**Smart Parking System using IOT**

**Phase 5: Project Documentation & Submission**

**Project’s Objective:**

The objective of this project is to design, develop, and implement an innovative and efficient parking system. The primary aim is to reduce the need for human intervention, allowing technology to take over and manage the parking process autonomously. However, it's important to note that the system still requires technical human resources to ensure its proper functioning and maintenance.

The project features a mobile application that provides real-time information about parking slot availability. This data is sourced from Firebase Cloud Service, where it is continuously updated and managed by an ESP32 microcontroller. The IoT components, integral to the system's operation, are also controlled and managed by the same microcontroller.

One of the key functionalities of this system is allowing individuals to park their cars and leave without the need for manual intervention. The system calculates parking charges based on the duration the car has been parked, providing a seamless and convenient experience for users while streamlining the payment process.

Technologies used:

* Internet of Things
* Cloud service
* Mobile application
* Python

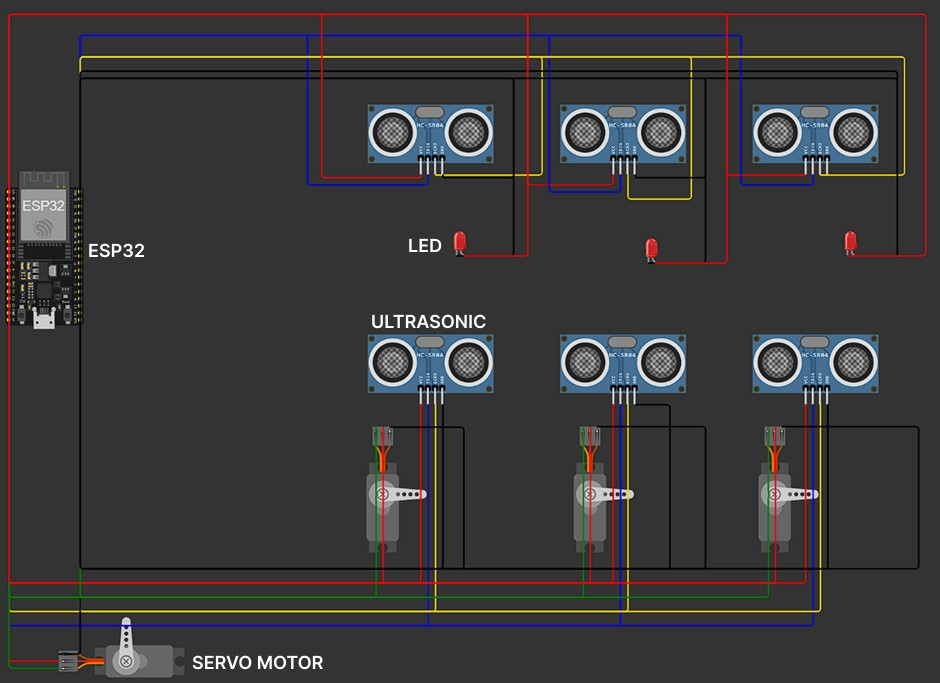
**IOT device setup:**

Components used:

* ESP32
* Led
* Ultrasonic Sensor
* Servo motor
* Wires

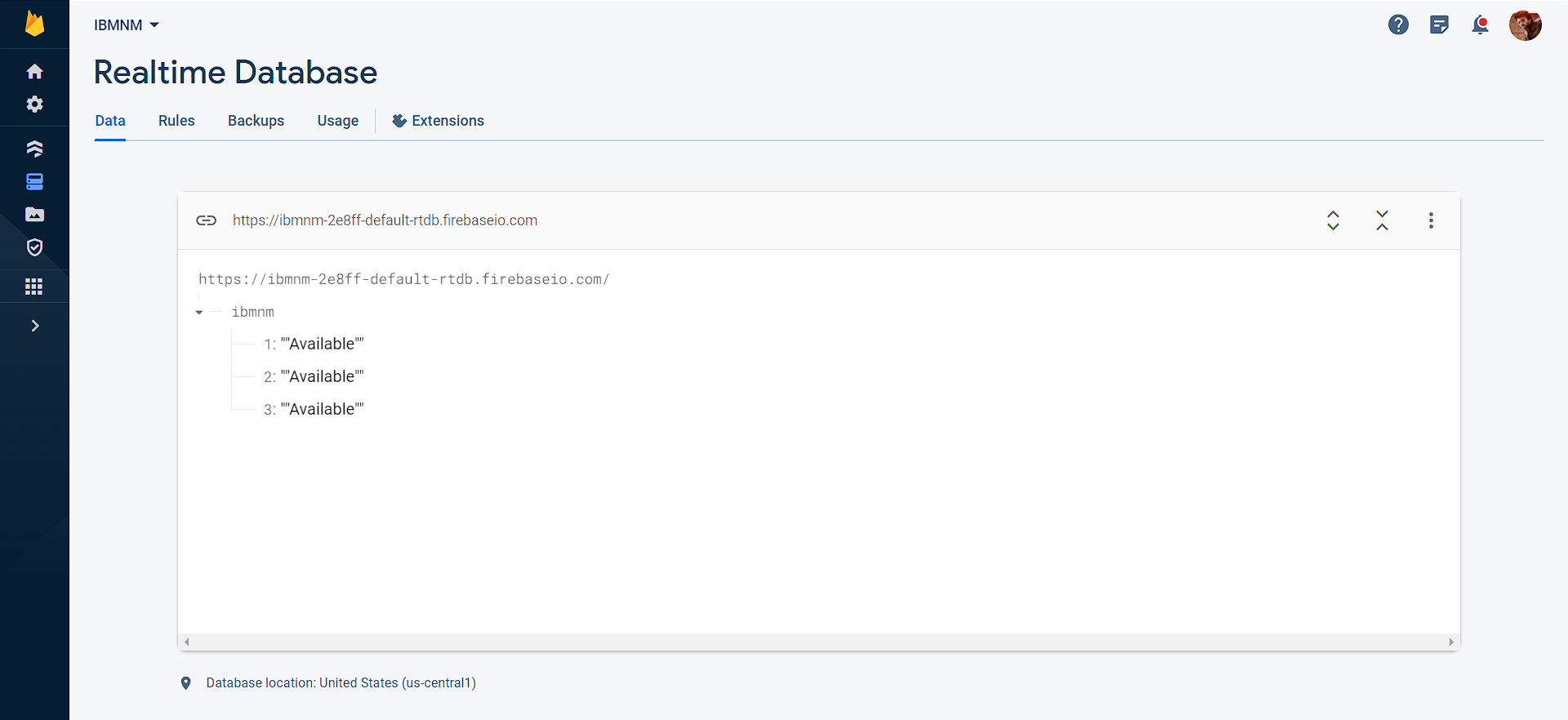
Working:

