### **Section 1:**

### **1.1 Connecting android application to firebase cloud**

Configuration steps:

1. To have a project in Firebase cloud we have first created a Gmail account and then logged in to Firebase cloud by that account.
2. Then we created a Firebase project and registered our app with it.
3. After that, a google-service.json file which is Firebase configuration file was given to be added to the Android project. To enable Firebase products in our app, we added the following google-services plugin to build.gradle file:

* apply **plugin**: **'com.google.gms.google-services'**

1. The dependency for Realtime Database was also added to the build.gradle file:

* dependencies {  
  implementation **'com.google.firebase:firebase-database:16.0.4'** }

1. Also, we configured Firebase Database Rules. To be able to send and receive data to the database we defined the database rules of read and write as true which allows to read and write to it.
2. After doing all the above configuration, we can start reading or writing to the database.

Retrieving data from Firebase Realtime Database:

Data stored in a Firebase Realtime Database is retrieved by attaching an asynchronous listener to a database reference. The listener is triggered once for the initial state of the data and again anytime the data changes. The following table illustrate witch listener was used:

|  |  |  |
| --- | --- | --- |
| Listener | Event callback | Typical usage |
| ValueEventListener | onDataChange() | Read and listen for changes |

Inserting data to Firebase Realtime Database:

The Realtime Database accepts multiple data types **String, Long, Double, Boolean, List<Object>** to store the data. It also allows us to use **custom java objects** to store the data which is very helpful when storing model class directly in database.

For example: to store user information,

* Step1: we created User model with all properties that we decided to store
* Step2: we got the reference to **‘**users’ node.
* Step3: we used the reference to generate a unique Id by calling push() method which creates an empty node with unique key.
* Step4: we created user object
* Step5: we used the generated unique Id in step3 to push user to 'users' node

Deleting data from Firebase Realtime Database:

To delete data, we simply called removeValue() method on to database reference.

### **1.2 Connecting the website to firebase cloud**

Configuration steps:

1. Creating an account and project on firebase as mentioned in (XXX).

* After accomplishing steps above, starting with configuration steps by obtaining the config object. This required opening the project and selecting “Add Firebase to the web app”, and finally copying the snippet shown below and adding it the front-end.

var config = {

apiKey: "AIzaSyCYgApFzAtXIVgdI4J09RWseIxlqfAS8hA",

authDomain: "fir-auth-45665.firebaseapp.com",

databaseURL: "https://fir-auth-45665.firebaseio.com",

projectId: " fir-auth-45665",

storageBucket: " fir-auth-45665.appspot.com",

messagingSenderId: "777762792693",

}

1. Configuring Nodejs app by firstly installing package.json and firebase npm package after running the commands below:

npm init

npm install --save firebase

1. Using Firebase module after requiring them from JavaScript.

var firebase = require("firebase/app");

require("firebase/auth");

require("firebase/database");

1. Initializing the Realtime database and configuring the rules as mentioned in (XXX)

firebase.initializeApp(config);

// Get a reference to the database service

var database = firebase.database();

1. After doing all the above configuration, we can start reading or writing to the database.

Retrieving, inserting and deleting data in Firebase Realtime Database:

Retrieving, inserting and deleting data in the website have the same logic as application however website uses different type of listeners and functions.

The following table illustrate the type of listener used to retrieve data:

|  |  |  |
| --- | --- | --- |
| Event | Event callback | Typical usage |
| Value | On() and once() | Read and listen for changes |

### **1.1 Connecting hardware (NodeMCU) to firebase cloud**

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.

### **Section 2: Tools and frameworks used**

The Hardware tools are Arduino IDE and Fritizing. Arduino IDE is used to program both Arduino and NodeMCU, while Fritzing was used to create the hardware Connectivity and Schematic diagrams.

Additionally, for android application, we used Android Studio as it is the official IDE for Android development. The language used to develop the entire logic of the application was java. Also, XML (Extensible Markup Language) used as Markup Languages to design the user interface.

