IoT-enabled Smart Parking System

Anh Tuan Doan (Student ID: 103526745)

April 2023

Contents

1 Summary

The Internet of Things has the ability to connect multiple physical sensors, actuators and edge devices to form an interconnected network and allow those devices to collect and exchange information. Therefore, IoT can be used in many services to automate tasks, reduce costs and improve productivity. One of those applications that could be improved by IoT is parking space services. Traditional parking services require human labour to manage customers, collect fees and manually open the parking lot gate. Moreover, customers also have to waste much time finding an available parking space especially when the parking lot is crowded. For this reason, this report will propose an IoT-enabled Smart Parking System that uses physical sensors and actuators instead of human labour to automatically manage customers, monitor entries to the parking space and track the number of available parking slots. Moreover, the system

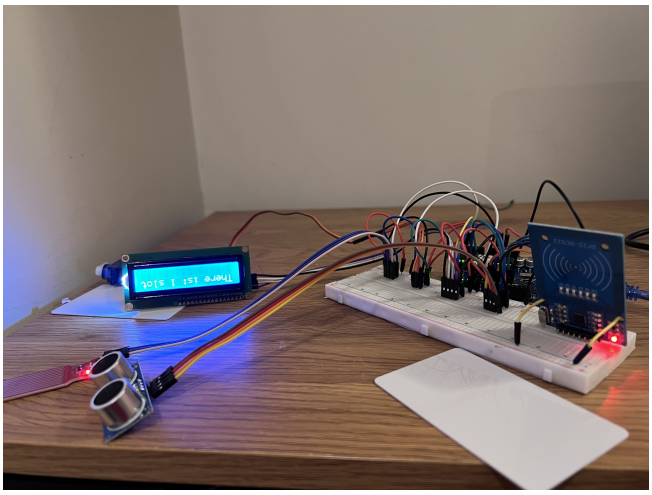

Figure 1: Smart Parking System

also comes with a web-based interface that allows the manager to see the customer list, analyze collected data, change the password for the gate and interact with the physical devices in real-time.

In particular, the proposed system consists of 2 main parts: the hardware and software components. The hardware component includes an Arduino Uno microcontroller, an edge device running Raspberry Pi OS, multiple physical sensors, and actuators. On the other hand, the software component includes a web-based user interface as well as a database to store and analyze the collected data. In terms of the hardware, this system consists of multiple physical devices including an ultrasonic sensor to detect a vehicle, an RFID reader to read the customer's information on the Mifare card, a water sensor to detect rain, a LCD screen to display the number of available parking space, and finally a servomotor to control the gate. About the software component, the edge device hosts a small MySQL database to store the customer information as well as the entries to the parking space. Moreover, the edge device also runs a simple web-based interface for the manager to see the collected data and interact with the physical gate in real-time.

The proposed solution can bring many benefits as it can operate without human labour to reduce cost and improve efficiency. Moreover, the system also enhances the customers' experience and satisfaction. When a driver wants to use the parking lot, they just need to put their Mifare card on top of the RFID reader. If the Mifare card has the correct password, the gate will automatically open and allows the driver to get inside. Furthermore, the customers can look at the LCD screen to see how many spaces are available before going inside the parking lot. In addition, the system can also detect rain and reminds the customers to grab their umbrellas before leaving their vehicles.