* When a car comes in the ultrasonic detects the change in the area, then the led turns on and the servo motor acts as a toll which will close. After then the firebase is updated that this slot is occupied.
* When the car leaves it will the ESP32 will update the firebase that the slot is available.
* If the payment is successful the servo motor will open the car can leave.



**Firebase:**

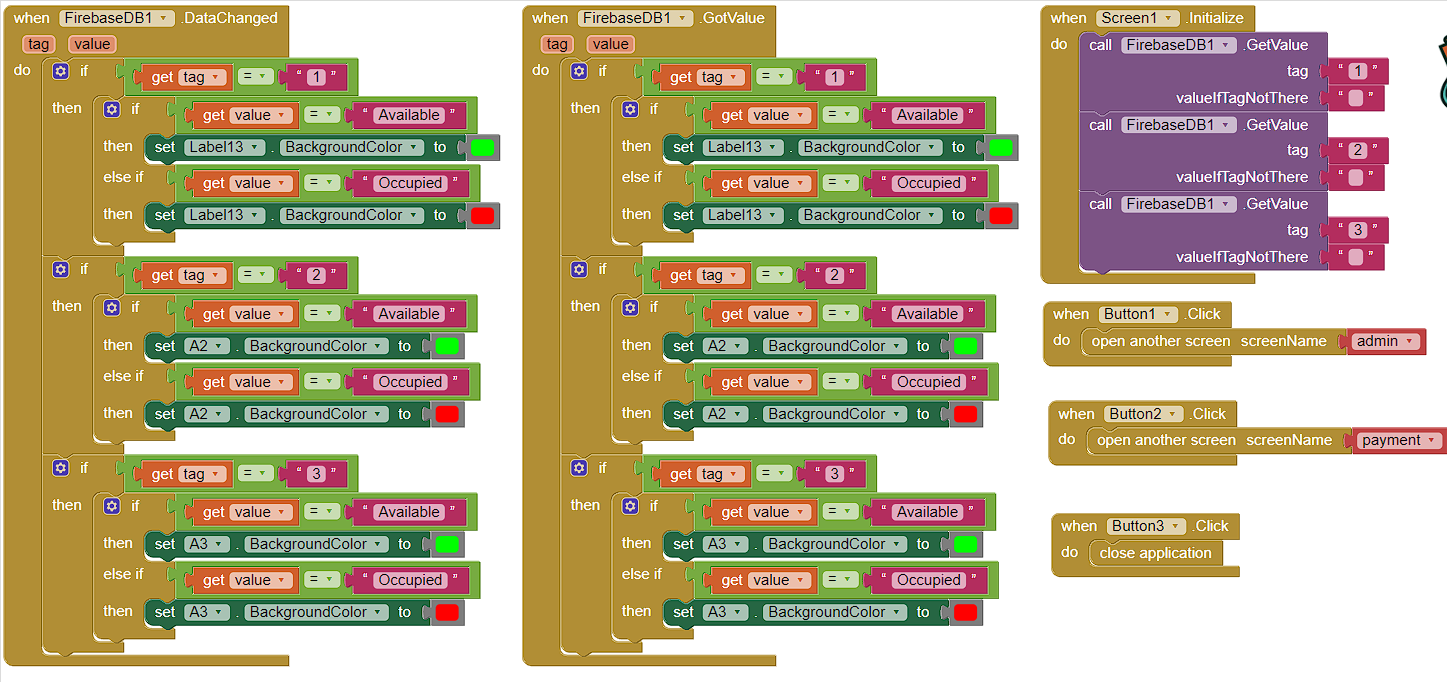
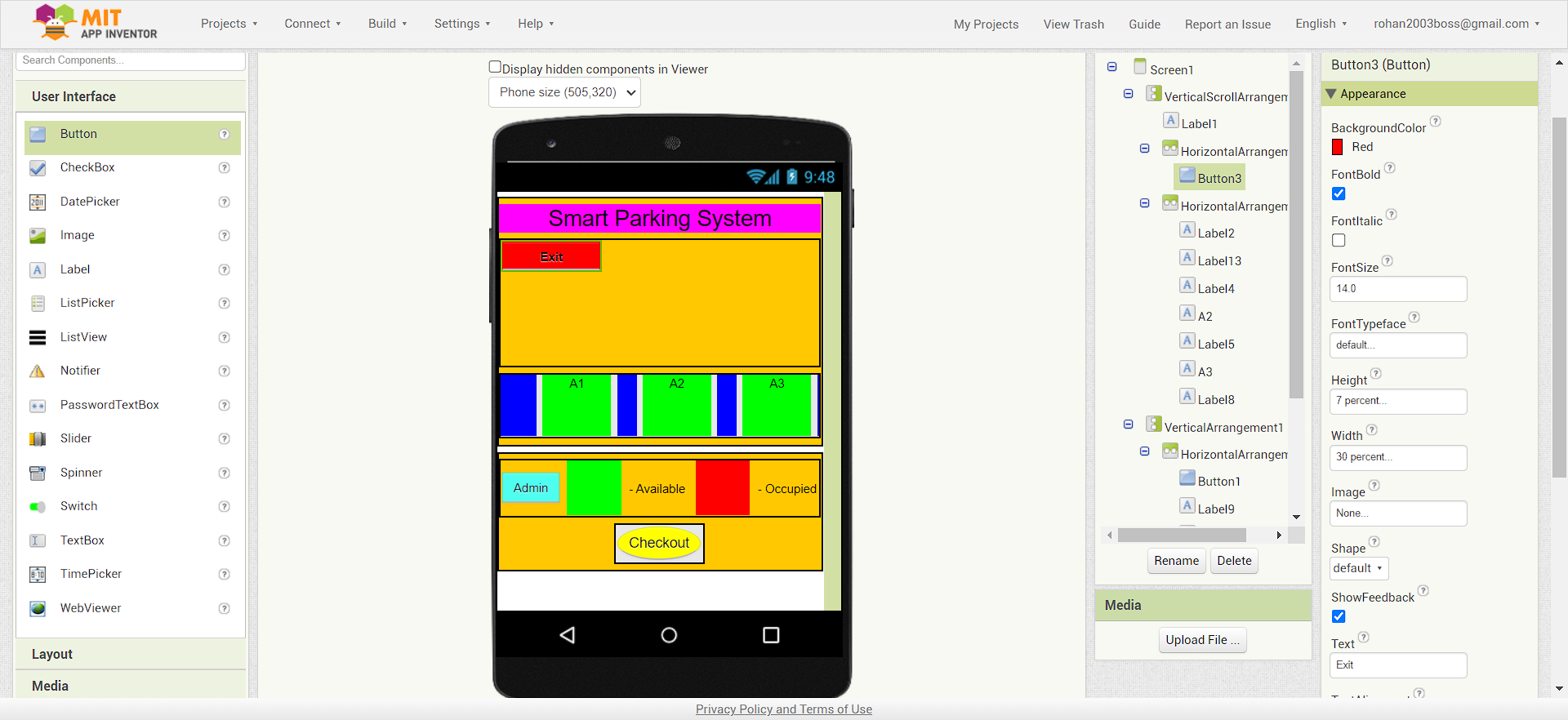
* The firebase stores the as key values datatype.
* The key is the slot number and the value are the availability of the slot.
* Whenever a car enters the parking slot the value will be changed to availability and whenever the car exits the availability changes to occupied.