For website development, the following table illustrate all tools and frameworks that we have used in different stage of implementing our website:

|  |  |
| --- | --- |
| Tools, frameworks and technologies | |
| Front-end Languages / Platforms | |
| HTML | Hyper Text Markup Language used to build the structure and the content of the website page |
| CSS | Style sheet language used for the presentation of web pages writing in HTML. |
| JavaScript | Programming language used for client-side scripting. |
| Front-end frameworks | |
| Bootstrap | An open-source CSS framework used to make the website responsive and allow the pages’ style to act differently based on the browser size. |
| Back-end frameworks and Libraries | |
| Node.js | An open-source, JavaScript run-time environment used to execute server-side scripting. |
| Express Framework | Node.js framework used to make the HTTP requests such GET and POST requests. |
| Body-parser | Node.js middleware for handling JSON used to get the body from the request and work on it, and to get the data sent in the body. |
| Express-handlebars | Express module used to create dynamic front end pages using html. |
| Path | Node.js module used to set the css,js and images files and everything related to the frontend into public folder and access it from the pages. |
| App | JavaScript web application development framework used to Create instance form Express |
| Express-flash | Node.js module for flash messages for Express application. Used to display messages on success and on errors. |
| Express-session | Node.js module creates a middleware with the given options. We used it to save the session of the user. |
| Moment | A lightweight JavaScript date library for parsing and formatting dates. Used to make a lot of changes on the displaying, adding time or getting time difference. |
| Nodemailer | Node.js module used to send e-mail from Node.js |
| Task Runners / Package Managers | |
| NPM | Pack manager for JavaScript. |

### **Section 3: implementation**

### 3.1 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)

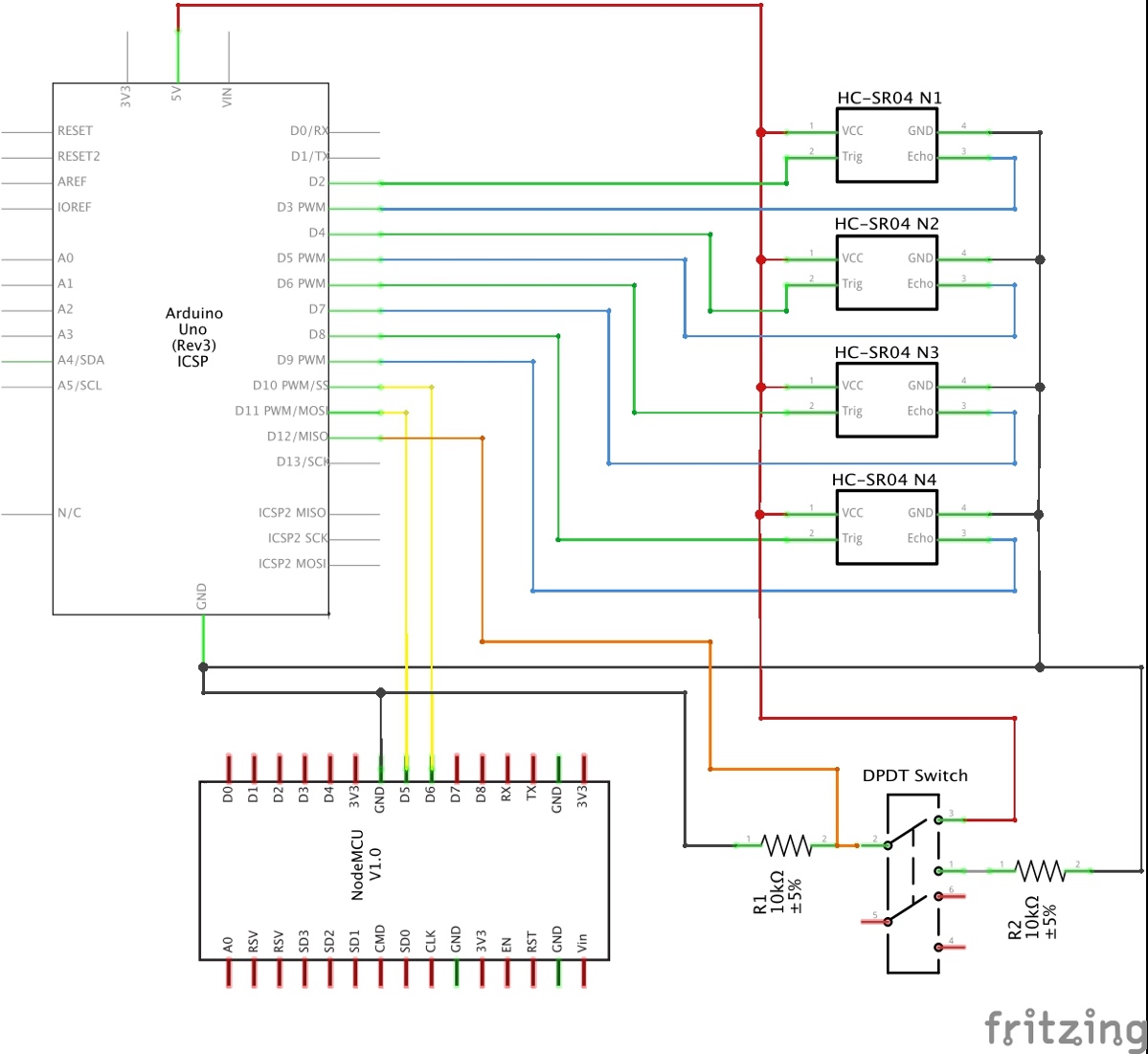
Afterwards, certain functions in the library are called to get current time and date. For instance, getFormattedDate(), getHours(), getMinutes(), getDay().

