**6. Testing**

In this section, several test mechanisms are applied to the prototype to verify that it has met the functional requirements and the design constraints mentioned in section 3. This section also proves how our prototype provides the solution to the problem stated in section 1.

**6.1. Functional Testing**

Functional testing is part of black box testing that tests the system functionalities without inspecting the internal process. The test ensures that the functional requirements of the system has been met.

(paste table here)

**6.2. Connectivity Testing**

The connectivity test measures the average time it takes for a connection to pass data between two certain components in the system. This test shows to which extent is our system real time and in what speed. The whole tests were done in the same environment, internet connection and around the same time. For each type of connection, the average time is computed by (4 محاولات (. we were able to compute time by following this algorithm:

* Step 1: read system time and store it in a variable called startTime
* Step 2: write code
* Step 3: read system time again and store it in variable called endTime
* Step 4: subtract the startTime from endTime and display result in the console

The test is done through several steps:

* Between Arduino and NodeMCU
* Between NodeMCU and Firebase
* Between Application and Firebase
* Between Website and Firebase

6.2.1. Between Arduino and NodeMCU

The serial communication between the Arduino and NodeMCU is tested by measuring the time the Arduino or NodeMCU takes to transfer or receive data. This test transfers a 131 bytes JSON object from Arduino to NodeMCU and vice versa. The time taken for each trial is specified as follows

6.2.2. Between NodeMCU and Firebase

The internet connection between NodeMCU and Firebase is tested by measuring the time NodeMCU needs to read from or write to the Firebase database.

The system requires NodeMCU to deal with two main operations when communicating with the Firebase. First operation is to get all reservations in the database and second operation is to get or update (set) a specific value in a node. Both operations have different data sizes and thus the time taken for each operation differ.

1st Operation: Get reservations (Node of Objects)

|  |  |
| --- | --- |
| NodeMCU connection time to Firebase | |
| Trials | **Get Reservation**  **(177 Bytes)** |
| 1 | 0.350258 s |
| 2 | 0.401654 s |
| 3 | 0.330333 s |
| 4 | 0.331711 s |
| Average | **0.353489 s** |

2nd Operation: Update a value

|  |  |  |
| --- | --- | --- |
| NodeMCU connection time to Firebase | |  |
| Trials | **Get a value**  **(15 Bytes)** | **Set a value**  **(15 Bytes)** |
| 1 | 0.306381 s | 0.383343 s |
| 2 | 0.295115 s | 0.390443 s |
| 3 | 0.296623 s | 0.387329 s |
| 4 | 0.306095 s | 0.393172 s |
| Average | **0.301054 s** | **0.388572 s** |

6.2.3. Between Application and Firebase

The internet connection between Application and Firebase is tested by measuring the time Application needs to read from or write to the Firebase database for several data sizes.

|  |  |  |
| --- | --- | --- |
| *Application Average Connection Time to* | | |
|  | **Post** | **Get** |
| Reservation (177 bytes) | 0.011777 s | 0.007429 s |
| Currently Looking (52 bytes) | 0.012627 s | 0.007296 s |
| User (113 bytes) | 0.012112 s | 0.007586 s |
| Zone (8606 bytes) | 0.017236 s | 0.007666 s |
| Spot (49 bytes) | 0.011526 s | 0.007123 s |

In general, we observed that reading data from firebase database is more faster than posting data in the firebase regardless to the number of bytes which is good as our system should be in real time which mean that any changes in the data will be fetched and reflected to the UI very quickly.

6.2.4. Between Website and Firebase

To test internet connection to and from Application and Firebase in several data sizes

**6.3. Outdoor Testing**

ParQU is a system that should be able to be implemented outdoor in a real parking area. As an initial test to verify that our sensors are able to withstand the harsh environments of being outdoors in Qatar University and satisfy its purpose, we tested the Ultrasonic Sensor outdoors to check the availability of a parking spot.

The sensor was placed below the parking spot. Figures 6.x and 6.x shows our test accompanied with the database showing the current status of the parking spot. Figure 6.x the parking is empty so the database shows that the status is “available”, while Figure 6.x has a car so the database shows that the current status is “not available”.