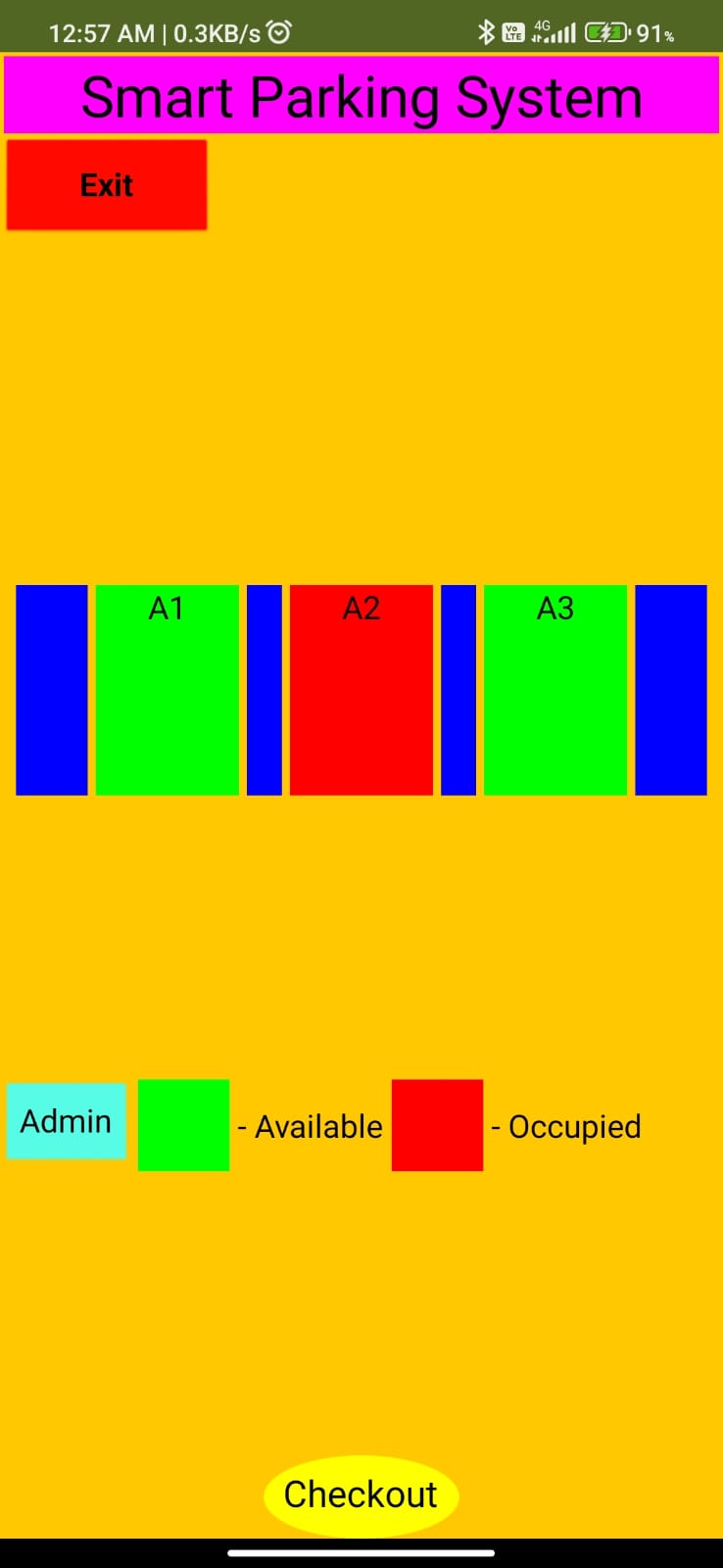
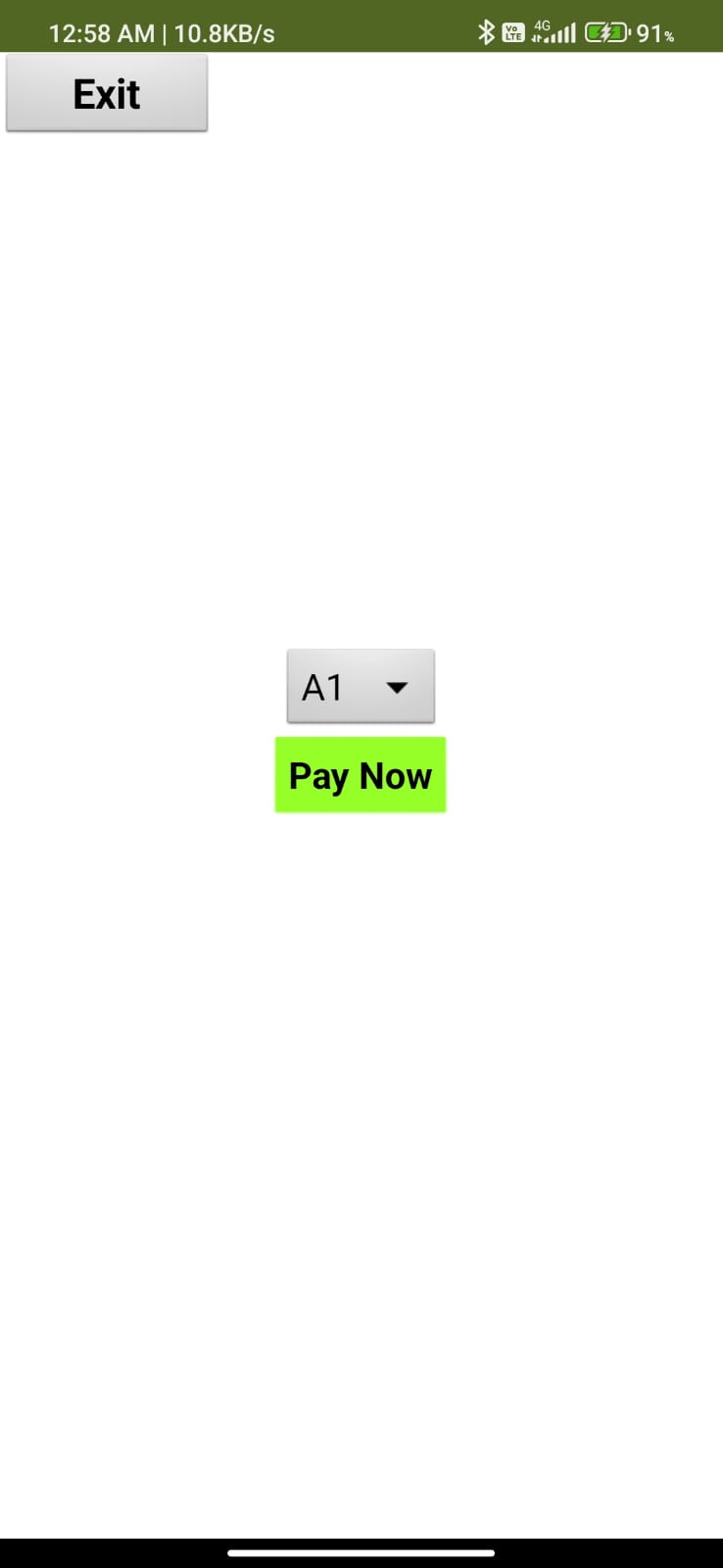


