### **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 to firebase cloud**

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

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 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

### 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 |

|  |
| --- |
| Website Design: |
| HTML section contains list of zones’ images, with buttons appears on hovering for more options. |
| Map implemented by using Modal that pops up after “show Abaialbilty” buttons get cliked by a user, so he can get back to another zone without leaving the page. |

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

|  |
| --- |
| Website Design: |
| This page mainly implemented by using HTML forms,buttons and inputs.  The left side buttons’ colors implemnted to be changed based on avaiabilty perncetage of the parking at the date and the zone that user selects. |
| If reservation created successfully, the user will be redirect to “My Reservation” page. Only current and upcoming reservation will be listed with two buttons for more options. |

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

|  |
| --- |
| Website Design: |
| In the same page “ My Resrvation”, a success or error message implemented by using **Express Flash** will be shown to the user. |

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

|  |
| --- |
| Website Design: |
| A pop up box for confirmation implmented by JavaScript method **window.confirm()** |
|  |

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.

|  |
| --- |
| Website Design: |
| Availability progress bar implemented using Bootstrap CSS |

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:

|  |
| --- |
| Website Design: |
|  |

Current Occupancy Trend:

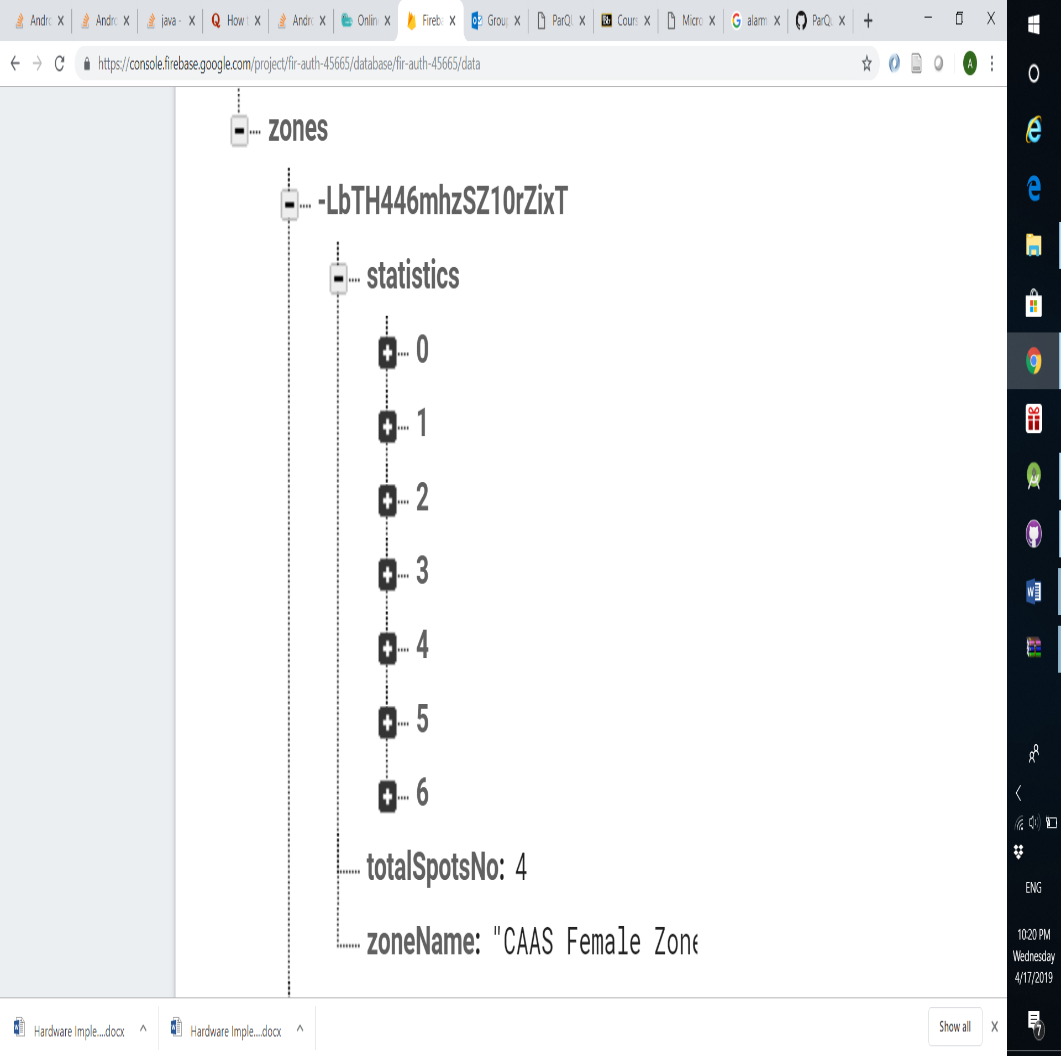
Structure of statistics field for each zone in database

