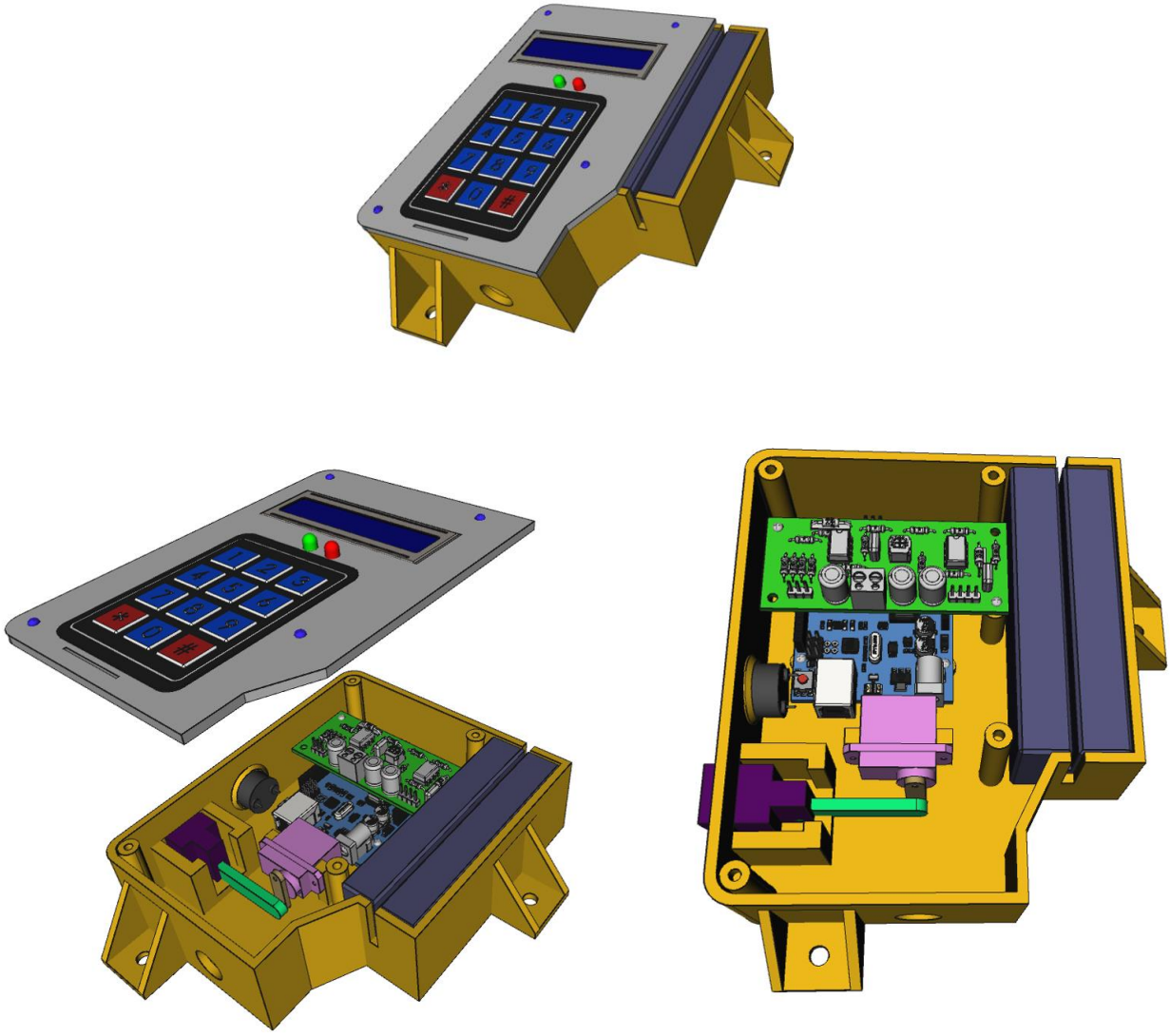
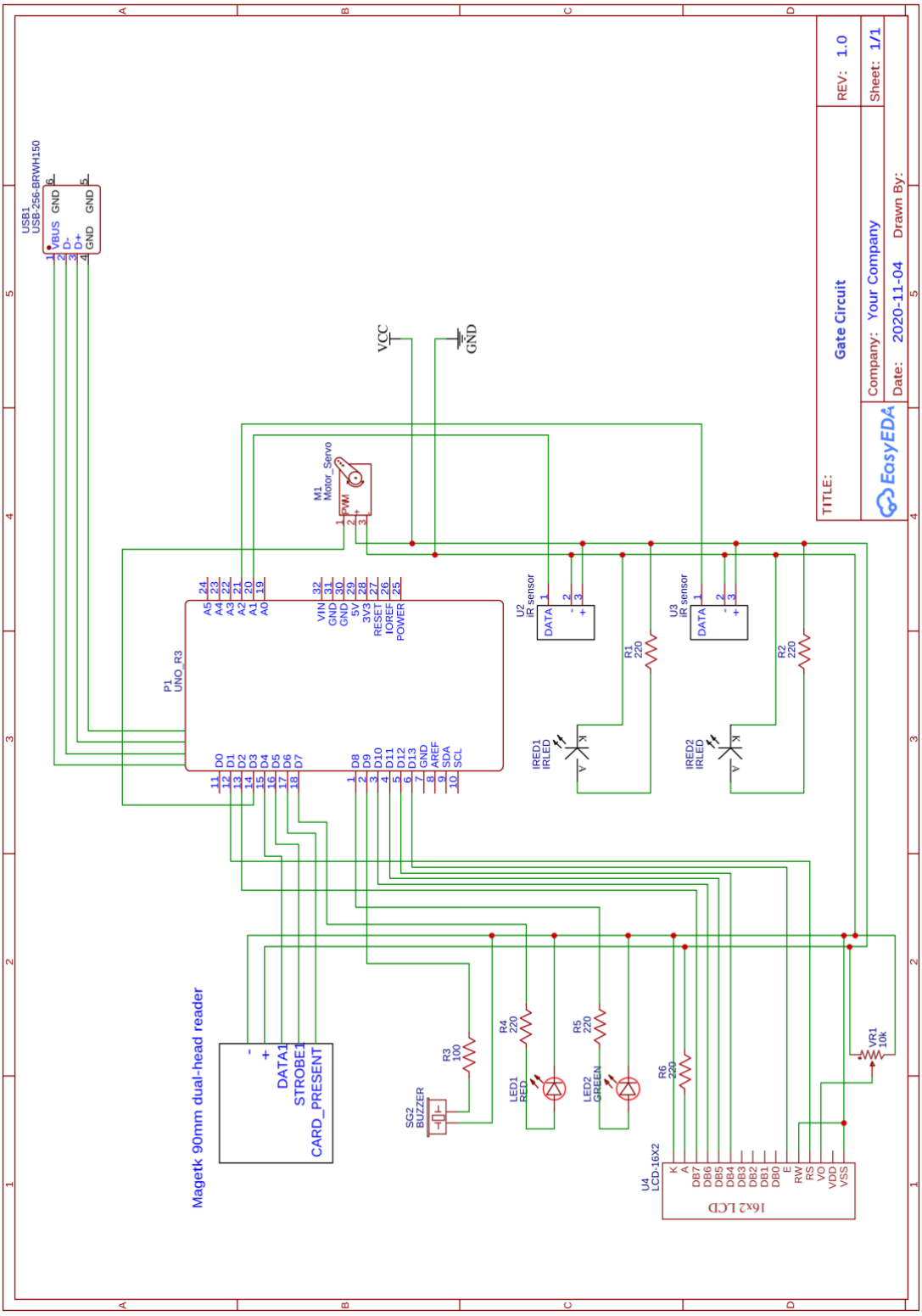


