***Tools and Framework***

Arduino IDE

Used to program both Arduino and NodeMCU.

Fritzing

Used to create Connectivity and Schematic circuits for the hardware.

**Firebase**

*NodeMCU connection to Firebase*

NodeMCU must be connected to the internet for the connection to be established.

FirebaseArduino.h

NodeMCU uses FirebaseArduino.h library to connect to Firebase and add/delete/update data to the Firebase database. The library supports NodeMCU as a client.

Configuration

1. In Firebase Database Rules, the read and write conditions are changed to true.
2. In the code, we add the database link ‘FIREBASE\_HOST’ and its secret key ‘FIREBASE\_AUTH’ for authentication (found in the database settings)

#define FIREBASE\_HOST "fir-auth-45665.firebaseio.com"

#define FIREBASE\_AUTH "oYCggxTfYvvMEwPoQN2vM59ZzTX2Lt2A7KFBT31U"

1. Then Firebase connection is started with

Firebase.begin(FIREBASE\_HOST, FIREBASE\_AUTH);

Update data to and from Firebase

FirebaseArduino gets or sets/updates a value with a path. This means that a nested value or a complete node in the JSON tree can be retrieved or updated by passing the path as an argument to the function. For example the path below is the status of the first spot in CENG Female Zone.

path = “spots/ -LZaE7RMP-v3D7gQ3-eb/status”;

Afterwards, FirebaseArduino.h functions are used to either get or set a value in that specific node. The path is created in setting functions, if it does not exist in the database the path.

To retrieve status:

status = Firebase.getString(path);

To update/insert status:

Firebase.setString(path, "not available");

The process is different with objects (whole nodes).

To retrieve:

FirebaseObject nodeReserv = Firebase.get("reservations");

JsonObject& reservations = nodeReserv.getJsonVariant();

To go through the tree as JSON, we convert the received FirebaseObject to a JsonObject (created with ArduinoJson.h library) using the FirebaseArduino.h fuction getJsonVariant().

The project’s hardware only handles the three cases above. However, FirebaseArduino.h can also delete, append, retrieve, set (insert/update) to both values and nodes.

***Hardware Implementation***

*Libraries*

The main critical libraries for understanding the hardware implementation are:

FirebaseArduino.h

Explained in Firebase connection section

SoftwareSerial.h

Allows serial communication on digital pins using software. The SoftwareSerial gives the same functionalities as the hardware serial communication (pins 0 and 1).

The hardwired serial communication is restricted to one serial communication link and is always on pins 0 and 1. Even though we only use one serial communication link for each module, debugging can only be done through the hardware serial link. Hence, SoftwareSerial is added to allow us to debug on hardware serial whenever it is needed.

ArduinoJson.h

A C++ Json library for Arduino and Internet of Things (IoT). The library allows Arduino and NodeMCU to create JSON structured data (As explained in database schema 4.x).

In our project, JSON data is used in:

* Send/Receive data in Serial with SoftwareSerial.h
* Send/Receive data in Firebase with FirebaseArduino.h

NTPClient.h

NTP is “Network Time Protocol” a standard Internet Protocol (IP) that synchronizes time on computers to UTC (Coordinated Universal Time) with the access of internet connection. As the name hints, the NTP applied in our project works in a server-client manner. The process of getting time in general is:

* Client (NodeMCU) contacts the NTP server
* Client gets the current UTC (time stamp) from NTP server
* Client applies any local time zone offset (in Qatar UTC +3.00)