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

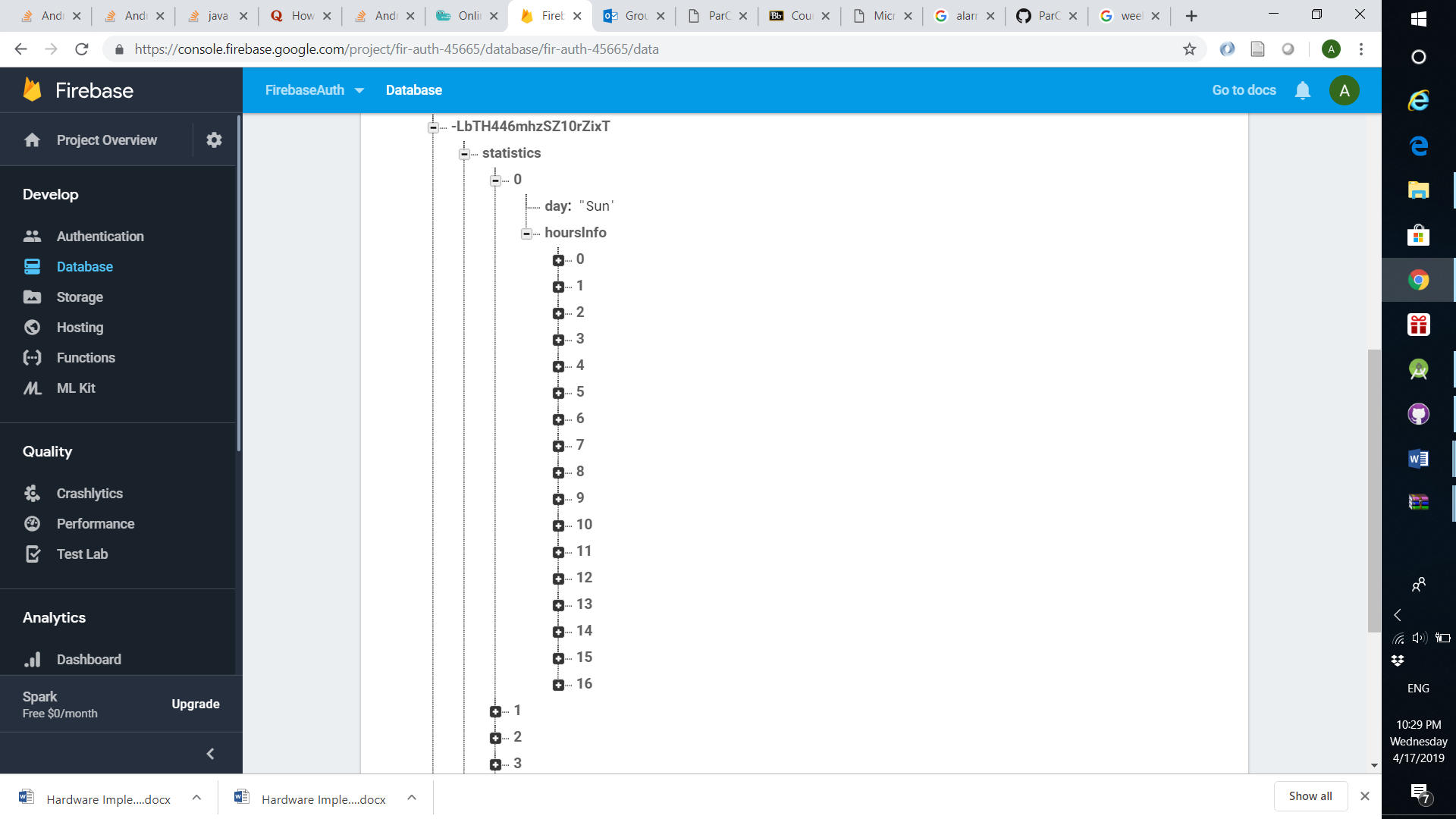
Each day object will have field called hoursInfo which is an array of object where each object storing the data for each hour (6 AM – 9 PM). For example, the first index (0) store data for 6 AM (see figure). Each hour object stores 3 data which are hour, count array and date array where count is the 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 and Amcharts javaScript for website to set the calculated percentage for each hour into bar chart.

* Design:

|  |
| --- |
| Website Design: |
| Availability progress bar implemented using Bootstrap CSS |

### **Section 4: challenges faced**

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