2 Conceptual Design

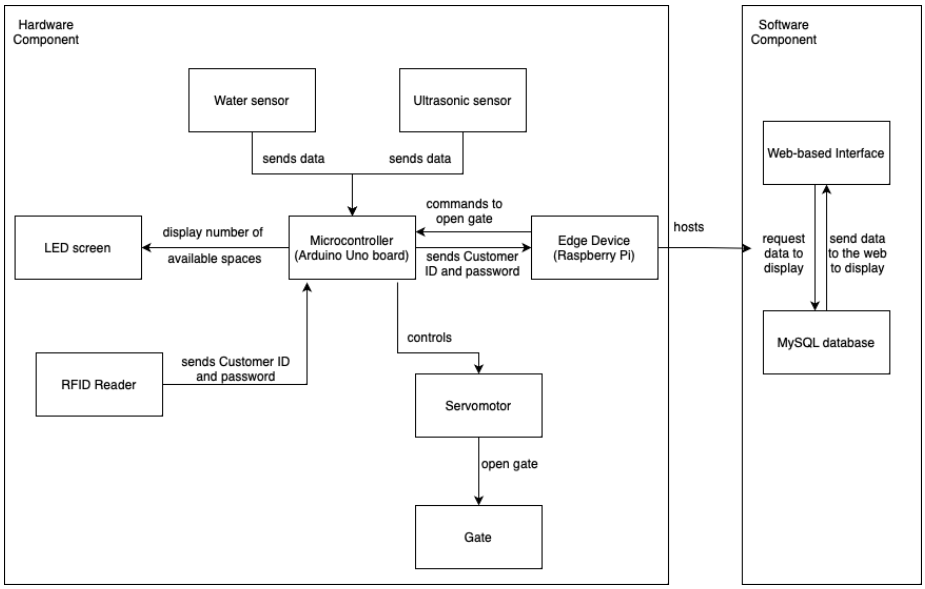

Figure 2: Hardware and Software components

Overall, the proposed system can be divided into 2 main parts: the hardware and the software component. The hardware consists of physical devices including the microcontroller, the edge device, different sensors and actuators while the software component is made up of a web-based interface and a MySQL database. The central unit of the hardware component is the microcontroller as it connects every physical device and allows them to exchange information. It can receive and process data collected from the physical sensors to control the actuators and send the information to the edge device. Below is the list of the sensors and actuators that are used for the system.

- RFID Reader: This device can be used to read the data written on the customer's Mifare card including a customer ID and a password.
- Ultrasonic Sensor: An ultrasonic sensor is installed at every parking slot to detect the presence of a vehicle. By connecting to the sensors corresponding to the parking slots, the microcontroller can calculate the total number of free parking spaces and display it on the LCD screen.
- Water Sensor: A water level sensor is used to detect when it is raining and reminds the customers to grab their umbrellas.
- LCD Screen: the screen is used to display the number of available slots in the parking area and also reminds the customers to grab their umbrellas when it is raining.

- Servomotor: the servomotor is used to control the gate of the parking space.

Besides the sensors and actuators, the microcontroller is also connected to an edge device via serial communication. The functionality of the edge device is to receive data sent by the microcontroller and organise the collected information into a database. Moreover, the edge device can also send a command back to the microcontroller to control the servomotor and open the physical gate. In addition, the device also hosts a web-based interface that can display and analyze relevant data from the database. Furthermore, the manager can use the website to change the password for the gate and remotely open the gate via a button.

Next, we will look at the application's flowchart and how different devices interact with each other in more detail (Figure 3). Firstly, the microcontroller read the data from the water sensor and prints the text "Grab an Umbrella" if it detects rain. Next, the microcontroller will use the ultrasonic sensor to detect the presence of a vehicle and calculate the number of available parking slots. Then, it reads from the serial line to check if the edge device has sent an "open" command, if that is the case, the microcontroller will control the servomotor to automatically open the gate. To detect and read the customers' Mifare cards, the system uses an RFID reader. When the sensor detects a card, the microcontroller will read the CustomerID and the password written on the card's chip and send those data to the edge device. Here, the edge device will check the password written on the card against the password stored locally on the edge device. If the passwords match, the edge device will send an "open" command to the Arduino to open the gate and record the entry to the database. Finally, the whole process gets repeated in a loop.

3 Implementation

3.1 Sensors

- Ultrasonic sensor (Digital sensor): The ultrasonic modules are installed at every parking slot to detect the presence of a vehicle. Therefore, the microcontroller can read the data from the ultrasonic sensor to determine the total number of available parking slots at the current moment. In particular, the ultrasonic sensor works by sending high-frequency sound waves. It waits for the waves to reach an object and bounce back to the sensor. Then, the device can estimate the distance between the sensor and the object based on the amount of time for the waves to travel to the object and returns back to the sensor. The sensor used in the prototype is the HC-SR04 module. To connect the ultrasonic sensor to the microcontroller, we can connect the Trigger and Echo pins on the sensor to two digital pins on the Arduino. The Trigger pin is used to trigger the ultrasonic sound pulses and the Echo pin is used to detect when the waves return.