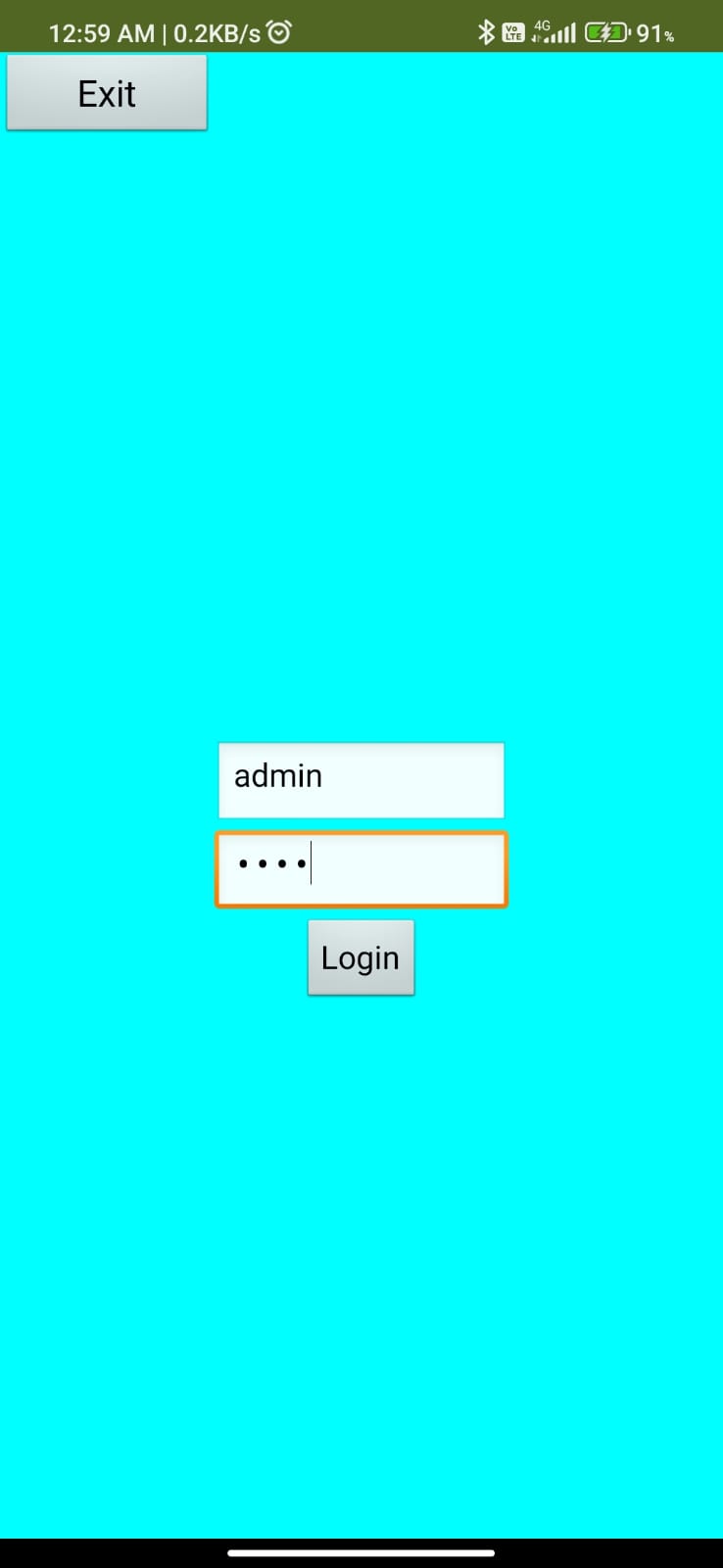
**Mobile Application:**

* The mobile application, developed using MIT App Inventor, provides a user-friendly interface for the smart parking system. It visually represents parking slots on its home screen, with slots changing to red when occupied and green when available.
* Users can select their parking slot within the application to pay parking charges and