Figure: Design Overview

CIRCUIT DIAGRAM OF THE HAND HELD DEVICE IS AS FOLLOWS



ARDUINO UNO REV3

Arduino Uno is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started... You can tinker with your Uno without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

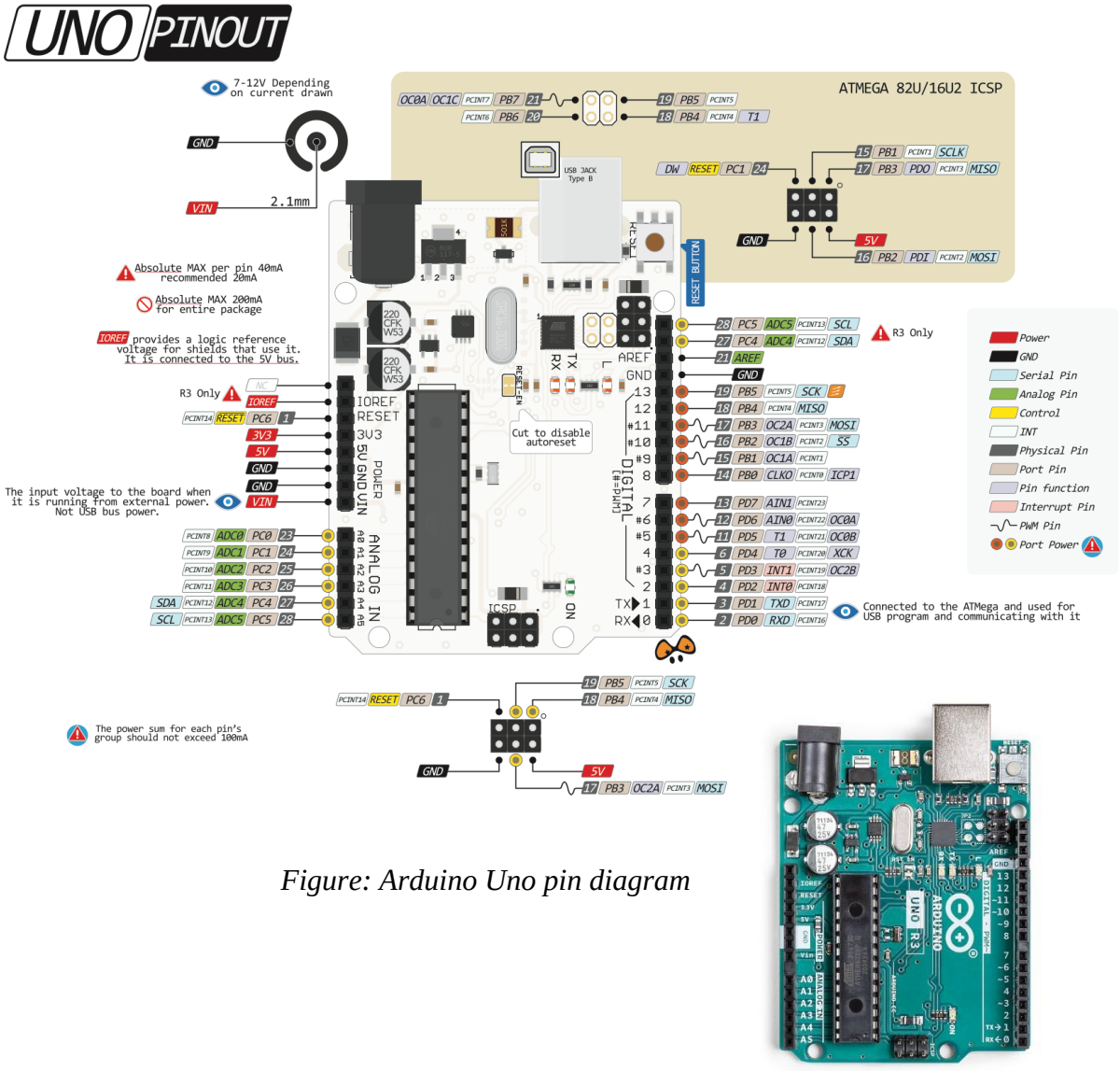


Figure: Arduino Uno pin diagram

More details: https://store.arduino.cc/usa/arduino-uno-rev3

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

MAGETK 90MM DUAL-HEAD READER

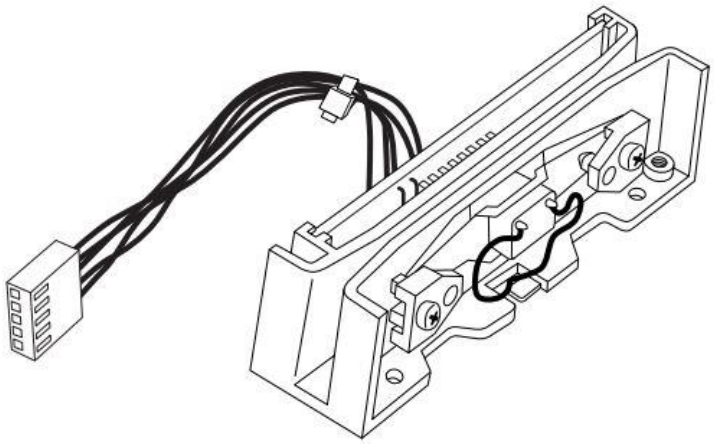


Figure: 90-millimeter Compatible Swipe Reader

Product Documentation:
https://www.magtek.com/content/documentationfiles/d99831083.pdf

Specification

IEC: Meets or Exceeds Requirements for:	IEC 1000-4-2 ESD (Electro Static Discharge) IEC 1000-4-3 Radiated EMC Field (2X requirement) IEC 1000-4-4 Electrical Fast Transient Burst requirement (transmission on I/O cable)
Flammability	Meets UL94V-0
Recording Method	Two-Frequency Coherent Phase (F2F)
Speed	Card speed through the unit may vary from: 2 to 125 in/s at 75 bpi (5.1 to 318 cm/s at 29.5 b/cm) 2 to 60 in/s at 210 bpi (5.1 to 152.4 cm/s at 82.7 b/cm)
Power Requirements	Single Track: 2.7 to 5.5VDC at 1mA, typical Dual Track: 2.7 to 5.5VDC at 2mA, typical 3 Tracks: 2.7 to 5.5VDC at 3mA, typical
Output Signal Levels	V_{ol} = 0.4V at 2mA V_{oh} = V_{cc} -0.5V at -2mA
Operating Temperature	-30°C to 70° C
Operating Humidity	10% to 90% relative humidity
Life	300,000 passes Single Track 1,000,000 passes Multi-Track
MTBF Electronics	125,000 h
Dimensions	Length: 3.54" (90.0mm) Height: 0.95" (24.13mm) Width: 0.88" (22.4mm)
Cable Length:	Single Track: 6" (150mm) Dual Track: 4" (101.6mm) 3 Track 5" (127mm)
Connector	See Section 2, Connectors
Colors available	Black, Standard

ARDUINO SERVO MOTOR

Figure: Arduino servo motor

Specifications:

Operating Voltage is +5V typically.