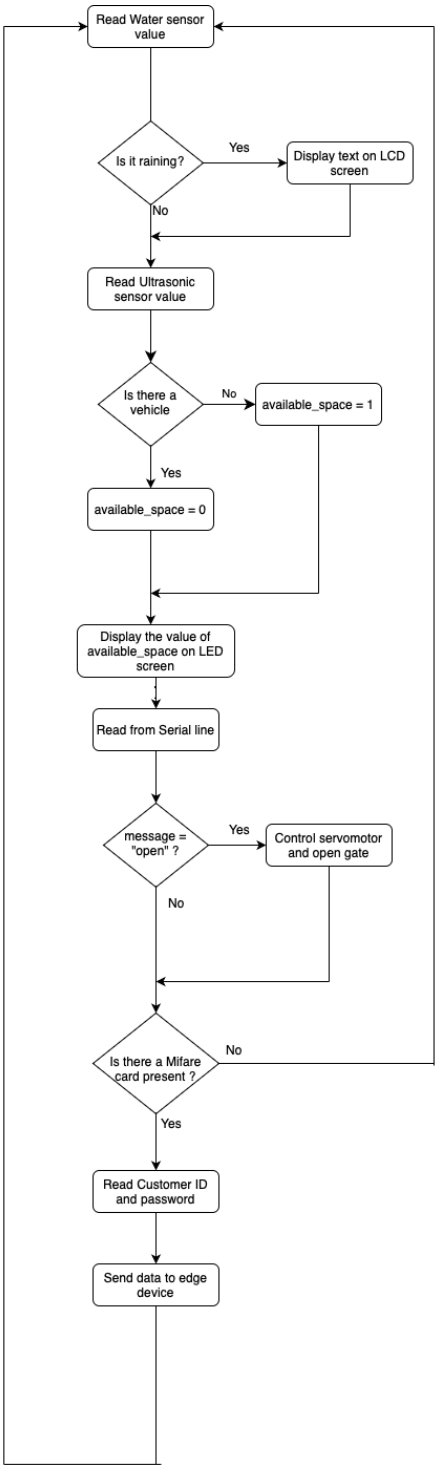

Figure 3: The application's flowchart

- RFID reader and Mifare cards: In this application, we use an RFID reader to read the customer information on the Mifare cards. When a customer registers for the parking lot, they are provided with a Mifarecard that has a chip with the customer ID and a password written on it. When the customer wants to enter the parking lot, they just need to tap their Mifare card on the RFID reader. The RFID reader can read the data on the Mifare card via wireless radio waves. If the password written on the Mifare card is correct, the microcontroller will open the gate and allows them to get inside. As an additional note, the RFID reader used in the prototype is the RFID-RC522 module which can be connected to the Arduino microcontroller via SPI protocol.

- Water sensor (Analog sensor): A water sensor is used to detect rain and reminds the customer to grab an umbrella before leaving their vehicles. In particular, the signal pin of the sensor is connected to an analog pin on the Arduino microcontroller. When the value of the analog pin is greater than 100, it means that the sensor has detected a water drop. Then the microcontroller will use the LCD screen to print out a text to remind the customers to grab their umbrellas.

3.2 Actuators

- LCD Screen: The LCD screen is used to communicate with the customers. It can display the number of available parking slots and reminds the customers to grab their umbrellas when it rains. The prototype uses the LCD1602 screen which can be easily connected to the Arduino microcontroller via an I2C module.

- Servomotor: In the prototype, a small 9G servomotor is used to control the gate for the parking lot. In particular, when the microcontroller receives the "open" command from the edge device, it will control the servomotor to open the gate for the customers.

3.3 Software/Libraries

- Web-based interface: The proposed solution also comes with a web-based user interface that allows the manager to see the collected data and interact with the physical devices in real-time. The backend of the website was written using the Python Flask framework while the frontend is made up of HTML, CSS and Javascript code. Moreover, the backend and the frontend sides can communicate with each other via both HTTP and WebSocket protocols. When the manager goes to the website, it will display a list of the customers as well as some analysis such as the number of entries to the parking lot in the last 7 days and the average number of entries per day. Furthermore, the manager can also change the password and remotely open the physical gate via a button on the website. Especially, the application uses multiple threads to both run the website and listen

to the serial line at the same time. Therefore, it can handle web requests and receive data from the serial line. When a customer puts their card on top of the RFID reader, the microcontroller will send the customer ID and password written on the card to the edge device. Here, the edge device will check the password and send the "open" command back to the microcontroller if the password is correct. In addition, the application also records the customer ID and the time when the customer enters the parking space into a database for analysis.

- MySQL: This application uses a MySQL database to record the customers' information as well as the entries to the parking lot. The database has 2 tables: Customer and Parking_Entry tables. The first table record the customer ID, first name, last name, phone number, car model, and plate number, while the second table stores the time when a customer enters the parking lot and their customer ID. Therefore, the application can use the Parking_Entry table to calculate the number of entries in a day and do simple analyses such as finding the average daily usage (Figure 4).

- Additional libraries: Below is the list of additional libraries that is used in the projects:

  - MFRC522 library by Miki Balboa: This is an Arduino Library for the RFID reader to read the information on the Mifare cards.
  - ArduinoJson library by Benoit Blanchon: This library is used to serialize data collected from sensors into JSON format
  - Servo library: This is a built-in library from Arduino to control servomotor
  - SPI library: This built-in library allows the Arduino to communicate with sensors via SPI protocol.
  - Ultrasonic library by Erick Simões: This library is for Ultrasonic modules. It can calculate the distance from the sensor to an object.
  - LiquidCrystal I2C library by Frank de Brabander: This library is used to display text on I2C LCD displays.
  - Flask-SocketIO Python library by Miguel Grinberg: This library is used in the web-based interface to facilitate WebSocket communication between the client and the server.
  - MySQL Connector: This library is used in the Python application to connect to the MySQL database.
  - PySerial: This library is used to facilitate serial communication between the edge device and the Arduino board.
  - Chart.js: This library is used to create simple charts on the web interface.