trigger the opening of the toll gate (servo motor) for a smooth exit.
* In-charge personnel have admin login access to manually operate slots, making it convenient for users who prefer manual payment methods or encounter payment issues.
* The credential for admin is:
  + Username: admin
  + Password: 1234

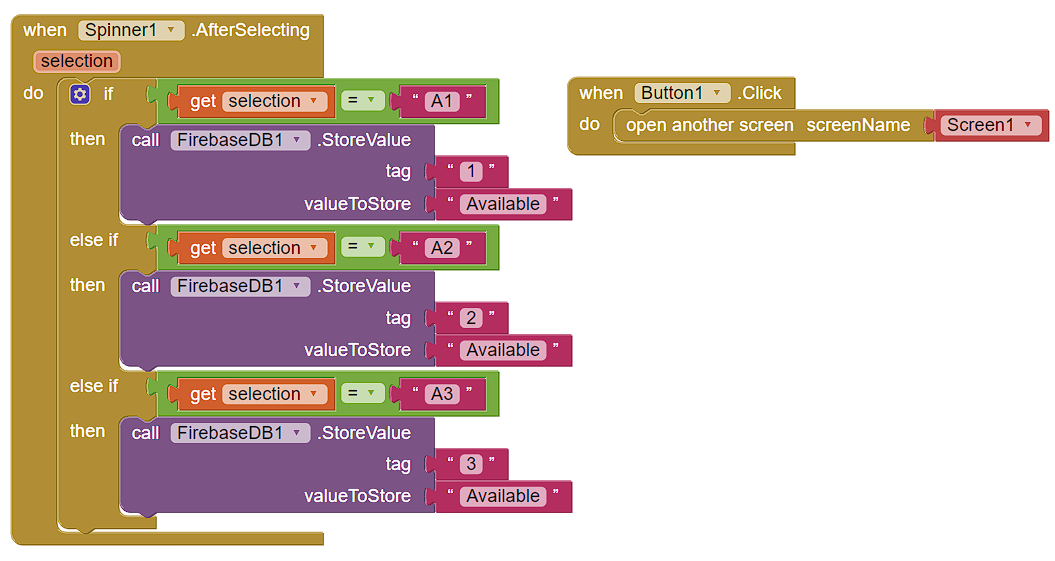
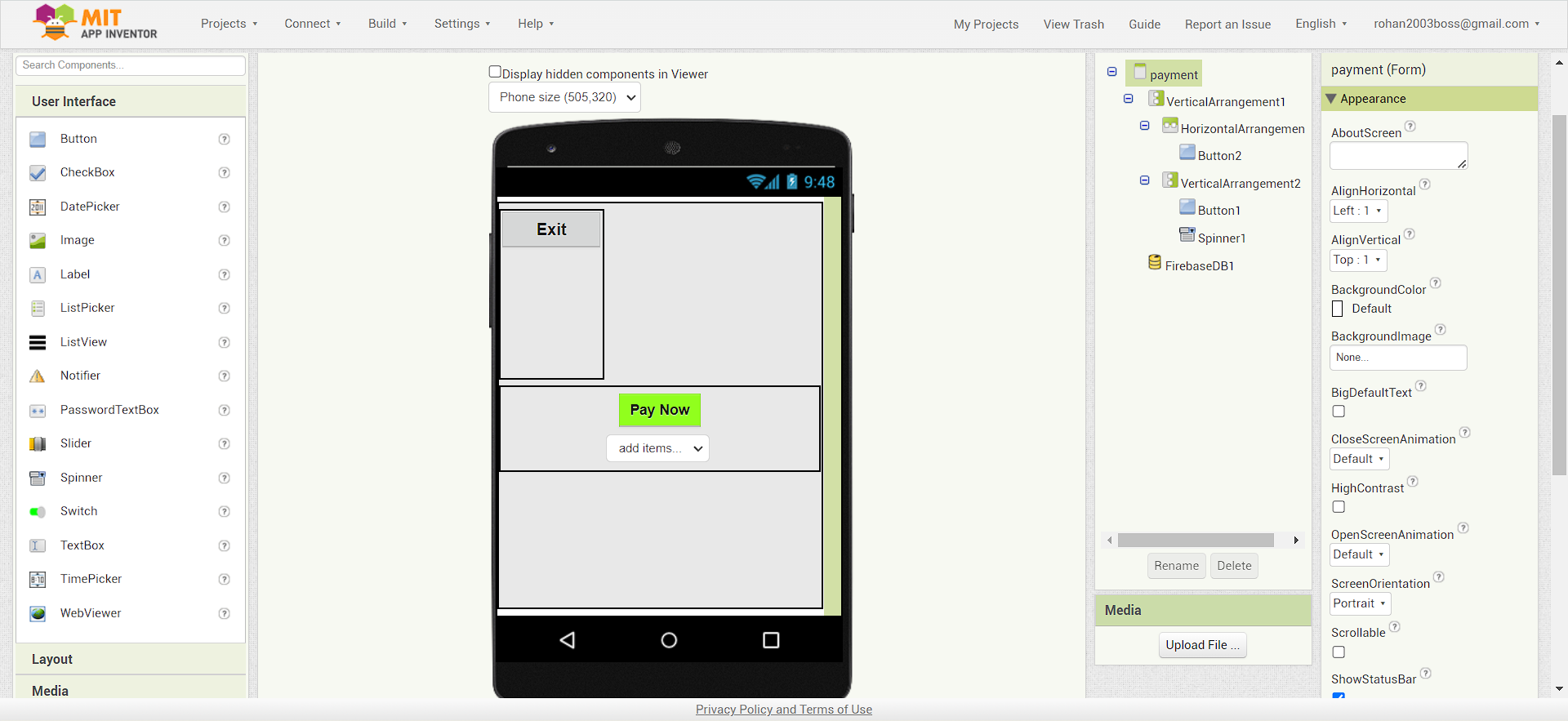
**Screen 1:**