Torque: 2.5kg/cm.

Operating speed is 0.1s/60°

Gear Type: Plastic.

Rotation: 0°-180°

Weight of **motor**: 9gm.

Package includes gear horns and screws.

How to Connect

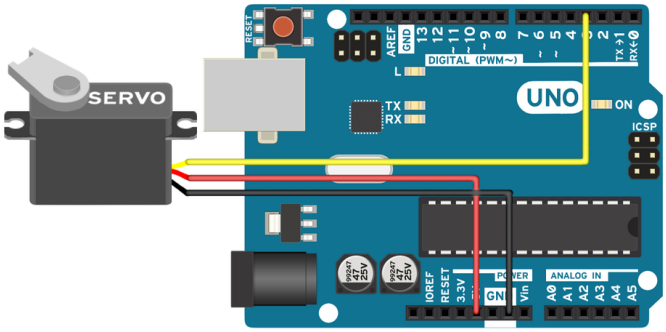


Figure: How to connect servo motor

A servo motor has everything built in: a motor, a feedback circuit, and most important, a motor driver. It just needs one power line, one ground, and one control pin.

Following are the steps to connect a servo motor to the Arduino:

1. The servo motor has a female connector with three pins. The darkest or even black one is usually the ground. Connect this to the Arduino GND.

2. Connect the power cable that in all standards should be red to 5V on the Arduino.
3. Connect the remaining line on the servo connector to a digital pin on the Arduino.

Code

```
// Include the Servo library
#include <Servo.h>

// Declare the Servo pin

int servoPin = 3;

// Create a servo object

Servo Servo1;

void setup() {

   // We need to attach the servo to the used pin number

   Servo1.attach(servoPin);

}

void loop(){

   // Make servo go to 0 degrees

   Servo1.write(0);

   delay(1000);

   // Make servo go to 90 degrees

   Servo1.write(90);

   delay(1000);

   // Make servo go to 180 degrees

   Servo1.write(180);

   delay(1000);

}
```

If the servo motor is connected on another digital pin, simply change the value of servoPin to the value of the digital pin that has been used.

How it works

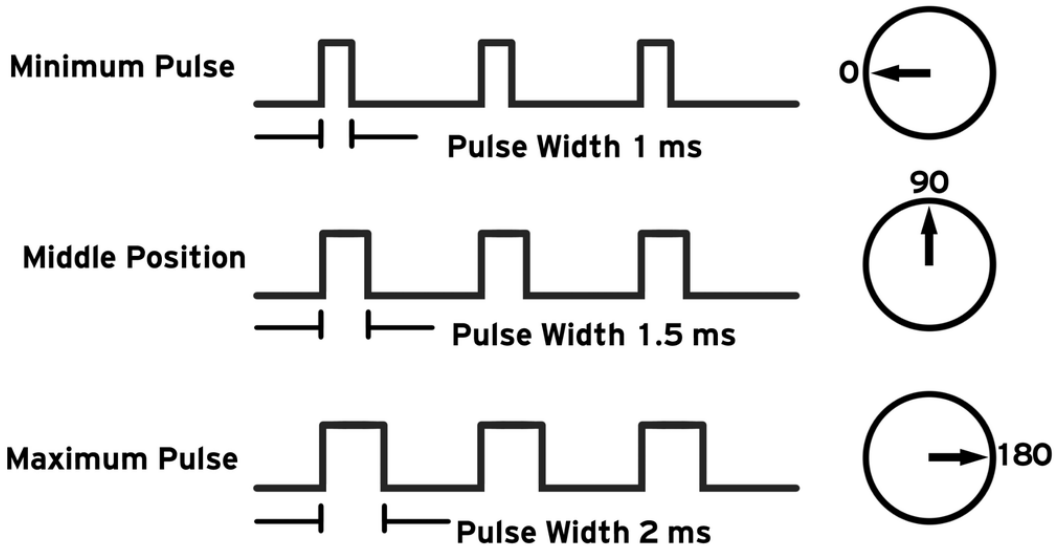


Figure: How it works

BUZZER

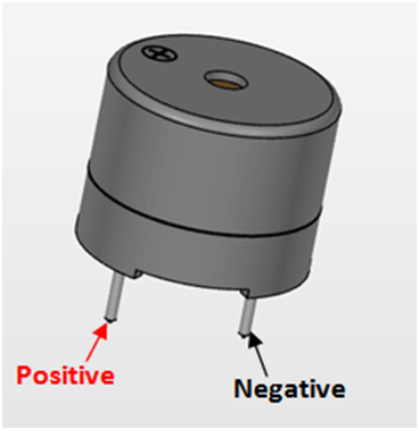


Figure: Buzzer

Buzzer Pin Configuration

Pin Number	Pin Name	Description
1	Positive	Identified by (+) symbol or longer terminal lead. Can be powered by 6V DC
2	Negative	Identified by short terminal lead. Typically connected to the ground of the circuit

Buzzer Features and Specifications

- Rated Voltage: 6V DC
- Operating Voltage: 4-8V DC
- Rated current: <30mA
- Sound Type: Continuous Beep
- Resonant Frequency: ~2300 Hz
- Small and neat sealed package
- Breadboard and Perf board friendly

How to use a Buzzer

A **buzzer** is a small yet efficient component to add sound features to our project/system. It is very small and compact 2-pin structure hence can be easily used on breadboard, Perf Board and even on PCBs which makes this a widely used component in most electronic applications.

There are two types are buzzers that are commonly available. The one shown here is a simple buzzer which when powered will make a Continuous Beeeeeeppp.... sound, the other type is called a readymade buzzer which will look bulkier than this and will produce a Beep. Beep. Beep. Sound due to the internal oscillating circuit present inside it. But, the one shown here is most widely used because it can be customised with help of other circuits to fit easily in our application.

This buzzer can be used by simply powering it using a DC power supply ranging from 4V to 9V. A simple 9V battery can also be used, but it is recommended to use a regulated +5V

or +6V DC supply. The buzzer is normally associated with a switching circuit to turn ON or turn OFF the buzzer at required time and require interval.