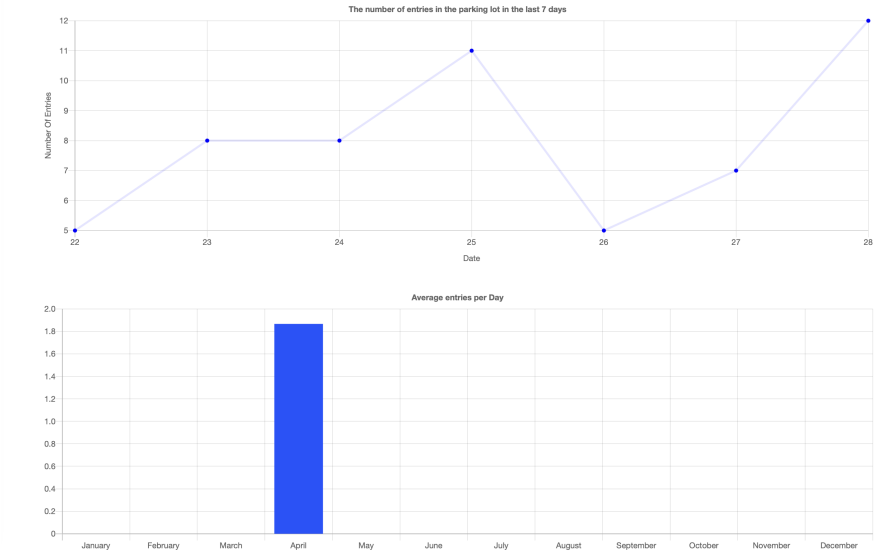

Figure 4: Data Analysis

4 Resources

Below is the list of the online tutorials, guides, manuals, and libraries that were utilised in the project:

- Balboa, M. (2021, November 2). RFID: Arduino Rfid Library for MFRC522. GitHub. Retrieved April 1, 2023, from `https://github.com/miguelbalboa/rfid`.
- Blanchon, B. (2014). Efficient JSON serialization for embedded C++. ArduinoJson. Retrieved April 1, 2023, from `https://arduinojson.org/?utm_source=meta&utm_medium=library.properties`.
- Simoes, E. (2018, October 25). Ultrasonic. GitHub. Retrieved April 2, 2023, from `https://github.com/ErickSimoes/Ultrasonic`.
- Brabander, F. de. (2016, March 7). Liquidcrystal Arduino Library for the dfrobot I2C LCD displays. GitHub. Retrieved April 2, 2023, from `https://github.com/johnrickman/LiquidCrystal_I2C`.
- Grinberg, M. (2014). Flask-SocketIO: Socket.io integration for flask applications. GitHub. Retrieved April 2, 2023, from `https://github.com/miguelgrinberg/flask-socketio`.
- MySQL Connector/Python Developer Guide. MySQL. (2023, April 26). Retrieved April 3, 2023, from `https://dev.mysql.com/doc/connector-python/en/`.

- Pyserial. (2020, September 21). pyserial: Python serial port access library. GitHub. Retrieved April 4, 2023, from https://github.com/pyserial/pyserial.
- Chart.js. Open source HTML5 Charts for your website. (n.d.). Retrieved April 3, 2023, from https://www.chartjs.org
- Viral Science - The home of Creativity. (2017, August 25). Arduino Rfid Sensor (MFRC522) tutorial. YouTube. Retrieved April 4, 2023, from https://www.youtube.com/watch?v=KQiVLEhzzV0&t=245s.
- JR-ECT. (2020, January 22). Arduino using LCD1602 display with I2C module (experiment 6 - get started with Arduino). YouTube. Retrieved April 3, 2023, from https://www.youtube.com/watch?v=z4CX3UaJHFo&t=129s.
- TheGeekPub. (2019, May 12). Arduino water level sensor tutorial. YouTube. Retrieved April 3, 2023, from https://www.youtube.com/watch?v=n7WRi5U5lQk&t=104s.
- Dom. (2020, July 21). Creating beautiful HTML tables with CSS. Retrieved April 2, 2023, from https://dev.to/dcodeyt/creating-beautiful-html-tables-with-css-428l

5 Appendix

This section will list all related source code from the project. The source code can also be found on https://github.com/TuanDoan14112003/SmartParkingSystem.