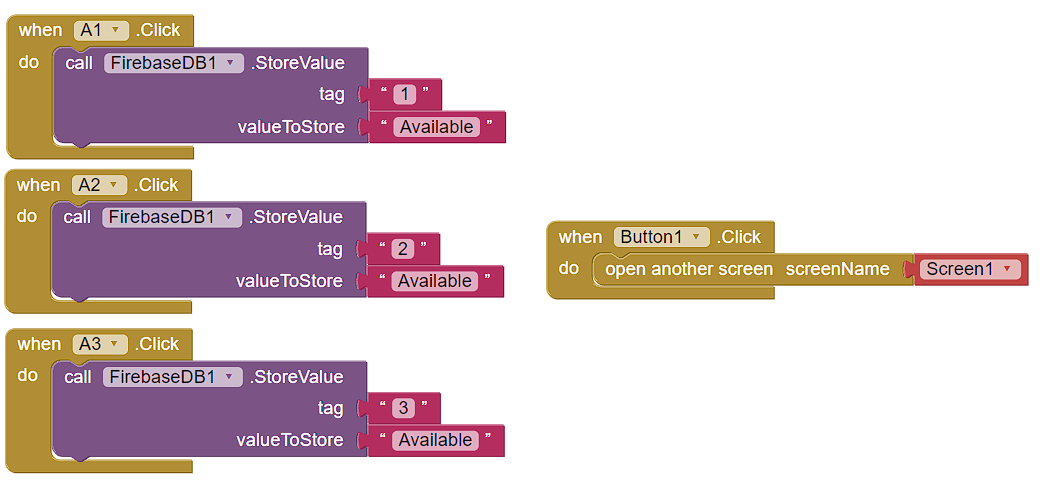
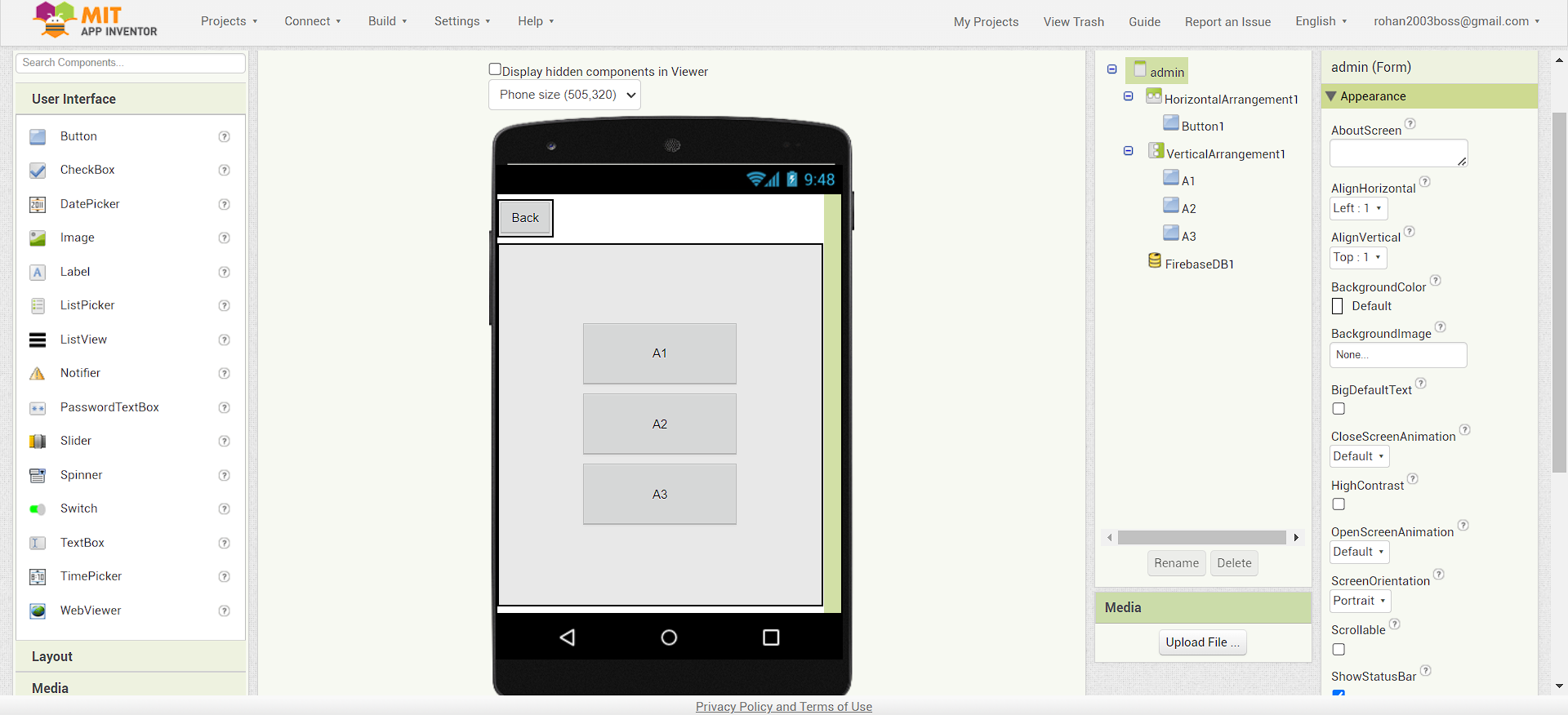
 