16X2 LCD DISPLAY

Product Documentation: https://components101.com/displays/16x2-lcd-pinout-datasheet

Figure: 16x2 lcd display pin diagram

Pin No:	*Pin Name:*	*Description*
1	Vss (Ground)	Ground pin connected to system ground
2	Vdd (+5 Volt)	Powers the LCD with +5V (4.7V – 5.3V)
3	VE (Contrast V)	Decides the contrast level of display. Grounded to get maximum contrast.
4	Register Select	Connected to Microcontroller to shift between command/data register
5	Read/Write	Used to read or write data. Normally grounded to write data to LCD
6	Enable	Connected to Microcontroller Pin and toggled between 1 and 0 for data acknowledgement
7	Data Pin 0	Data pins 0 to 7 forms a 8-bit data line. They can be connected to Microcontroller to send 8-bit data. These LCD's can also operate on 4-bit mode in such case Data pin 4,5,6 and 7 will be left free.
8	Data Pin 1	
9	Data Pin 2	
10	Data Pin 3	
11	Data Pin 4	
12	Data Pin 5	
13	Data Pin 6	
14	Data Pin 7	
15	LED Positive	Backlight LED pin positive terminal

16	LED Negative	Backlight LED pin negative terminal

Features

- Operating Voltage is 4.7V to 5.3V
- Current consumption is 1mA without backlight
- Alphanumeric LCD display module, meaning can display alphabets and numbers
- Consists of two rows and each row can print 16 characters.
- Each character is built by a 5×8 pixel box
- Can work on both 8-bit and 4-bit mode
- It can also display any custom generated characters
- Available in Green and Blue Backlight

IR SENSOR

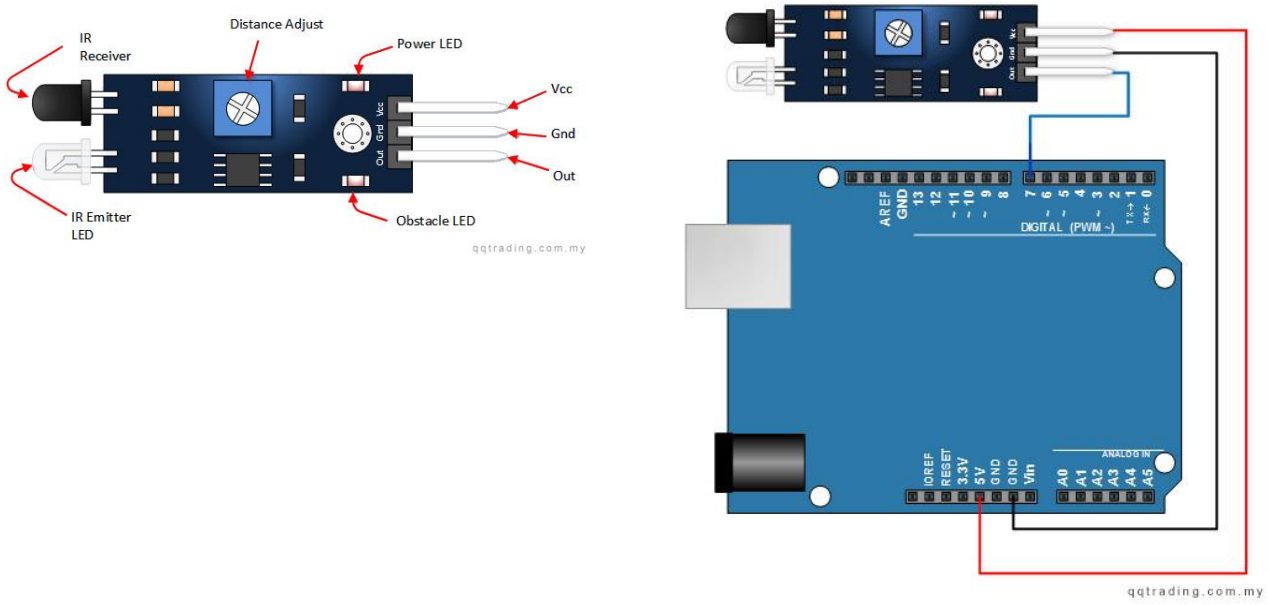


Figure: Overview

Pin, Control Indicator	*Description*
Vcc	3.3 to 5 Vdc Supply Input
Gnd	Ground Input
Out	Output that goes low when obstacle is in range
Power LED	Illuminates when power is applied
Obstacle LED	Illuminates when obstacle is detected
Distance Adjust	Adjust detection distance. CCW decreases distance. CW increases distance.
IR Emitter	Infrared emitter LED
IR Receiver	Infrared receiver that receives signal transmitted by Infrared emitter.

Code

```
// IR Obstacle Collision Detection Module


int LED = 13; // Use the onboard Uno LED
int isObstaclePin = 7; // This is our input pin
int isObstacle = HIGH; // HIGH MEANS NO OBSTACLE


void setup() {
    pinMode(LED, OUTPUT);
    pinMode(isObstaclePin, INPUT);
    Serial.begin(9600);
}


void loop() {
    isObstacle = digitalRead(isObstaclePin);
    if (isObstacle == LOW) {
        Serial.println("OBSTACLE!!, OBSTACLE!!");
        digitalWrite(LED, HIGH);
    } else {
        Serial.println("clear");
```

```
        digitalWrite(LED, LOW);
    }
    delay(200);
}
```

ID LED

Figure: IR led

Pin Configuration

The IR LED or Infrared LED has polarity i.e.it has a positive and negative pin. The pin which is long is the positive pin (anode) and the pin which is short is the negative pin (cathode) as shown in the above **IR LED pinout**.

Technical Specifications

- Forward current (IF) is 100mA (normal condition) and 300mA (max.)
- 1.5A of surge forward current
- 1.24v to 1.4v of forward voltage
- Temperature for storage and operation varies from -40 to 100 ℃
- Soldering Temperature should not exceed 260 ℃
- Power Dissipation of 150mW at 25℃ (free air temperature) or below

- Spectral bandwidth of 45nm
- Viewing angle is 30 to 40 degree

How to use?

The most common use of this LED is in IR sensor, with companion to IR receiver. An IR sensor works as it sends IR signal through the IR transmitter and receives through IR receiver. If we placed an object near to the IR sensor the LED connected to the sensor goes high.