5.1 Arduino Script

```
#include <SPI.h>
#include <MFRC522.h>    //MFRC522 library by Miki Balboa
#include <ArduinoJson.h> //ArduinoJson library by Benoit Blanchon
#include <Servo.h>
#include <Ultrasonic.h>      // Ultrasonic library by Erick Simoes
#include <LiquidCrystal_I2C.h> // LiquidCrystal I2C library by Frank de
    Brabander

#define RST_PIN 9
#define SS_PIN 10
#define TRIG_PIN 7
#define ECHO_PIN 6
#define RAIN_SENSOR_PIN A3
#define SERVO_PIN 8
```

```
#define CUSTOMERID_BLOCK 54
#define PASSWORD_BLOCK 53
#define NAME_BLOCK 57

LiquidCrystal_I2C lcd(0x27, 20, 4);
Ultrasonic ultrasonic(TRIG_PIN, ECHO_PIN);
StaticJsonDocument<200> doc;
MFRC522 mfrc522(SS_PIN, RST_PIN);
Servo servo1;
int numberOfSlot; // The number of available parking slots in the
    parking area.
int waterVal = 0;

void setup() {
  Serial.begin(9600);
  SPI.begin();
  lcd.init();
  lcd.backlight();
  mfrc522.PCD_Init();
  servo1.attach(SERVO_PIN);
  servo1.write(0);
  lcd.clear();
}


bool readData(byte blockNumber, byte buffer1[]) {
  // This function reads the data on the blockNumber of the Mifare card
      and copy the data into a buffer
  // blockNumber: The block number on the Mifare card
  // buffer1: The buffer that the data will be copied into
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
  MFRC522::StatusCode status;
  byte len = 18;

  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A,
      blockNumber, &key, &(mfrc522.uid)); // authenticate
  if (status != MFRC522::STATUS_OK) {
    return false;
  }

  status = mfrc522.MIFARE_Read(blockNumber, buffer1, &len); // read the
      data on the blockNumber and copy the data into buffer
  if (status != MFRC522::STATUS_OK) {
    return false;
  }

  return true;
}
```

```
void loop() {
  waterVal = analogRead(RAIN_SENSOR_PIN); // Read the data of the water
      sensor
  lcd.setCursor(0, 1);
  if (waterVal > 100) {
    lcd.print("Grab an umbrella"); // Display the text if the value is
        larger than 100
  } else {
    lcd.print("               "); // Clear the text
  }

  int distance = ultrasonic.read(); // Read the value of the ultrasonic
      sensor

  if (distance < 10) { // If there is a vehicle in less than 10cm
    numberOfSlot = 0;
  } else {
    numberOfSlot = 1;
  }

  lcd.setCursor(0, 0);
  lcd.print("There is: " + String(numberOfSlot) + " slots"); // Display
      the number of available slots on the screen
  if (Serial.available() > 0) {                          // Read from the
      serial line
    String a = Serial.readString();
    a.trim();
    if (a == "open") {
      servo1.write(90); // Open the gate if it receive command "open"
      delay(2000);     // Hold the gate for 2 seconds
      servo1.write(0); // Close the gate
    }
  }


  if (!mfrc522.PICC_IsNewCardPresent()) { // If the RFID reader detects
      a Mifare card
    return;
  }


  if (!mfrc522.PICC_ReadCardSerial()) {
    return;
  }


  byte nameBuffer[18];
  byte passwordBuffer[18];
  byte customerIDBuffer[18];
```

```
  if (readData(NAME_BLOCK, nameBuffer) && readData(PASSWORD_BLOCK,
      passwordBuffer) && readData(CUSTOMERID_BLOCK, customerIDBuffer))
      { // Read Customer name, Customer ID, and password
    String name;
    for (uint8_t i = 0; i < 16; i++) { // Convert data into string
      name += char(nameBuffer[i]);
    }
    name.trim();

    String customerID;
    for (uint8_t i = 0; i < 16; i++) {
      customerID += char(customerIDBuffer[i]);
    }

    customerID.trim();

    String password;
    for (uint8_t i = 0; i < 16; i++) {

      password += char(passwordBuffer[i]);
    }
    password.trim();

    // Serializing the data into JSON format
    doc["password"] = password;
    doc["name"] = name;
    doc["customerID"] = customerID;
    serializeJson(doc, Serial); // Send the data to Serial line
    Serial.println();
  }


  mfrc522.PICC_HaltA(); // Stop reading the card
  mfrc522.PCD_StopCrypto1();
}
```