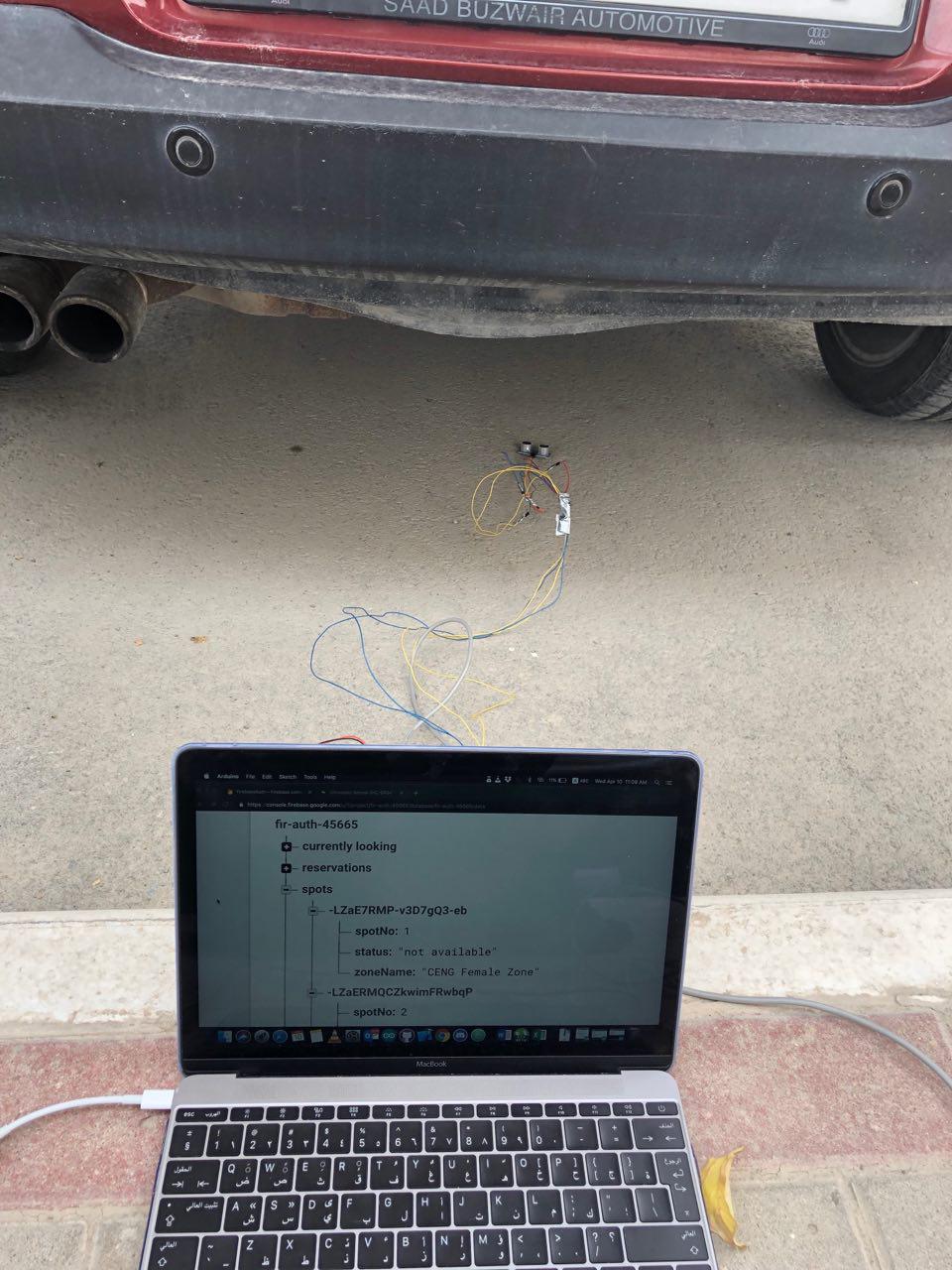
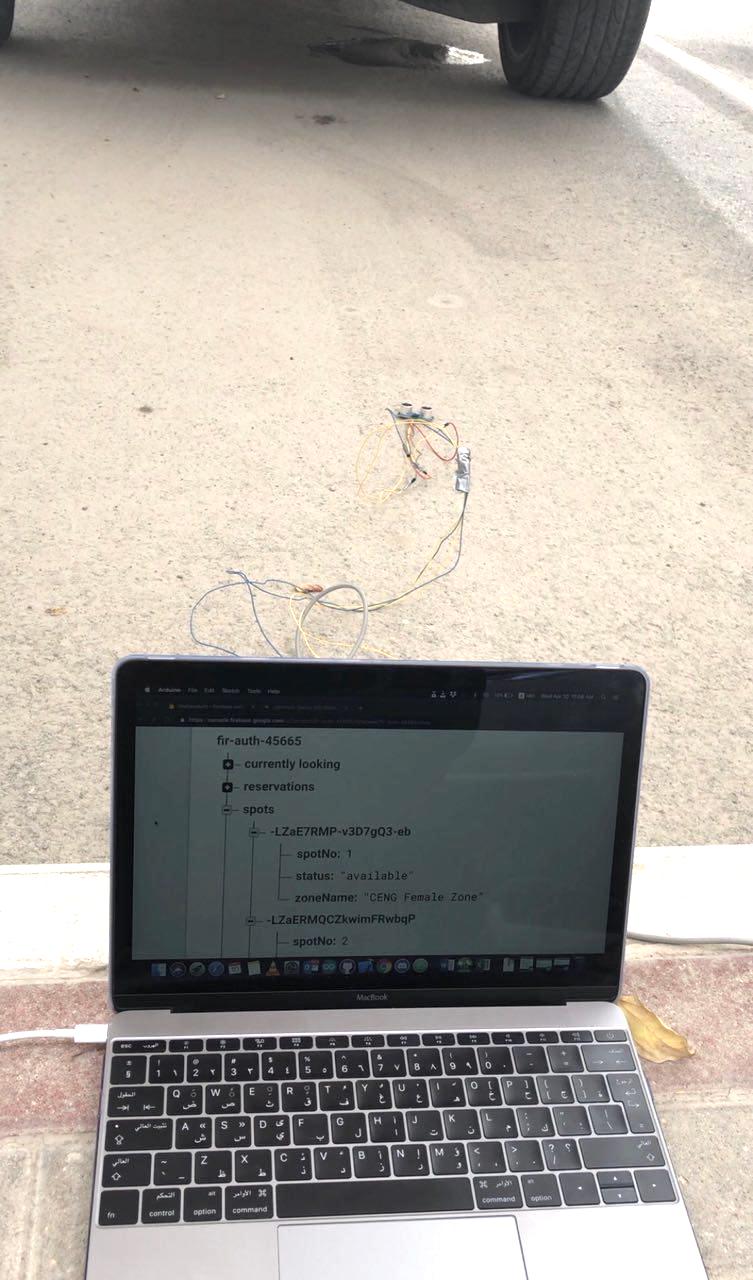
****

Figure 1 Status "not available"

Figure 1 Status "available"

Where to install the sensor?

Above the parking spot

Distance: considers the shortest possible car distance as the limit.

Pros:

Cons: Needs a parking spot shade to place the sensor under it.

Below the parking spot

Distance: considers the highest possible car distance as the limit

Pros:

Cons: Easily subjected to dirt (Sensor needs some kind of shield)

In front the parking spot

Distance: Farthest possible distance a car could park as the limit

Pros:

Cons: Cars may bump the sensor, needs a stand (Costly)

On the side (on the middle)

Distance: Smallest possible car width distance as the limit

Pros:

Cons: Cars may bump the sensor, needs a stand (Costly)

**6.4. Acceptance Testing**

(To do now: write intro to the subsection)

(To do after getting feedback from professor: do the test, write review on the results)

**6.5. Design Constraints Evaluation**

(paste table here)