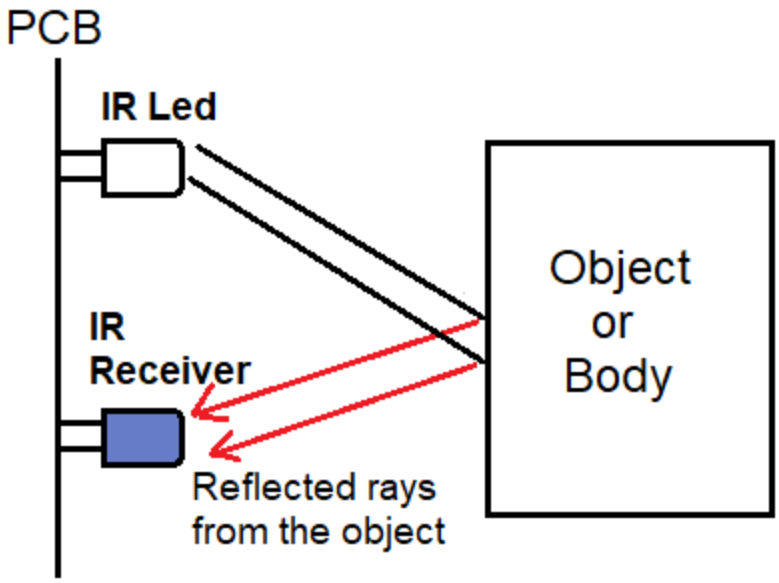


Figure: How to use

3×4 KEYPAD

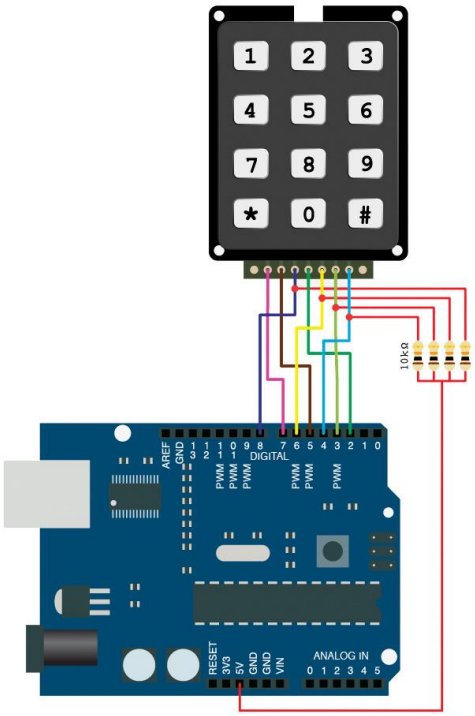


Figure: Keypad Circuit

To wire your 3×4 Keypad to your Arduino, connect the following pins:

Keypad Pin 1 –> Arduino Pin 7
Keypad Pin 2 –> Arduino Pin 5
Keypad Pin 3 –> Arduino Pin 8
Keypad Pin 4 –> Arduino Pin 2
Keypad Pin 5 –> Arduino Pin 6
Keypad Pin 6 –> Arduino Pin 3
Keypad Pin 7 –> Arduino Pin 4

Instructions and Code

```
#include
<Keypad.h>

const byte ROWS = 4; // four rows
const byte COLS = 3; // three columns


char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
```

```
  {'7','8','9'},
  {'*','0','#'}
};


byte rowPins[ROWS] = {5, 4, 3, 2}; // connect to the row pinouts of the keypad
byte colPins[COLS] = {8, 7, 6};    // connect to the column pinouts of the keypad


Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );


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

void loop() {
  char key = keypad.getKey();

  if (key){
    Serial.println(key);
  }
}
```

Software Design

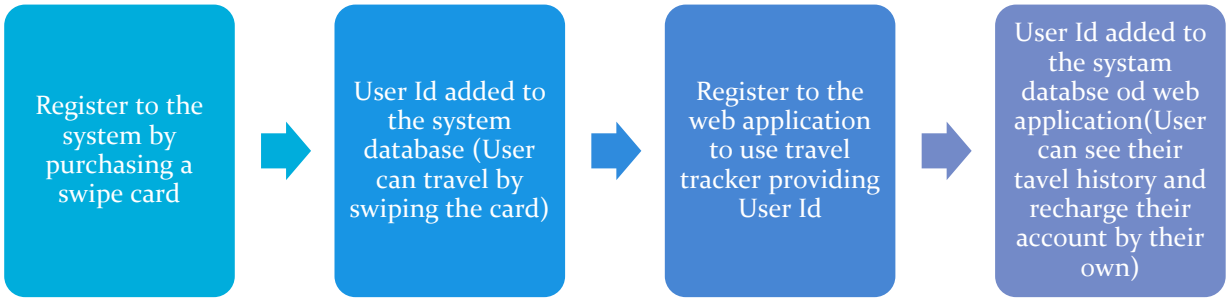


Figure: Software solution summery

ER Diagram

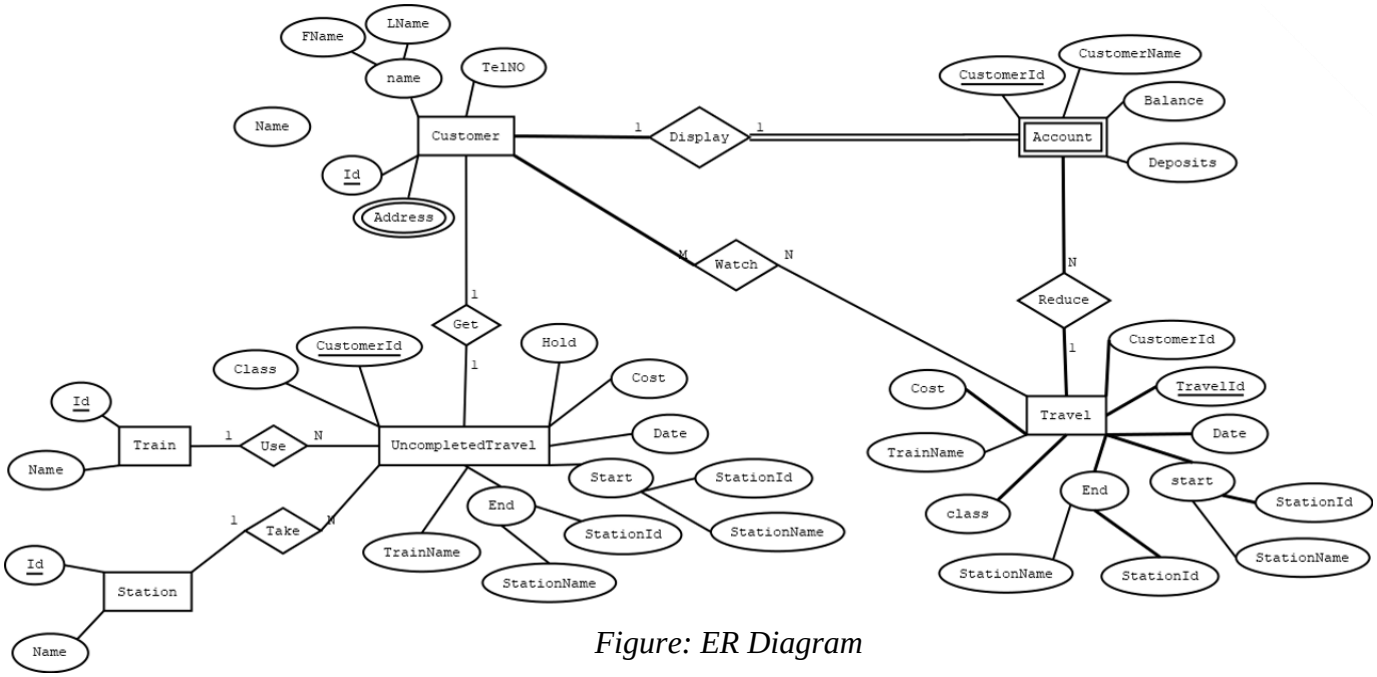


Figure: ER Diagram

Above figure shows the relationships between tables in the SQL database. But our solution for the data base is a NO SQL cloud database.