5.2 Edge device script

```
from flask import Flask, render_template, request, redirect
from flask_socketio import SocketIO # WebSocket library
import mysql.connector # MySQL library to connect to the database
import json # To deserialize the data sent by the Arduino
import serial # PySerial library for Serial communication
import threading
from calendar import monthrange, month_name
from datetime import datetime, timedelta
```

```
password = None
with open("password.txt", 'r') as file:
    password = file.readline().strip() # reading the local password

# Connecting to the data base
mydb = mysql.connector.connect(
    host="localhost",
    user="tuandoan",
    database="smart_parking_db"
)
mycursor = mydb.cursor(dictionary=True)
device = '/dev/cu.usbmodem11301'
arduino = serial.Serial(device, 9600) # Setup the Serial communication


app = Flask("arduino project")
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)


@app.route('/')
def index():
    entryDictionary = {}
    averagePerDay = {}

    for i in range(7):
        # calculate the number of entries in the last 7 days
        previousDay = datetime.now() - timedelta(days=i)
        entryQuery = f"SELECT * from Parking_Entry WHERE Date(time) =
            '{previousDay.strftime('%Y-%m-%d')}'"
        mycursor.execute(entryQuery)
        myresult = mycursor.fetchall()
        numberOfEntries = mycursor.rowcount
        entryDictionary.update({str(previousDay.day):
            str(numberOfEntries)})

    currentYear = datetime.now().year
    for month in range(1, 13):
        # calculate the average entries per day
        lastDay = monthrange(2023, month)[1]
        totalEntresInMonthQuery = f"SELECT * from Parking_Entry WHERE
            time BETWEEN '{currentYear}-{month}-01' AND
            '{currentYear}-{month}-{lastDay}'"
        mycursor.execute(totalEntresInMonthQuery)
        mycursor.fetchall()
        averagePerDay.update(
            {str(month_name[month]): str(mycursor.rowcount / lastDay)})

    customerQuery = f"SELECT * from Customer"
```

```
    mycursor.execute(customerQuery)
    customers = mycursor.fetchall()

    # find the customer with the highest number of entries
    bestCustomerQuery = f"SELECT * FROM customer WHERE numberOfEntries =
        ( SELECT MAX(numberOfEntries) FROM customer ) LIMIT 1;"
    mycursor.execute(bestCustomerQuery)
    bestCustomers = mycursor.fetchall()
    print(bestCustomers)
    return render_template('index.html', customerList=customers,
        entries=entryDictionary, averagePerDay=averagePerDay,
        bestCustomer=bestCustomers[0])


@app.route('/changepassword', methods=["POST"])
def changePassword():
    # change the locally stored password
    global password
    password = request.form.get('password')
    with open("password.txt", 'w') as file:
        file.write(password)
    return redirect("/")


@socketio.on('message')
def handle_message(data):
    # command the Arduino to open the gate
    if data == "open gate":
        print(data)
        arduino.write(b"open")


def edge():
    # read from the serial line to detect data from the Arduino
    while True:
        serialData = arduino.readline()
        decodedData = serialData.decode("utf-8")
        entry = json.loads(decodedData)
        print(entry)
        customerID = entry["customerID"]
        # check the password on the Mifare card against the locally
            stored password
        if entry["password"] == password:
            customerID = int(customerID, 0)
            # update the numberOfEntries column
            updateEntriesQuery = f"UPDATE customer SET numberOfEntries =
                numberOfEntries + 1 WHERE CustomerID ={customerID}"
            mycursor.execute(updateEntriesQuery)
            mydb.commit()
            arduino.write(b"open")
```

```
            time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            sql = f"INSERT INTO Parking_Entry (id, CustomerID, time)
                VALUES (NULL,{customerID},'{time}')"
            mycursor.execute(sql)
            mydb.commit()
            print(mycursor.rowcount, "record inserted.")
        else:
            print("not authenticated")


def startServer():
    app.run(threaded=True, debug=True, use_reloader=False,
            host='0.0.0.0', port=8080) # run the web application


thread1 = threading.Thread(target=edge)
thread1.start()

thread2 = threading.Thread(target=startServer)
thread2.start()
```

5.3 Website HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="{{ url_for('static',filename='style.css')
      }}">
  <script
      src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.4/Chart.js"></script>
  <script src="https://cdn.socket.io/4.6.0/socket.io.min.js"
    integrity="sha384-c79GN5VsunZvi+Q/WObgk2in0CbZsHnjEqvFxC5DxHn9lTfNce2WW6h2pH6u/kF+"
        crossorigin="anonymous">
  </script>
  <script
      src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.js"></script>

</head>

<body>
```

```
<header class="center">
  <h1>Smart Parking System</h1>
</header>
<section id="main-section">
  <div id="background-information">
    <div id="introduction">
      <h2>Introduction</h2>
      <p>The Internet of Things has the ability to connect multiple
          physical sensors, actuators and edge devices to
        form an interconnected network and allow those devices to
            collect and exchange information. Therefore, IoT can
        be used in many services to automate tasks, reduce costs and
            improve productivity. One of those applications
        that could be improved by IoT is parking lots. This project
            will propose an IoT-enabled Smart Parking System
        that uses physical sensors and actuators instead of human
            labour to automatically manage customers, monitor
        entries to the parking space and track the number of available
            parking slots. Moreover, the system also comes
        with a web-based interface that allows the manager to see the
            customer list, analyze collected data, change
        the passcode for the gate and interact with the physical
            devices in real time. </p>