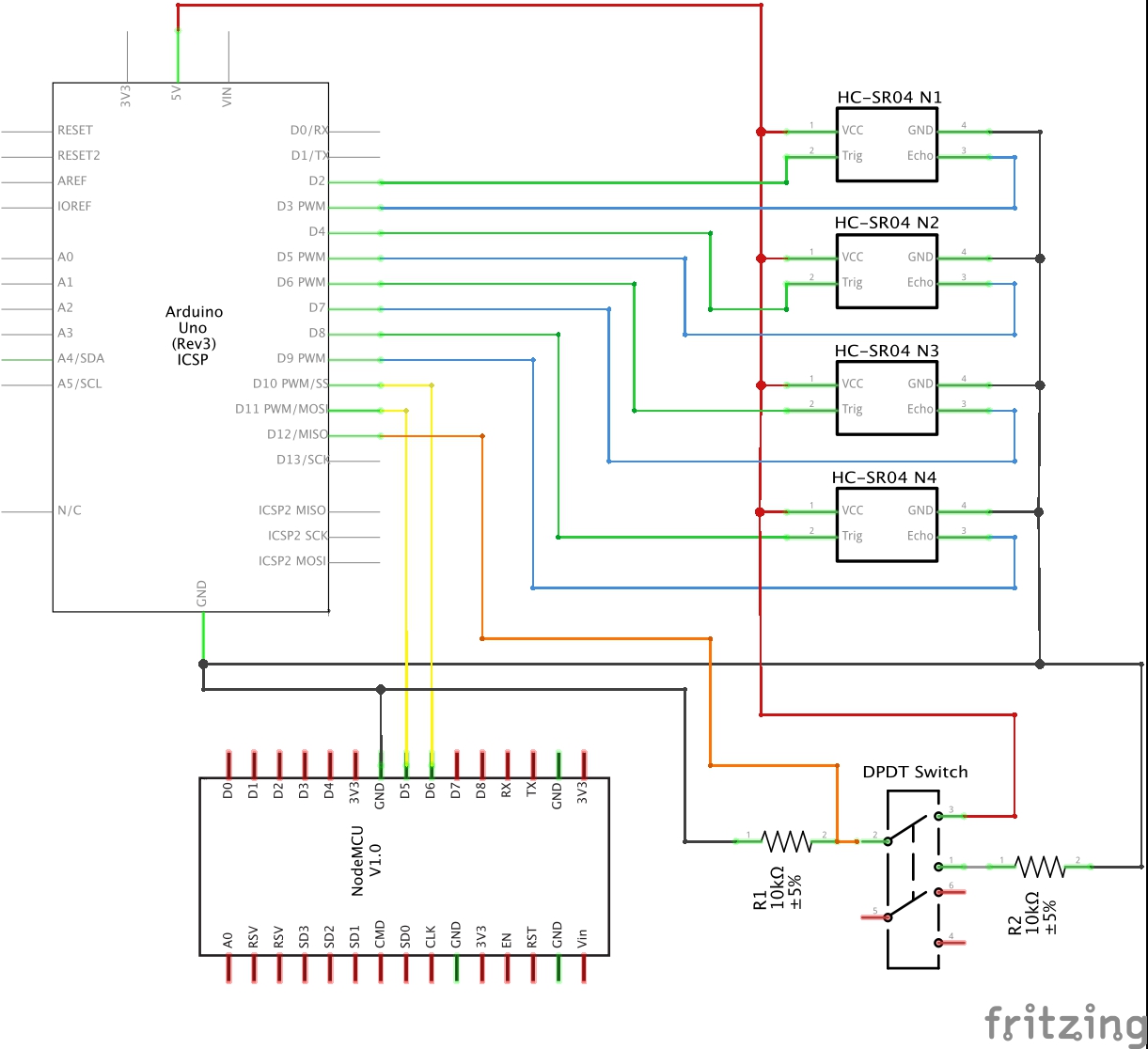
Afterwards, certain functions in the library are called to get current time and date. For instance, getFormattedDate(), getHours(), getMinutes(), getDay().

***Implementation***

The hardware implementation is divided according to the module and each module consists of two main parts: Arduino and NodeMCU.

*Reservation Free Parking Module*

As a follow up to the solution section (Sections 4.2 and 4.3), the schematic diagram in Figure 5.X shows how the module is wired along with the pin names of the components. The module components are Arduino, NodeMCU, Ultrasonic sensor and the DPDT (Double Pole Double Throw) switch.



(In hardware solution)

Ultrasonic Sensor (HC-SR04)

DPDT Switch

*The Software implementation of the hardware (Arduino and NodeMCU)*

The system’s high level architecture in Figure 4.x shows that the logic flow of the module is a one-way flow starting from Arduino to NodeMCU and this section follows the same flow accordingly.

*Arduino*

In Arduino IDE, the code is *always* divided into a setup part (executed only once) and a loop part (executed repeatedly).

*Setup*

* Arduino used pins are setup as either output or input.
* Serial connection is initiated using SerialSoftware.h library

*Loop*

The loop consists of four main steps:

* ***Step 1:*** Read the current zone from the switch button
  + Low (not pressed): CENG Female Zone
  + High (pressed): CAAS Female Zone
* ***Step 2:*** Take a reading from the four Ultrasonic sensors

For each ultrasonic sensor:

* + Step A: The trigger pin is held high for at least 10 us
  + Step B: Wait *till* the echo pin is held high (start of the pulse)
  + Step C: Measure how long the echo pin is held high. The time the echo is held high is the pulse width
  + Step D: Calculate the distance read from the pulse width.