***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.



*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

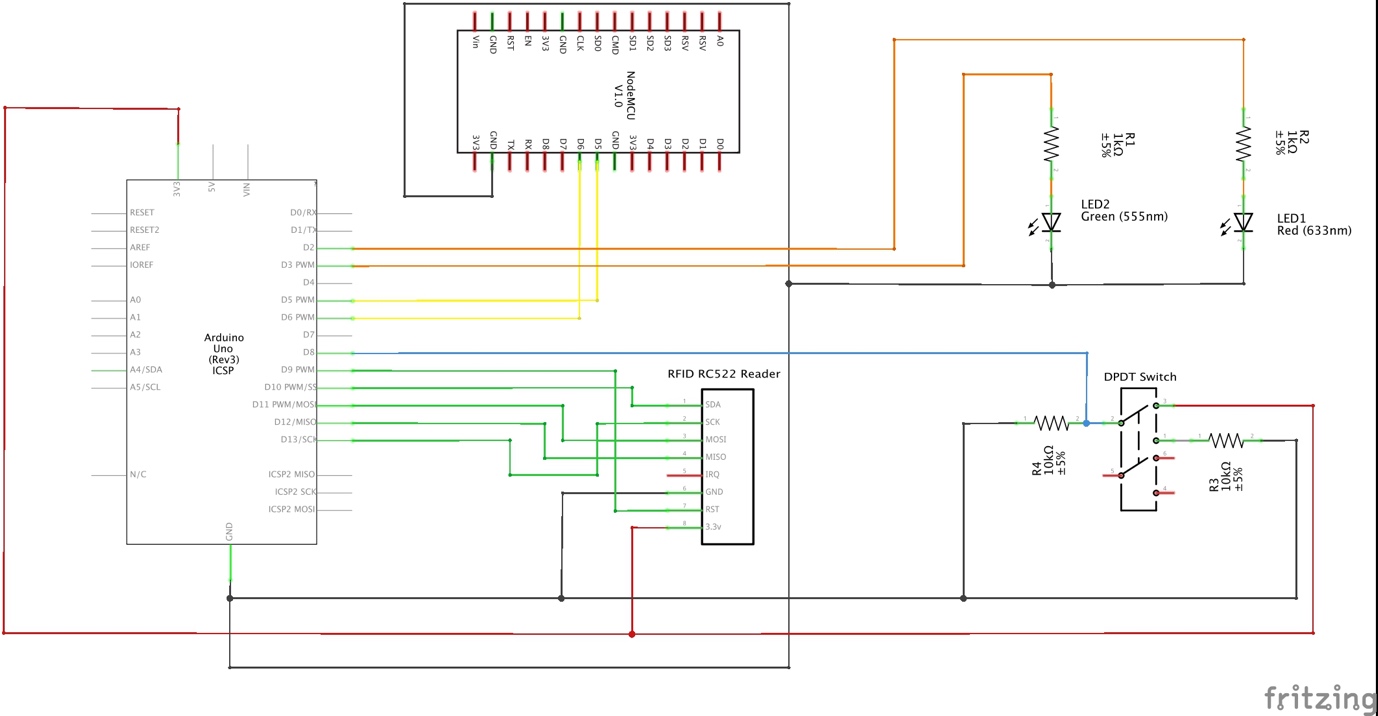
0: “2019-04-25” 0: 3

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.

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)*

### 3.2 Software implementation

The software implementation consists of many services and feature. In this section we are going to discuss the algorithm and the design for each service.

Check Availability:

* Algorithm:

Step 1: Wait till the user selects zone

Step 2: Load all the spots that are in the selected zone

Step 3: Check each spot status, if the status is “available” set the visibility of the car image to be “invisible” and if it is “not available” set the visibility of the car image to be “visible”

Step 4: Wait till the user select spot number and click on “Get Direction” button

Step 5: Provide directions to the wanted parking spot from his/her current location (device location) by using google map

* Design:

|  |  |
| --- | --- |
| Application Design: | |
| User clicks  Zone list was implemented by creating custom list view | Map was designed by using image view |

Reserve a parking:

* Algorithm:

Step 1: Wait till the VIP user selects date

Step 2: Load all the reservation that are in the selected date and the status not equal to cancel

Step 3: For each reservation, loop through the time array to find how many reservations in each hour

Step 4: For each hour find availability percentage (number of reservation in specified hour/ total allowable reservation per hour) \* 100. Based on the calculated percentage the color of each hour will be changed in the UI. Green if >75% is left*,* orange if 50% is left, red if 25% is left and gray if 0% is left.

Step 5: Wait till the VIP user selects start time and duration and clicks on reserve button

Step 6: Whenever VIP user click on the reserve button, 5 conditions will be checked

* Check if the VIP user does not have a reservation at the selected hours.
* Check if there is an available parking spot at all selected hours
* Check if selected date is equal to today or tomorrow (selected date == current date || selected date == current date +1) as the VIP user can only reserve at the same day or one day before the reservation date.