    </div>
    <div class="paragraph-container">
      <figure>
        <!-- Source: https://unsplash.com/photos/fZB51omnY_Y -->
        <img src="{{ url_for('static',filename='arduino.jpg') }}"
            alt="Trulli">
      </figure>
      <div class="info">
        <h2>Sensors</h2>
        <p>
          <ul>
            <li> RFID Reader: This device can be used to read the data
                written on the customers Mifare card including
              a customer ID and a password.</li>
            <li> Ultrasonic Sensor: An ultrasonic sensor is installed
                at every parking slot to detect the presence of
              a vehicle. By connecting to the sensors corresponding to
                  the parking slots, the microcontroller can
              calculate the total number of free parking spaces and
                  display it on the LCD screen.</li>
            <li>Water Sensor: A water level sensor is used to detect
                when it is raining and reminds the customers to
              grab their umbrellas.</li>
          </ul>
        </p>
      </div>
```

```
</div>
<div class="paragraph-container">
  <div class="info">
    <h2>Actuators</h2>
    <p>
      <ul>
        <li>LCD Screen: The LCD screen is used to communicate with
            the customers. It can display the number of
          available parking slots and reminds the customers to grab
              their umbrellas when it rains. The prototype
          uses the LCD1602 screen which can be easily connected to
              the Arduino via an I2C module.</li>
        <li>Servomotor: In the prototype, a small 9G servomotor is
            used to control the gate for the parking lot.
          In particular, when the microcontroller receives the open
              command from the edge device, it will
          control the servomotor to open the gate for the
              customers. </li>
      </ul>
    </p>
  </div>
  <figure>
    <!-- Source:
        https://www.electronicwings.com/particle/servo-motor-interfacing-with-particle-photon
        -->
    <img src="{{ url_for('static',filename='servo.jpeg') }}"
        alt="Trulli">
  </figure>

</div>



<div id="data-visualization">
  <h2>Collected data</h2>
  <p>This application uses a MySQL database to record the
      customers information as well as the entries to the
    parking lot. The database has 2 tables: Customer and
        Parking_Entry tables. The first table record the customer
    ID, first name, last name, phone number, car model, and plate
        number, while the second table stores the time
    when a customer enters the parking lot and their customer ID.
        Therefore, the application can use the
    Parking_Entry table to calculate the number of entries in a day
        and do simple analyses such as finding the
    average daily usage. </p>
  <div id="customer-list">
    <table>
      <thead>
        <tr>
```

```
          <th>Customer ID</th>
          <th>First Name</th>
          <th>Last Name</th>
          <th>Phone Number</th>
          <th>Car model</th>
          <th>Car plate</th>
        </tr>
      </thead>
      <tbody>
        {% for customer in customerList %}
        <tr>
          <td>{{customer.CustomerID}}</td>
          <td>{{customer.firstName}}</td>
          <td>{{customer.lastName}}</td>
          <td>{{customer.phone}}</td>
          <td>{{customer.model}}</td>
          <td>{{customer.plate}}</td>
        </tr>
        {% endfor %}
      </tbody>
    </table>

  </div>
  <p>The best customers (with the highest number of entries) is:

    {{bestCustomer.firstName}} {{bestCustomer.lastName}} with
        {{bestCustomer.numberOfEntries}} entries

  </p>
  <div class="chart-containter">
    <canvas id="lineChart"></canvas>
  </div>
  <div class="chart-containter">
    <canvas id="barChart"></canvas>
  </div>




</div>
<div id="interaction">
  <h2>Real Time Interaction</h2>
  <div>
    <form method="POST" action="/changepassword">
      <h3>Change password</h3>
      <input type="text" id="password" name="password"><br>
      <button type="submit"> Change password</button>
    </form>
```

```
  </div>
  <h3>Open gate in real time</h3>
</div>
<script>
  const xValues = [{{', '.join(entries.keys())}}];
  console.log(xValues);
  const yValues = [{{', '.join(entries.values())}}];
  console.log(yValues);
  new Chart("lineChart", {
    type: "line",
    data: {
      labels: xValues.reverse(),
      datasets: [{
        fill: false,
        lineTension: 0,
        backgroundColor: "rgba(0,0,255,1.0)",
        borderColor: "rgba(0,0,255,0.1)",
        data: yValues.reverse()
      }]
    },
    options: {
      maintainAspectRatio: false,
      title: {
        display: true,
        text: 'The number of entries in the parking lot in the last
            7 days'
      },
      legend: {
        display: false
      },

      scales: {
        yAxes: [{
          scaleLabel: {
            display: true,
            labelString: 'Number Of Entries'
          }
        }],
        xAxes: [{
          scaleLabel: {
            display: true,
            labelString: 'Date'
          }
        }],