Distance (cm) = pulse width (us) / 58.0

* + Step E: The distance is checked (Distance can be from 2-400 cm)
    - Distance less than 20 cm: Car is present; new status of the parking spot is “not available”
    - Distance more than 20 cm: Car is not present; new status of the parking spot is “available”
* ***Step 3:*** Check if any of the new readings are different than the old readings (i.e. there is a change in status)
  + Both new and old readings are stored in an array and compared with respect to the index.
    - If new = old: No change in status
    - If new != old: Change in status
* ***Step 4:*** Send changes to NodeMCU serially
  + The data sent to NodeMCU is in a JSON format with the help of ArduinoJson.h library (as explained in the database design section 4.x) and is structured as follows:

{

"zoneNumber" : 0,

"spot1" : 1,

"spot2" : 3,

"spot3" : 2,

"spot4" : 3

}

* + Zone Number: could either be 0 or 1
    - 0: CENG Female Zone
    - 1: CAAS Female Zone
  + Spot 1, 2, 3 and 4: Shows the updated status of each parking spot
    - 3: No update in status
    - 2: Status updated to “available”
    - 1: Status updated to “not available”
* ***Step 5:*** Mark the new readings as old readings
  + The old readings are deleted and the new readings are considered as old.

NodeMCU

Setup

* Connect to a nearby Wi-Fi
  + Wi-Fi’s SSID and password must be hardcoded.
* Connect to Firebase using FirebaseArduino.h library
  + As explained in section 5.x.
* Prepare the NTP Client using NTPClient.h library
  + Get current data and time from a NTP (Network Time Protocol) server using the NTP client
* Serial connection is initiated using SerialSoftware.h library.
* Arduino used pins are setup as either output or input.

Loop

The loop is divided into two parts:

Part 1: Updating parking spots status in Firebase

Part 2: Get parking spots status for histogram (Current Occupancy Trend)

*Part 1*

* ***Step 1:*** Receive changes from Arduino serially
  + NodeMCU reads the received JSON format data using ArduinoJson.h library as explained in Arduino’s Loop Step 4.
* ***Step 2:*** Check if there is at least one spot with an update.
  + If all the spot values are 3 (i.e. No update), step 3 is skipped
* ***Step 3:*** Update changes to Firebase.
  + All parking spots keys in Firebase are hardcoded.
  + As explained in Firebase section 5.x, The new status is updated using the parking spot path passed into FirebaseArduino.h library function setString(path, status)

*Part 2*

Each hour, NodeMCU updates the Firebase with the current parking spots status. This is done to get the Current Occupancy Trend available in application and website.

* ***Step 1:*** Get current time using NTP client
* ***Step 2:*** Check if an hour has passed since last update
  + 8 AM parking status is read at 8:30 AM. (To get the most efficient reading of the hour)
  + NodeMCU proceeds to step 3, if

(currentMinutes >= 30) and (currentHourUpdated != true)

* ***Step 3:*** Get current parking spots status from Firebase
  + The parking spot path is passed to the FirebaseArduino.h function getString(path)
* ***Step 4:*** Send new parking spots status to Firebase
  + For each hour in each zone, there is a count and date array, where count is the number of cars parking in that specific hour at the corresponding date in date array. To illustrate
  + To illustrate, below is an example of hour 13 at zone “CENG Female Zone”

Date: Count:

3: “2019-04-04” 3: 4

2: “2019-04-11” 2: 2

1: “2019-04-18” 1: 4

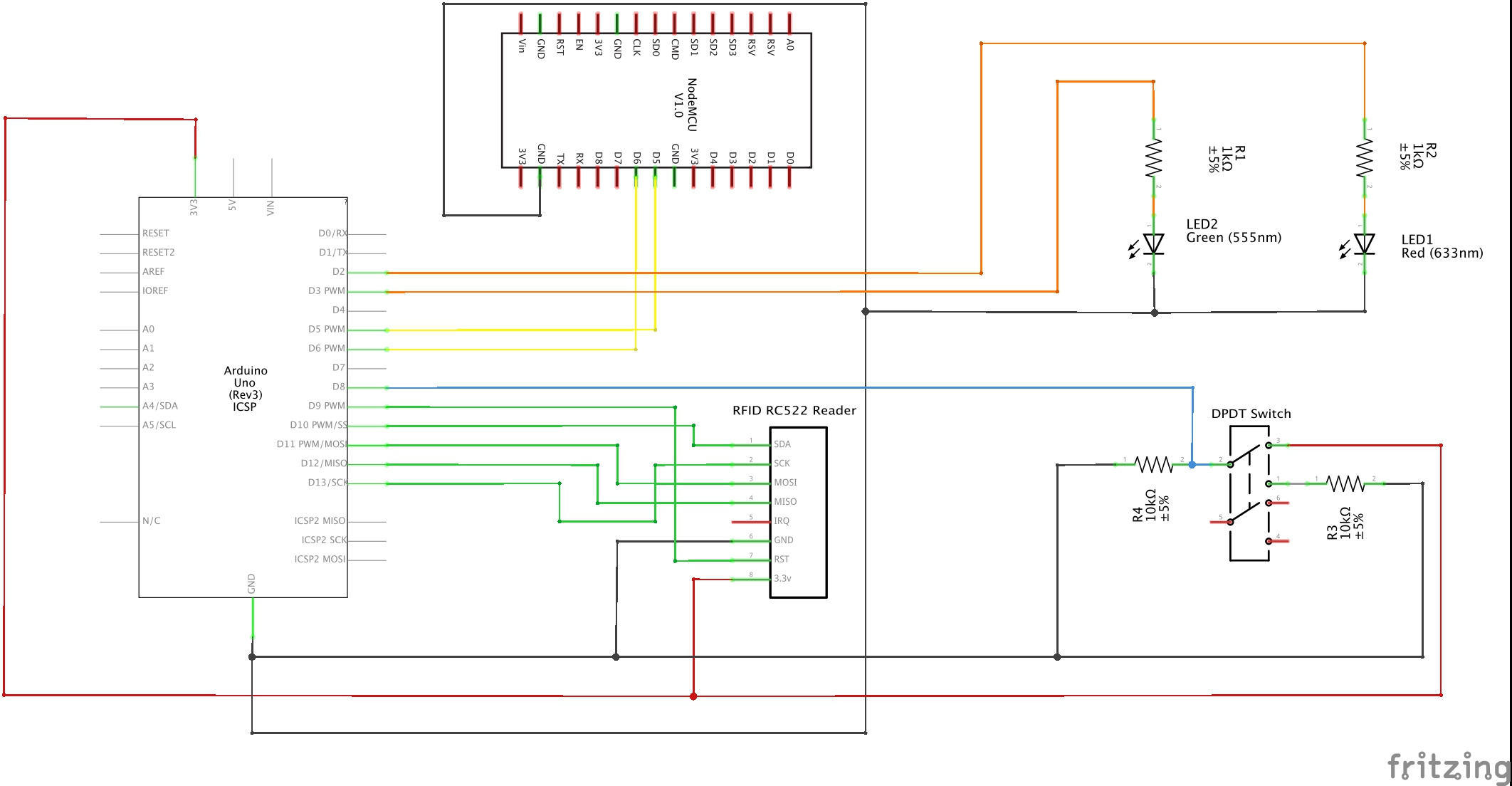
0: “2019-04-25” 0: 3

The first line means on 4/4/2019 at 1pm, there were four cars parked at CENG Female Zone.

* + To update the above structure with the new read data
    - Step A: Oldest date and count at index 3 is deleted
    - Step B: Other dates and counts are shifted upwards (i.e. 2 becomes 3, 1 becomes 2 and 0 becomes 1)
    - Step C: New date and count is added at index 1

*Reserved Parking Module*

The schematic diagram of the module is shown in figure 5.x. The components of this module are Arduino, NodeMCU, RFID reader and DPDT Switch.



*The Software implementation of the hardware (Arduino and NodeMCU)*

The reserved parking module shown in the system’s high level architecture in Figure 4.x shows that the logic flow of the module is a two-way flow starting from Arduino to NodeMCU then back to Arduino again, and this section follows the same flow.

RFID Reader

DPDT Switch

Arduino (First part)

Setup

* Arduino used pins are setup as either output or input.
* Serial connection is initiated using SerialSoftware.h library
* RFID reader MFRC522 is initiated

Loop

* ***Step 1:*** RFID Reader scans UID tag if there is any within the reading range.
  + If a UID tag is scanned and read, Arduino proceeds to step 2
* ***Step 2:*** Read the current zone from the switch button
  + Low (not pressed): LIB Female & Male Zone
  + High (pressed): CBAE Female & Male Zone
* ***Step 3:*** Send UID to NodeMCU serially
  + As explained above, the data is in JSON format when sent serially. This data is structured as follows:

{

"zoneNumber" : 0,

"A" : 01001010,

"B" : 00001111,

"C" : 01110001,

"D" : 00111111

}

* + Zone Number: could either be 0 or 1
    - 0: LIB Female & Male Zone
    - 1: CBAE Female & Male Zone
  + A, B, C and D: The four parts combined gives the tag UID, the numbers stored are in binary (0s and 1s)

NodeMCU

Setup

* Connect to a nearby Wi-Fi
  + Wi-Fi’s SSID and password must be hardcoded.
* Connect to Firebase using FirebaseArduino.h library
  + As explained in section 5.x.
* Prepare the NTP Client using NTPClient.h library
  + Get current data and time from a NTP (Network Time Protocol) server using the NTP client
* Serial connection is initiated using SerialSoftware.h library.
* Arduino used pins are setup as either output or input.