Flow Chart

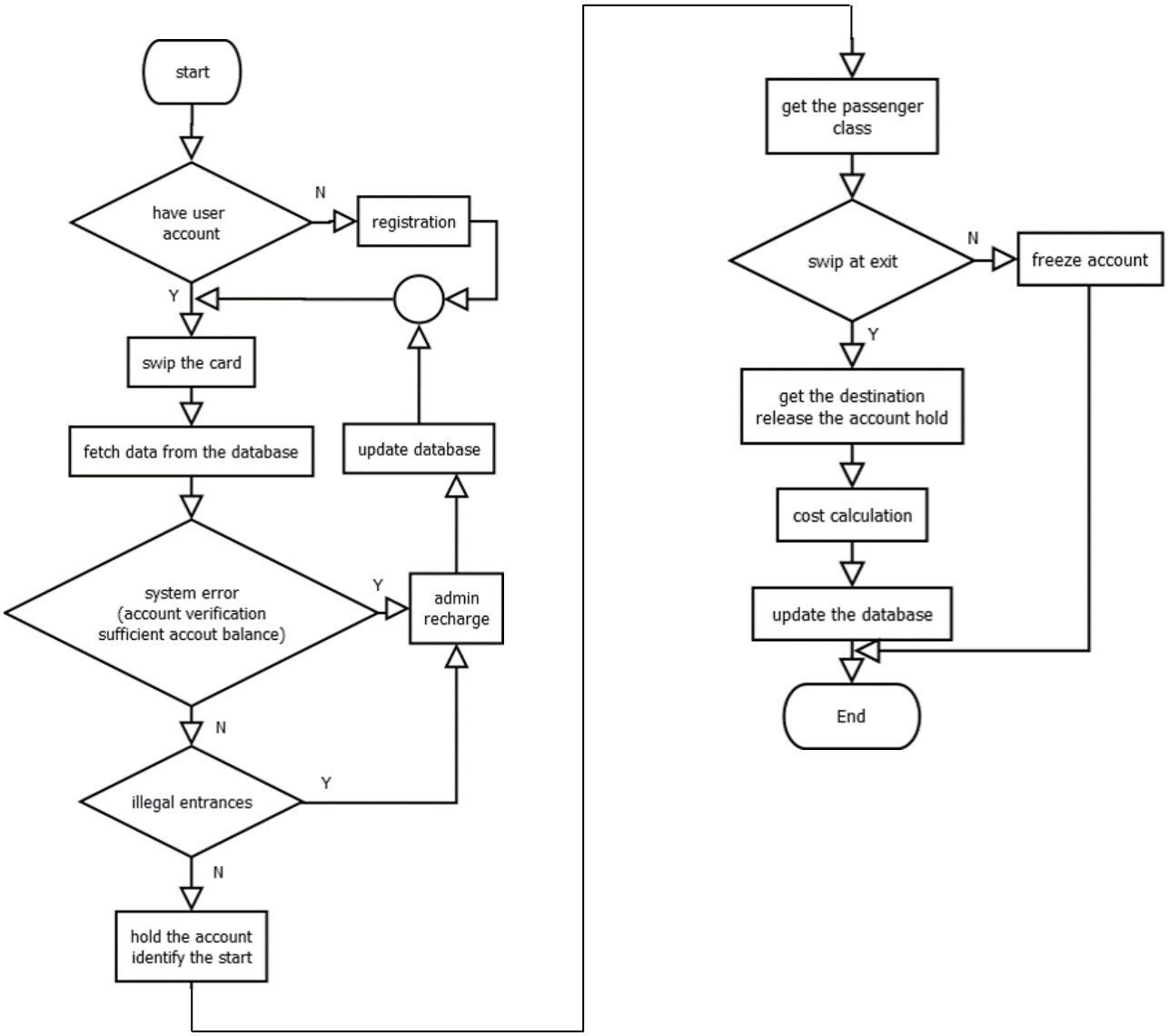


Figure: Flow Chart of the System

NETWORK TECHNOLOGIES & PROTOCOLS

- After swiping the card send client id over HTTPS
- Identify the user & verify according to the account balance
- Post JSON objects over HTTPS from java application in PC at station.
- Send data over HTTPS by handheld device using JSON objects
- For the website security we use username & password authentication. Passwords are encrypted using password hashing.

Station PC Controller

Used Libraries

- **ApacheHttpClient**: The library used for sending http requests to the server.
- **Jason**: Jason library is used to format Jason string objects and objects to Jason strings.
- **JavaFx**: For GUI

User Configuration

- **Station name**: Name of the station that the software has been installed.(Galle, Colombo fort,…etc.)
- **Station ID**: A unique station id is provided for each station when configuring.
- **Auth token**: The authentication token is a unique string that is used as the verification method when sending requests to the server. And it unique for each station.

Server Connection and Requests

A TCP connection will be established with the server only while sending a request it will be closed after receiving a respond. The current API is *"https://rts-railway.herokuapp.com/api/"*. Following requests and responds will be send and received.

- **Notify a swipe at the entrance gate:**
  - Request
    - URL = *https://rts-railway.herokuapp.com/api/uncom/start/:userID*
    - content = {S_Station_id : station_id}
  - Respond
    - code = 200 if ok, other if there any error.
    - content = {msg: Error if there any}

- **Notify a swipe at the exit gate:**
  - Request
    - URL = *https://rts-railway.herokuapp.com/api/uncom/end/:userID*
    - content = {E_Station_id : station_id}
  - Respond
    - code = 200 if ok, other if there any error.
    - content = {msg: Error if there any}

- **User configuration verification:** (To check if the values of user configuration are valid)
  - Request
    - URL = *https://rts-railway.herokuapp.com/api/userconfig*
    - content = {Station_name; station_name, Station_id: station_id}
  - Respond
    - code = 200 if ok, other if there any error.
    - content = {msg: Error if there any}

Web Application

DEVELOPMENT

Back End – NODEJS Express

Front End – React

Database – MongoDB Atlas

Access and authentication

- After swiping the card send client id over HTTPS
- Identify the user & verify according to the account balance
- Post JSON objects over HTTPS from java application in PC at station.
- Send data over HTTPS by handheld device using JSON objects
- For the website security we use username & password authentication. Passwords are encrypted using SH2.
- Login/ Logout handled by JSON web tokens.