      },

    }
  });
```

```
  var months = "{{ ','.join(averagePerDay.keys()) }}";
  months = months.split(',')
  const values = [{{', '.join(averagePerDay.values())}}];
  console.log(months)
  console.log(values)
  new Chart("barChart", {
    type: "bar",
    data: {
      labels: months,
      datasets: [{
        data: values,
        backgroundColor: ["#b30000", "#7c1158", "#4421af",
            "#1a53ff", "#0d88e6", "#00b7c7", "#5ad45a",
          "#8be04e", "#ebdc78", "#b30000", "#7c1158", "#4421af"
        ]

      }]
    },
    options: {
      legend: {
        display: false
      },
      title: {
        display: true,
        text: "Average entries per Day"
      },
      maintainAspectRatio: false,

    }
  });
</script>



<script type="text/javascript" charset="utf-8">
  var socket = io.connect("localhost:8080");
  socket.on('connect', function () {
    socket.send("connected");
  });
  var btn = $("<button class='button'>Open Gate</button>");
  $("#interaction").append(btn);

  function openGate() {

    socket.send("open gate");

  }
  btn.on("click", openGate);
</script>
```

```
  </section>
</body>

</html>
```

5.4 Website CSS

```
@import
    url('https://fonts.googleapis.com/css2?family=Open+Sans:ital,wght@0,300;0,400;0,500;0,600;1,300;1
* {
    margin:0px;
    font-family: 'Open Sans', sans-serif;

}


header {
    text-align: center;
    background-image: linear-gradient(0deg, #0000007d, rgba(0, 0, 0,
        0.5)),url("header.jpg"); /* Source:
        https://unsplash.com/photos/rCZQCbUAQvg*/
    color: white;
    height:250px;
    line-height: 250px;
    padding: 0px;
    background-size: cover;
    background-position: 45% 65%;
}

#main-section {
    margin-right: 7%;
    margin-left: 7%;
}
#introduction {
    margin-top:30px;
    margin-bottom: 100px
}
#introduction > p {
    text-align: justify;
}
#introduction > h2{
    text-align: center;
}

#introduction > p {
    text-align: justify;
}
.paragraph-container {
```

```
    margin-top: 25px;
    display: flex;
    gap: 5%;
}
.paragraph-container > figure {
    width: 50%;
}
.paragraph-container > figure > img {
    width: 100%;
}

.paragraph-container > .info {
    width: 50%;
    text-align: justify;


}
.paragraph-container > .info > h2 {
    text-align: center;
}

.chart-containter {
    margin-top: 30px;
    height: 400px;
}
#data-visualization {
    margin-top: 35px;
}
#data-visualization > h2 {
    text-align: center;
}

#data-visualization > p {
    text-align: justify;
}
#customer-list {
    margin-top: 35px;
    margin-bottom: 35px;
}

table {
    border-collapse: collapse;
    font-size: 1em;
    box-shadow: 0 0 30px rgba(0, 0, 0, 0.20);
    width: 100%;
}

thead tr {
    background-color: #055380;
    color: white;
```

```
}

th,
td {
    padding: 15px 20px;
    text-align: center;
}

tbody tr:nth-of-type(even) {
    background-color: #f3f3f3a9;
}


#interaction {
    margin-top:35px;
    margin-bottom: 40px;

}

#interaction h2 {
    text-align: center;
}


#interaction h3 {
    margin-top:35px;
    margin-bottom:10px;
}

#interaction button {
    margin-top:10px;
}


button {
  background-color: #13e672;
  border-radius: 5px;
  padding: 10px 20px;
}

button:hover {
  background-color: rgb(28, 209, 79);
  cursor:pointer;
}
```

5.5 MySQL Tables

```
USE smart_parking_db;
```

```
CREATE TABLE Customer( CustomerID int(11) AUTO_INCREMENT NOT NULL,
    firstName VARCHAR(20) NOT NULL, lastName VARCHAR(20) NOT NULL,
    phone VARCHAR(10) NOT NULL, model VARCHAR(10) NOT NULL, plate
    VARCHAR(7) NOT NULL, PRIMARY KEY(CustomerID));

CREATE TABLE Parking_Entry( ID int(11) AUTO_INCREMENT NOT NULL,
    CustomerID int(11) NOT NULL, time DATETIME NOT NULL ,PRIMARY
    KEY(ID),FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID));
```