Loop

* ***Step 1:*** Receive tag UID from Arduino serially
  + Arduino reads the received JSON format data using ArduinoJson.h library as explained in Arduino’s Loop Step 3.
  + Convert binary numbered UID to a hexadecimal number, then the UID number is converted to a string
* ***Step 2:*** Get current time and date using NTP client
* ***Step 3:*** Check if there is a reservation with the received UID
  + ***Step A:*** Get all reservations with the received UID from Firebase

*The following steps are looped through all received reservations*

* + ***Step B:*** Check if
    - reservation’s zone == zone sent serially
    - reservation’s date == current date

If any of the conditions is false, NodeMCU goes to the next reservation in the loop

* + ***Step C:*** Check if reservation has started

Current time >= reservation’s start time

If it has not started yet, NodeMCU goes to the next reservation in the loop

* + ***Step D:*** Check if reservation status is “created” and reservation time has not ended

(status == “created”) & (current time <= end time)

If *both* conditions hold, NodeMCU stores the reservation key in Firebase and goes to step 4

If *first* condition fails, NodeMCU goes to step E

If *second* condition fails, NodeMCU goes to the next reservation in the loop

* + ***Step E:*** Check if reservation status is either “extended” or “arrived” or “subcancelled”

If not, NodeMCU goes to the next reservation in the loop

* + ***Step F:*** Check if Automatic Cancellation can be applied

If there is at least one hour or more left to the reservation

current hour < reservation’s end time

*if condition is true*, automatic cancellation is applied by deducting half the cancelled hours price (2.5QR/hour)

price -= (2.5 \* number of hours left)

then NodeMCU stores the reservation key in Firebase and goes to step 5

*if condition is false*, NodeMCU goes to step G

* + ***Step G:*** Check if VIP user has exceeded his reserved time

Current hour > reservation’s end time

Note: VIP user is left without any penalty in the first 5 minutes after his/her reservation

*If condition is true*, the VIP user is penalized with triple the original price for one reservation hour (15 QR per extra hour)

price += (15\* number of extra hours)

then NodeMCU stores the reservation key in Firebase and goes to step 5

*If condition is false*, NodeMCU stores the reservation key in Firebase and goes to step 5

* + ***Step H:*** If none of the received reservations goes into steps 4 or 5, then NodeMCU goes to step 6 with a negative response
* ***Step 4:*** Update reservation status to “arrived” in the Firebase
  + The status is updated using the reservation key
  + NodeMCU goes to step 6 with a positive response
* ***Step 5:*** Update reservation to “ended” in the Firebase
  + The status is updated using the reservation key
  + NodeMCU updates the reservation prices if there is any addition (Step G) or deduction (Step F), afterwards goes to step 6 with a positive response
* Step 6: NodeMCU sends a positive or negative response to Arduino serially
  + Positive: to open the gate of the parking lot
  + Negative: to not open the gate of the parking lot

Arduino (second part)

Loop

* ***Step 4:*** Wait till a response is received from NodeMCU serially
  + Positive response: Light up the green LED (i.e. Send digital high to green led pin) to indicate that the gate is opened.
  + Negative response: Light up the red LED (i.e. Send digital high to red led pin) to indicate that the gate is closed.

*When a response is received Arduino goes back to step 1 (first part)*

**Challenges**

As we implement our project, we faced many challenges that required us to solve them to meet the project’s requirements.

1. Serial communication

Challenge: In Reservation Free Parking Module, the Arduino used to send the parking spots status to NodeMCU continuously. Afterwards, NodeMCU will detect if there is any change in status. This made the serial connection slow and required 15 seconds or so for a change to be available in the application/website.

Solution: We changed how the data is sent from Arduino. Now, the Arduino only sends data serially when it detects a change, otherwise no data is sent to NodeMCU. This significantly increased the efficiency of the serial communication and the system overall.

1. NodeMCU firebase connection timeout