* Check if the reservation start time has passed, if selected date is today by checking if the selected start hour is greater than current hour
* Check if the number of selected hours with selected date’s reserved hours is less than or equal to the number of allowable reservation hours per day (6 hours). Reserved hours in the selected date are calculated by going through the user reservations

Total reservation hours += number of hours in a reservation – number of hours extended – number of hours cancelled

If one of these conditions is not met an error message will appear to the VIP user

Step 7: If all the above conditions are met then a reservation record with status = “created” and price = 5 QR per each hour will be added to the database.

Step 8: Notifies the VIP user 30 minutes before expiry time

Show reservations:

* Algorithm:

Step 1: Load all current and upcoming reservations related to the VIP user

Loop through all reservations

* If reservation date > current date (which mean no need to check the time)

Check if (reservation plate number == user logged plate number && status != “cancelled” && status != “subcancelled”)

* If reservation date == current date ( time need to be checked)

For each reservation check if (reservation plate number == user logged plate number && status != “cancelled” && status != “subcancelled” && current time <= reservation start time)

Extend a reservation:

* Algorithm:

Step 1: Wait till the VIP user selects extend option

Step 2: Whenever VIP user click on extend option, 2 conditions will be checked

* Check if the selected reservation is at its last hour (current hour == last hour of reservation) as the VIP user can only extend his reservation in the last hour of the reservation
* Check if there is an available parking spot after the reservation time

If one of these conditions is not met an error message will appear to the VIP user

Step 3: If all the above conditions are met then extend reservation by updating the following field in the database: status changed to “extended”, price incremented by 5 QR and the number of hours extended is incremented by 1

Cancel a reservation:

* Algorithm:

Step 1: Wait till the VIP user selects cancel option

Step 2: Whenever VIP user click on extend option, 2 conditions will be checked

* If the reservation has not started (current time < start time), cancel the whole reservation by updating the following field in the database: status changed to “cancelled”, price decremented by 50 % and number of hours cancelled incremented based on reservation hours
* If the reservation has started (current time >= start time), cancel remaining reservation hours by updating the following field in the database: status changed to “subcancelled”, price decremented by 50 % for each cancelled hour and number of hours cancelled incremented based on number of hours that are cancelled

Request car care:

* Algorithm:

Step 1: Wait till the VIP user selects one of the car care services options

Step 2: Whenever VIP user click on extend option, 2 conditions will be checked

* If the application is installed in the phone, VIP user is successfully redirected to the Servesni application (talk about it in the functional requirement section)
* If the application is not installed in the phone VIP user redirected to the Play Store

Availability percentage:

* Algorithm:

Step 1: Load all the zones from database

Step 2: For each zone the availability percentage of spots is calculated by looping through all the spots and count how many spots are available

Step 3: Availability percentage is computed by following equation

availability percentage = (number of available spots (counted in step1) / total number of spots in zone) \*100

Step 4: Based on the computed percentage the progress bar color will be changed. Green if more than 75% of the spots are available, orange if 50% of the spots are available and red if 25% of the spots are available.

Currently looking:

* Algorithm:

We expand our database to have new relation called currently looking which has id, login time and zone name.

Step 1: When user click to view a zone, currently looking record will be added to the database

Step: 2 When user leave zone page, currently looking record will be deleted from the database

To handle the lost connection case, we applied the following logic:

Step 1: For each zone, load all the currently looking records from database

Step 2: Remove all records older than 10 minutes

Step 3: Show how many people are currently viewing each zone

* Design:

Current Occupancy Trend:

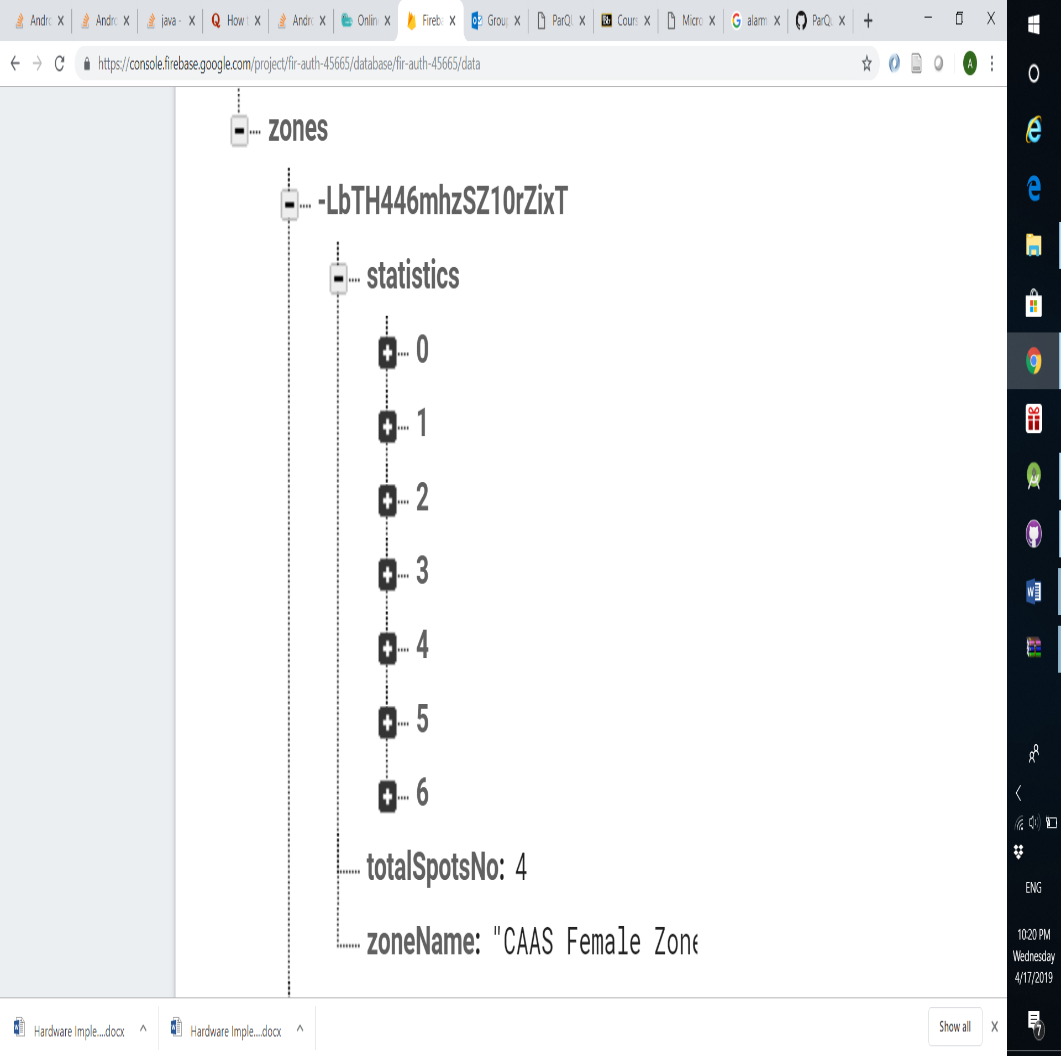
Structure of statistics field for each zone in database