DEPENDENCIES FOR WEB APPLICATION

Back End

```
"dependencies": {
        "bcrypt": "^5.0.0",
        "cloudinary": "^1.23.0",
        "concurrently": "^5.3.0",
        "cookie-parser": "^1.4.5",
        "cors": "^2.8.5",
        "dotenv": "^8.2.0",
        "express": "^4.17.1",
        "express-fileupload": "^1.2.0",
        "jsonwebtoken": "^8.5.1",
```

```
    "mongoose": "^5.10.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.4"
  }
```

Front End

```
"dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.5.0",
    "@testing-library/user-event": "^7.2.1",
    "axios": "^0.20.0",
    "bootstrap": "^4.5.3",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-owl-carousel": "^2.3.1",
    "react-paypal-express-checkout": "^1.0.5",
    "react-router-dom": "^5.2.0",
    "react-scripts": "3.4.3"
  },
```

Database

- **Customer Model**
  - To maintain details about customers who registered to traveling system by purchasing a swipe card

```
Id:{
        type: String,
        required: true,
        unique: true
    },
    name:{
        type: String,
        required: true
    },
    phone:{
        type: String
    },
    address1:{
        type: String,
```

```
    },
    address2:{
        type: String,
    },
    address3:{
        type: String,
    },
    balance:{
        type: Number,
        default: 0
    },
    deposit:{
        type: Number,
        default: 0
    }
},{
    timestamps: true
}
```

- Payment Model
  - To maintain travel costs for 3 classes (1^{st}, 2^{nd}, 3^{rd})

```
Id:{
        type: String,
        required: true,
        unique: true
    },
    first:{
        type: Number,
        required: true
    },
    second:{
        type: Number,
        required: true
    },
    third:{
        type: Number,
        required: true
    }
},{
    timestamps: true
}
```

- Station Model
  - To maintain details of Stations

```
Id:{
        type: String,
        required: true,
        unique: true
    },
    name:{
        type: String,
        required: true,
        trim: true,
        unique: true
    }
}, {
    timestamps: true
}
```

- Train Model
  - To maintain details of Trains

```
Id:{
        type: String,
        required: true,
        unique: true
    },
    name:{
        type: String,
        required: true,
        trim: true,
        unique: true
    }
}, {
    timestamps: true
}
```

- Travel Model
  - The model to store all the completed travels of customers

```
UserId:{
        type: String,
        required: true
    },
    S_StationId:{
        type: String,
        required: true
    },
    S_StationName:{
        type: String,
        required: true
    },
    class:{
        type: Number,
        default: 3
    },
    TrainId: {
        type: String
    },
    Train:{
        type: String
    },
    E_StationId:{
        type: String
    },
    E_StationName:{
        type: String
    },
    cost:{
        type: Number,
        default: 0
    }
},{
    timestamps: true
}
```

- Uncompleted Travel Model
  - To maintain travel details until a customer complete his/her travel.
  - If his/her account balance is not sufficient to pay the travel cost, that account will be blocked and all data will be stay here as it is.

```
UserId:{
        type: String,
        required: true,
        unique: true
    },
    S_StationId:{
        type: String,
        required: true
    },
    S_StationName:{
        type: String,
        required: true
    },
    class:{
        type: Number,
        default: 3
    },
    TrainId: {
        type: String,
        default: 'n/a'
    },
    Train:{
        type: String,
        default: 'n/a'
    },
    E_StationId:{
        type: String,
        default: 'n/a'
    },
    E_StationName:{
        type: String,
        default: 'n/a'
    },
    cost:{
        type: Number,
        default: 0
    }
},{
```

```
    timestamps: true
}
```

- User Model
  - To maintain user data of the web application users

```
Id:{
        type: String,
        required: true,
        unique: true
    },
    password:{
        type: String,
        required: true
    },
    role:{
        type: Number,
        default: 0
    }
},{
    timestamps: true
}
```

Tokens

- Access token & Refresh token combination handle the user authentication & Authorization.
- When he logs in, a refresh token will be stored as a cookie. He can get the access token by passing that cookie with the request.

```
// If login success, create access token and refresh token
            const accesstoken = createAccessToken({id: user._id})
            const refreshtoken = createRefreshToken({id: user._id})

            res.cookie('refreshtoken', refreshtoken, {
                httpOnly: true,
                path: '/user/refresh_token',
                maxAge: 7*24*60*60*1000 // 7d
            })

            res.json({accesstoken});
```

```
const createAccessToken = (user) =>{
    return jwt.sign(user, process.env.ACCESS_TOKEN_SECRET, {expiresIn: '1d
'})
}

const createRefreshToken = (user) =>{
    return jwt.sign(user, process.env.REFRESH_TOKEN_SECRET, {expiresIn: '7
d'})
}
```

- When user Logout, the cookie will be cleared.

```
res.clearCookie('refreshtoken', {path: '/user/refresh_token'})
```

- Password encryption method used to protect user passwords.

```
// Password Encryption
            const passwordHash = await bcrypt.hash(password, 10)
```

Middleware

To check whether receiving request is a valid request or not, two middleware codes used.

1. To check the receiving request is valid

```
const auth = (req, res, next) => {
    try {
       const token = req.header("Authorization")
       if(!token) return res.status(400).json({msg: "Invalid authenticatio
n"})

       jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, user) =>{
           if(err) return res.status(400).json({msg: "Invalid authenticati
on"})

           req.user = user
           next()
       })
```

```
    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
}
```

2. To check the receiving request is from an admin

- In the user details model there is a property called “role”. If it is equal 1, he/she is an admin, otherwise no.

```
const authAdmin = async (req, res, next) => {
    try {
        // Get user information by id
        const user = await Users.findOne({
            _id: req.user.id
        })
        if(user.role === 0)
            return res.status(400).json({msg: "Admin resources access denied"})

        next()

    } catch (err) {
        return res.status(500).json({msg: err.message})
    }
}
```

Source Code

https://github.com/cepdnaclk/e16-3yp-automated-railway-ticketing-system/tree/main/Web