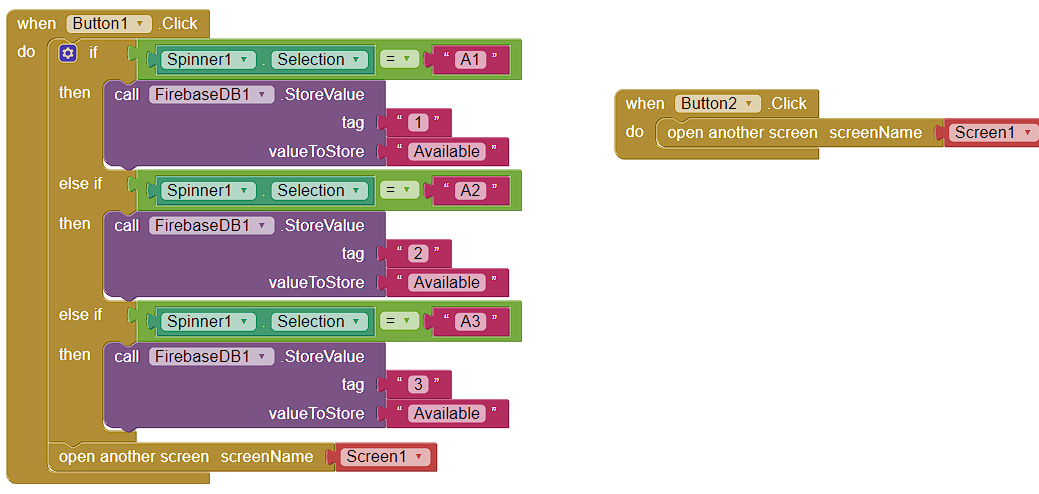
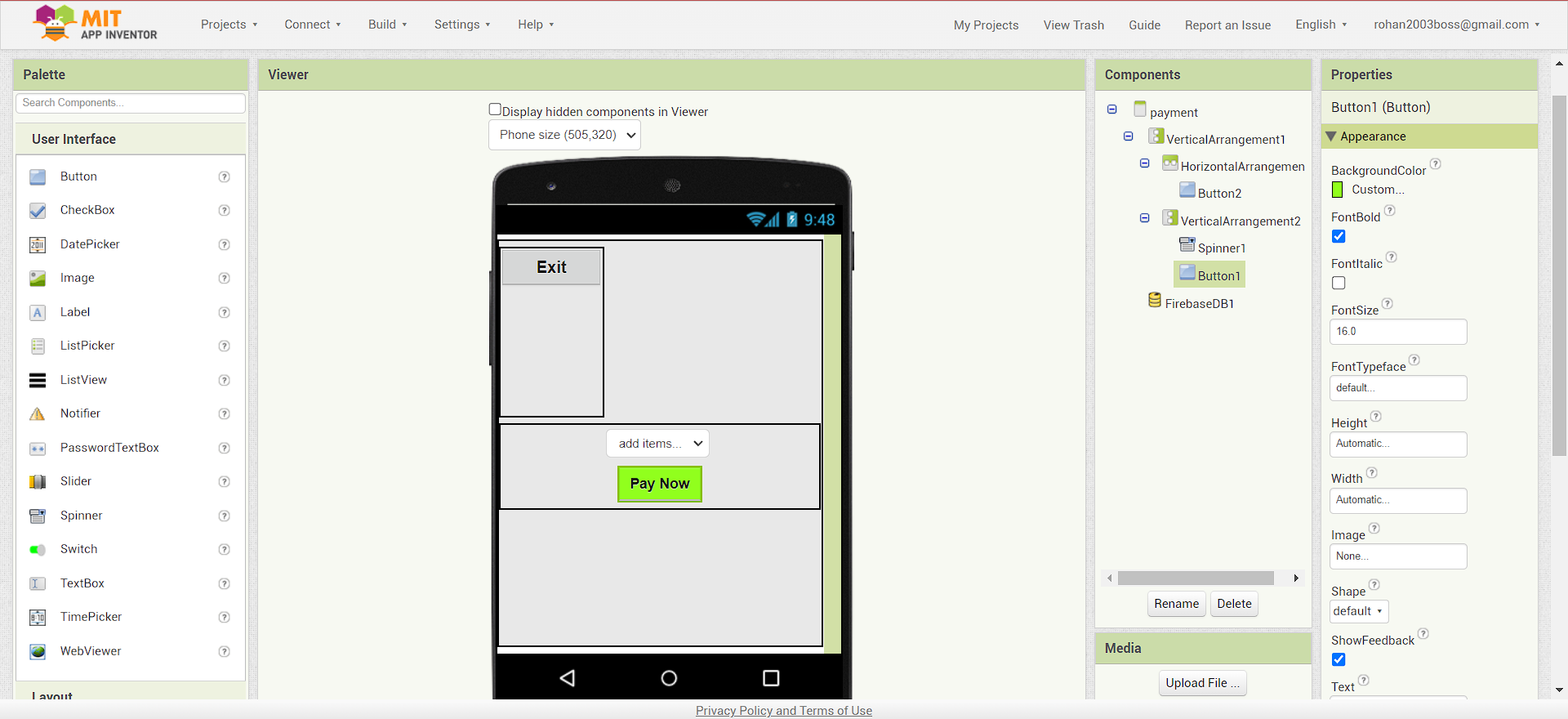
**Admin login:**