Statistics is an array of object where each object storing the data for week days (Sat, Sun, Mon, etc...) (see figure)

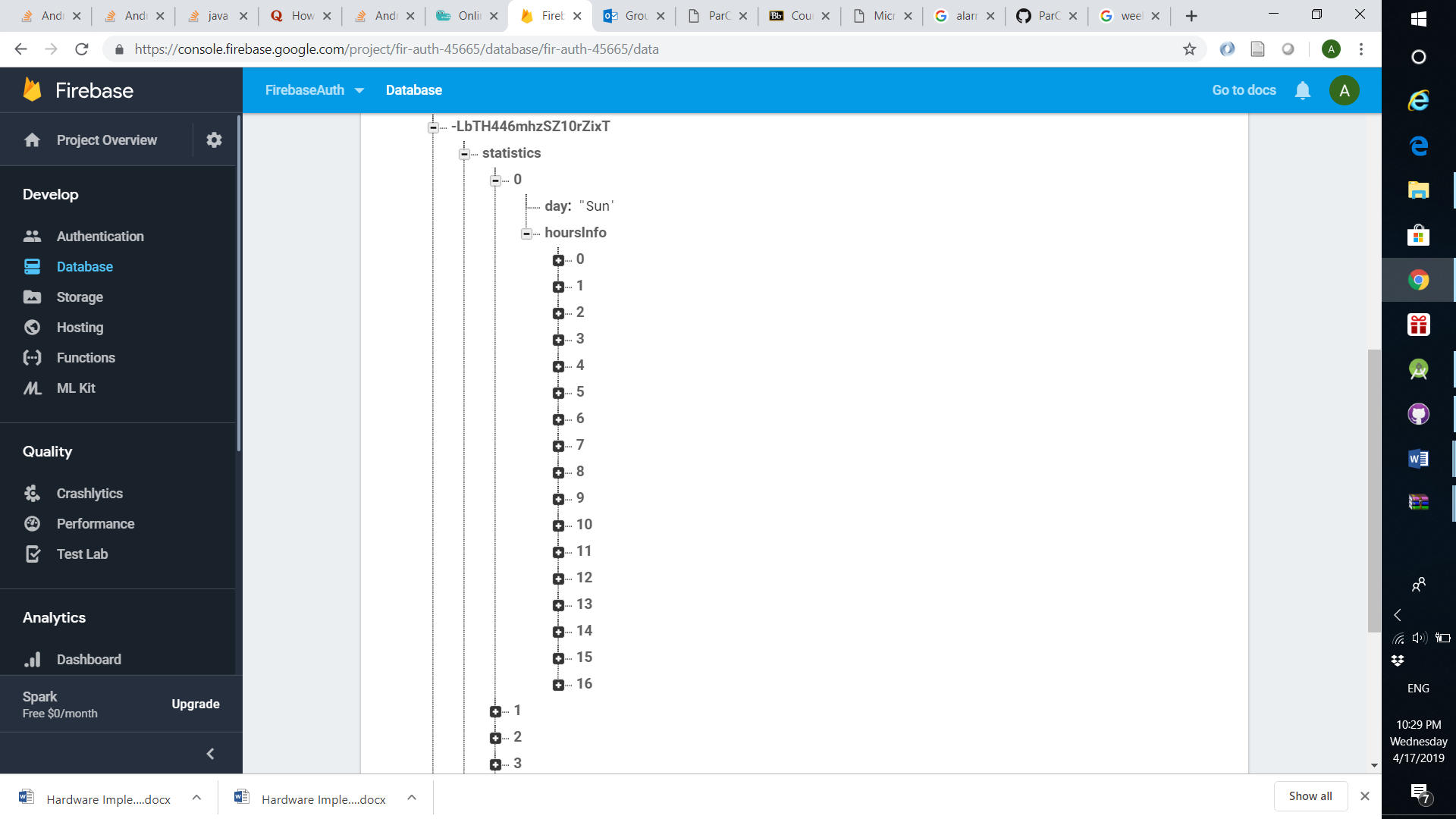
Each day object will have field called hoursInfo which is an array of object where each object storing the data for each hour (6 AM – 9 PM). For example, the first index (0) store data for 6 AM (see figure). Each hour object stores 3 data which are hour, count array and date array where count is the number of reservations in that specific hour at the corresponding date in date array. For example, in 2019-03-24 there is no reservations at 6 AM. We only storing the data for recent 4 (Sunday) in weeks