Deployment

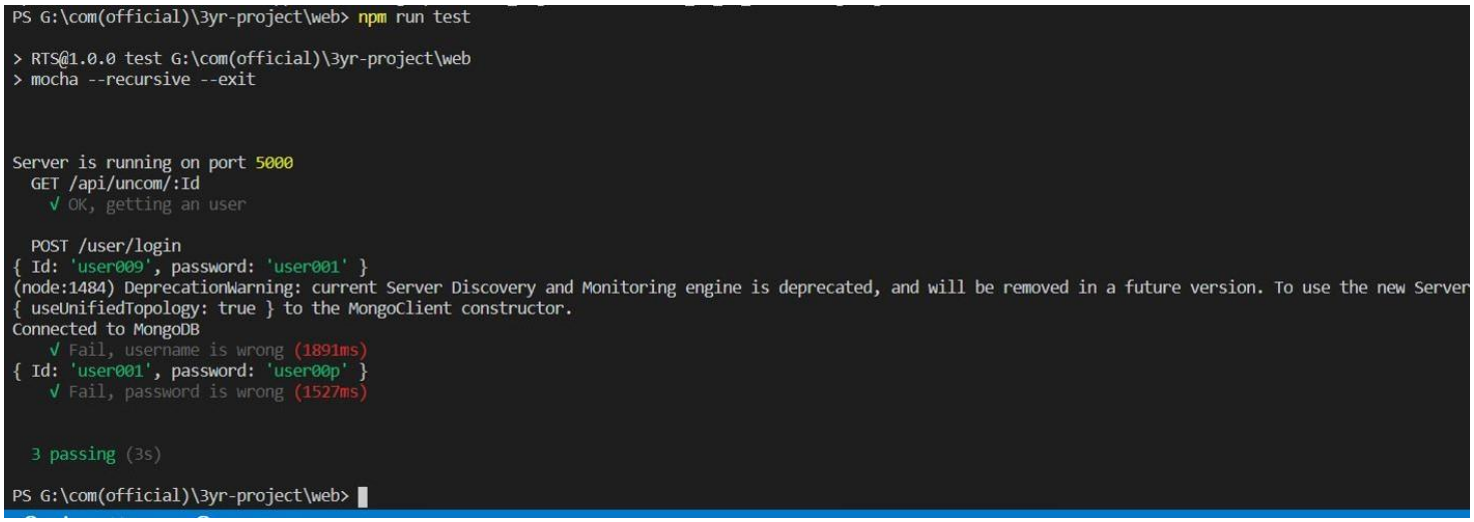
https://rts-railway.herokuapp.com/

TEST RESULTS

What was tested?

- Several main functionalities of the web application was tested.

Test Results



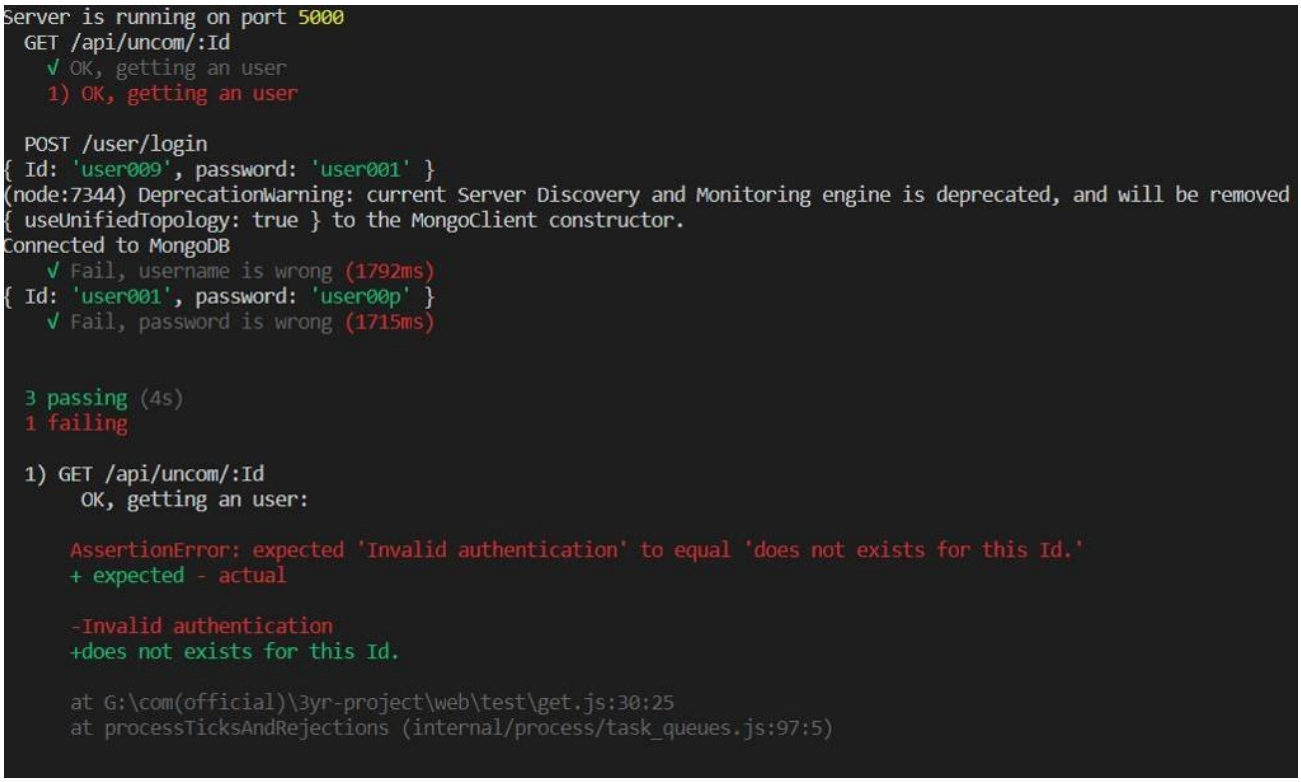


Figure: Test Results

Test Number	*Test Type*	*What was tested*	*Importance*	*Way of test done*	*results and findings*
1	Unit testing	Middleware: Post method for user logging Correctness of the user name was tested	Someone can enter any password and can enter to the system using several bruteforce checkings. Our system always gives security for the incorrect user names.	Using mocha,chai,supertest Tools Incorrect user names were given as json objects and system gives correct outputs as we expected	What we expected was given and sometimes expectation and output was differ and using those tools able to identify the code statements needed to be modified.

2	Unit testing	Middleware: Post method for user logging Correctness of the user password was tested	Using bruteforcing someone can enter the correct password. System works corrently for the password issues.	Using mocha,chai,supertest Tools Incorrect passwords were given as json objects and system gives correct outputs as we expected. Here password formats and other types were changed and tested and system worked correctly.	What we expected was given and sometimes expectation and output was differ and using those tools able to identify the code statements needed to be modified.
3	Unit testing	Middleware: Get method for admin authentication	By changing admin usernames and passwords System gave authentication fails warning correctly	Using mocha,chai,supertest Tools Incorrect passwords and admin names were given as json objects and system gives correct outputs as we expected. Here password formats and other types were changed and tested and system worked correctly.	What we expected was given and sometimes expectation and output was differ and using those tools able to identify the code statements needed to be modified.
4	Unit testing	Middleware: Get method for user details cheking after logged as an admin	Giving different usernames try to get user details. Have to have authenticate the admin correctly otherwise this didn’t work. Have to have correct tokens	Try to acces user details without logging as admin system gives authentication fails warnings. Tried different tokens gave authentication fails	Expected was given and sometimes expectation and output was differ and using those tools able to identify the code statements needed to be modified
5	Unit testing	Middleware: Post method for user	System generate unique tokens	Change the tokens and try different aspects in	Gives authentication fails warnings

		logging Check correctness of tokens	and it is the one responsible for activities done on the server after logging. If someone can cheat then security fails.	the web and try do things inside the web	and some functionalities were accessed so were able to identify them