**Admin:**

****

**Payment:**

****

**Platform Development:**

* **ESP32**: the code and diagram was designed in the wokwi.
* **Mobile Application**: the mobile application was designed and developed using MIT App inventor

**Code and implementation:**

The necessary codes are attached in the folder.

* Code for ESP32
* Link for wokwi: <https://wokwi.com/projects/379932606462682113>

main.py

import machine

from machine import Pin,PWM

from time import sleep\_us,sleep

from f import Servo

import hcsr04

import urequests

import network as MOD\_NETWORK

#Connect to Wifi

GLOB\_WLAN=MOD\_NETWORK.WLAN(MOD\_NETWORK.STA\_IF)

GLOB\_WLAN.active(True)

GLOB\_WLAN.connect("Wokwi-GUEST", "")

while not GLOB\_WLAN.isconnected():

  pass

# Firebase Realtime Database URL and secret

firebase\_url = "https://ibmnm-2e8ff-default-rtdb.firebaseio.com/"

firebase\_secret = '92CzMcbXSeaLMiHrvSMA8zhxtZre2Q2ORzlujqjD'

# Define the pins for the ultrasonic sensor and the LED

t1 = 0

e1 = 2

t2 = 4

e2 = 5

t3 = 16

e3 = 17

t4 = 18

e4 = 19

t5 = 21

e5 = 22

t6 = 25

e6 = 26

ser1 = 12

ser2 = 13

ser3 = 14

ser4 = 15

led1 = 32

led2 = 33

led3 = 27

us1 = hcsr04.HCSR04(trigger\_pin=t1, echo\_pin=e1)

us2 = hcsr04.HCSR04(trigger\_pin=t2, echo\_pin=e2)

us3 = hcsr04.HCSR04(trigger\_pin=t3, echo\_pin=e3)

us4 = hcsr04.HCSR04(trigger\_pin=t4, echo\_pin=e4)

us5 = hcsr04.HCSR04(trigger\_pin=t5, echo\_pin=e5)

us6 = hcsr04.HCSR04(trigger\_pin=t6, echo\_pin=e6)

# Initialize the LED

l1 = Pin(led1, Pin.OUT)

l2 = Pin(led2, Pin.OUT)

l3 = Pin(led3, Pin.OUT)

# Initialize the servo

pwm1 = PWM(Pin(ser1), freq=50, duty=0)

pwm2 = PWM(Pin(ser2), freq=50, duty=0)

pwm3 = PWM(Pin(ser3), freq=50, duty=0)

pwm4 = PWM(Pin(ser4), freq=50, duty=0)

# Define the detection range (in centimeters)

detection\_range = 250

l=["A1","A1","A3"]

def occup(led,ser,slot):

    led.on()

    Servo(0,ser)

    url = f'{firebase\_url}/ibmnm.json?auth={firebase\_secret}'

    data={slot:"Occupied"}

    response = urequests.patch(url, json=data)

    response.close()

def avail(led,ser,slot):

    led.off()

    Servo(90,ser)

    url = f'{firebase\_url}/ibmnm.json?auth={firebase\_secret}'

    data={slot:"Available"}

    response = urequests.patch(url, json=data)

    response.close()

def dist(usno):

    dis=usno.distance\_cm()

    return dis

while True:

    occup(l1,ser2,"1") if dist(us1)<detection\_range and dist(us2)<detection\_range else avail(l1,ser2,"1")

    occup(l2,ser3,"2") if dist(us3)<detection\_range and dist(us4)<detection\_range else avail(l2,ser3,"2")

    occup(l3,ser4,"3") if dist(us5)<detection\_range and dist(us6)<detection\_range else avail(l3,ser4,"3")

    if dist(us1)<detection\_range and dist(us2)<detection\_range and dist(us3)<detection\_range and dist(us4)<detection\_range and dist(us5)<detection\_range and dist(us6)<detection\_range:

        Servo(0,ser1)

* The application apk, MIT app inventor data aia file and necessary files required for the ESP32 are attached with the folder.
* Files attached:
  + main.py
  + diagram.json
  + f.py
  + hcsr04.py
  + servo.py
  + ibmnm.aia
  + ibmnm.apk

**How to deploy it:**

* First import -
  + main.py
  + diagram.json
  + f.py
  + hcsr04.py
  + servo.py

in wokwi to create the circuit and start to work.

* Install the ibmnm.apk in your device and run it.
* Then change the ultrasonic detection distance to view the change occurring.
* If you want to check if a car is parked then change both the ultrasonic sensors which are parallel to each other at range below 250cm.