Zooming in first day object



Zooming in first hour object



Updating statistics data for reservation zones

* Algorithm:

Step 1: Check which week of a day the date of reservation belongs to

Step 2: Update all hours that are equal to reservation hours in that date by the following logic

* if date exist in date array, increment the correspond count of date by 1
* if date is not existing, remove the oldest date corresponding with its count value (always in index zero), shift other dates and counts upwards then finally add the new date and count

Updating statistics data for check availability zones

* Refer to section

Displaying data in term of histogram:

To display data as bar chart we used an Android chart view library called MPAndroidChart

* Algorithm:

Step 1: Wait till user selects day of a week

Step 2: For each hour, occupancy percentage is calculated as following

% Occupancy = (total sum of count values / (number of dates \* total number of spots)) \* 100

Step 3: Use setData function from MPAndroidChart to set the calculated percentage for each hour into bar chart

* Design:

### **Section 4: challenges**

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

1. Currently looking implementation

Challenge:

It was quite challenging to implement this feature as the implementation logic in the application was different from the website. In the website, it was a bit easy to know how many one visit specific page by using IP address. However, this logic is impossible to be implemented in the application.

To implement this feature in the application, we benefit from a concept called "activity life cycle" where each activity goes through several stages. we benefited from onStart () and onStop (). whenever the activity gets started, onStart () method will be invoked so we should place the logic needed for increasing the number of people looking inside the onStart () method. In contrast, the logic needed for decrementing the number of people looking was placed in onStop () as this method will be called whenever the user leaves the page (activity).

After implementing this logic, we figure out that it will not work properly in the case that the user lost internet connection. Logically if the user lost the connection the number of people looking should be decremented by 1, however in reality the number stay as it is. The reason behind that was the database has not been updated as there is no internet connection.

Solution:

To solve this issue, we thought about adding new relation called “currently looking” in the database to store zone name, login time and temporary id for each access. Whenever the user closes the page, the stored record for this user will be removed. Whenever other users visiting page, 2 things will happen:

1. All the records that are older than 10 minutes will be removed.
2. Show users how many people are currently looking.

The logic used to solve this issue discussed above, refer to section ()

By applying this logic, we can give an approximate number of how many currently looking, as in some rare cases the user might stay looking in the page more than 10 minutes.

1. Current Occupancy Trend implementation

Challenge: We calculated the occupancy percentage based on all the history data stored in the database so with time this will affect the accuracy and not reflect what really happen in real life as any change in data is not sensitive after a long duration.

Solution: We decide to apply moving average concept by calculating the occupancy percentage based on the data of recent 4 weeks

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.