**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 a certain component in the system and Firebase. 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. The time was recorded by the following algorithm:

* Step 1: read system time and store it in a variable called startTime
* Step 2: Preform needed operation with Firebase
* 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

Four trials were computed for each connection, and then the average test time was calculated.

The test is done through several levels:

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

6.2.2. Between NodeMCU and Firebase

The internet connection between NodeMCU and Firebase was tested by measuring the time NodeMCU needed 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. Table 6.x shows the final results, and the detailed test is provided in the appendix.

|  |  |  |
| --- | --- | --- |
| NodeMCU *Average C*onnection Time to | |  |
|  | **Get** | **Set** |
| Reservation (177 bytes) | 0.353489 s | - |
| Update a value (15 bytes) | 0.301054 s | 0.388572 s |

6.2.3. Between Application and Firebase

The internet connection between Application and Firebase was tested by measuring the time Application needs to read from or write to the Firebase database for several data sizes. Table 6.x shows the final results, and the detailed test is provided in the appendix.

|  |  |  |
| --- | --- | --- |
| *Application Average Connection Time to* | | |
|  | **Post** | **Get** |
| Reservation (177 bytes) | 0.012112 s | 0.007586 s |
| Currently Looking (52 bytes) | 0.011627 s | 0.007296 s |
| User (113 bytes) | 0.011777 s | 0.007429 s |
| Zone (8606 bytes) | 0.017236 s | 0.008666 s |
| Spot (49 bytes) | 0.011526 s | 0.007123 s |

In general, we observe that reading (get) data from Firebase database is much faster than posting data in the Firebase regardless of the data size. The results are to our advantage since our system is real time, we care more about fetching updated data from the Firebase and reflecting it to the UI as quick as possible than we care for sending data to the Firebase.

6.2.4. Between Website and Firebase

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

|  |  |  |
| --- | --- | --- |
| *Website Average Connection Time to* | | |
|  | **Post** | **Get** |
| Reservation (177 bytes) | 0.4838 s | 0.4242 s |
| Currently Looking (52 bytes) | 0.3824 s | 0.3690 s |
| User (113 bytes) | 0.4512 s | 0.4048 s |
| Zone (8606 bytes) | 0.8122 s | 0.8303 s |
| Spot (49 bytes) | 0.3306 s | 0.3298 s |



In average, the total time needed to read/write to and from Firebase is considered a small number, which is good for our system as it is in real time. However, the website needed more time to fetch data compared to the application. And significantly, we can find that time is directly proportional to the number of bytes.

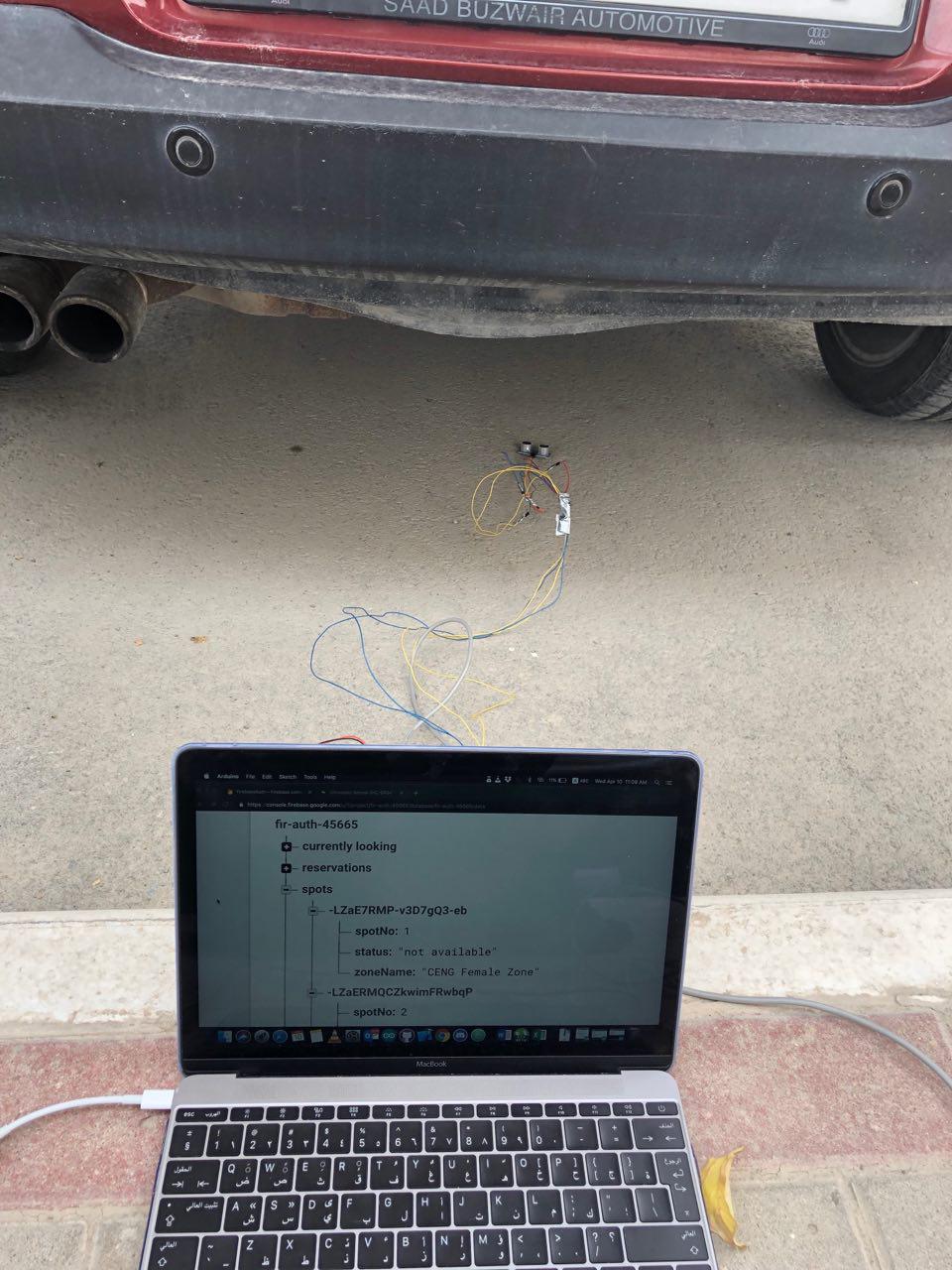
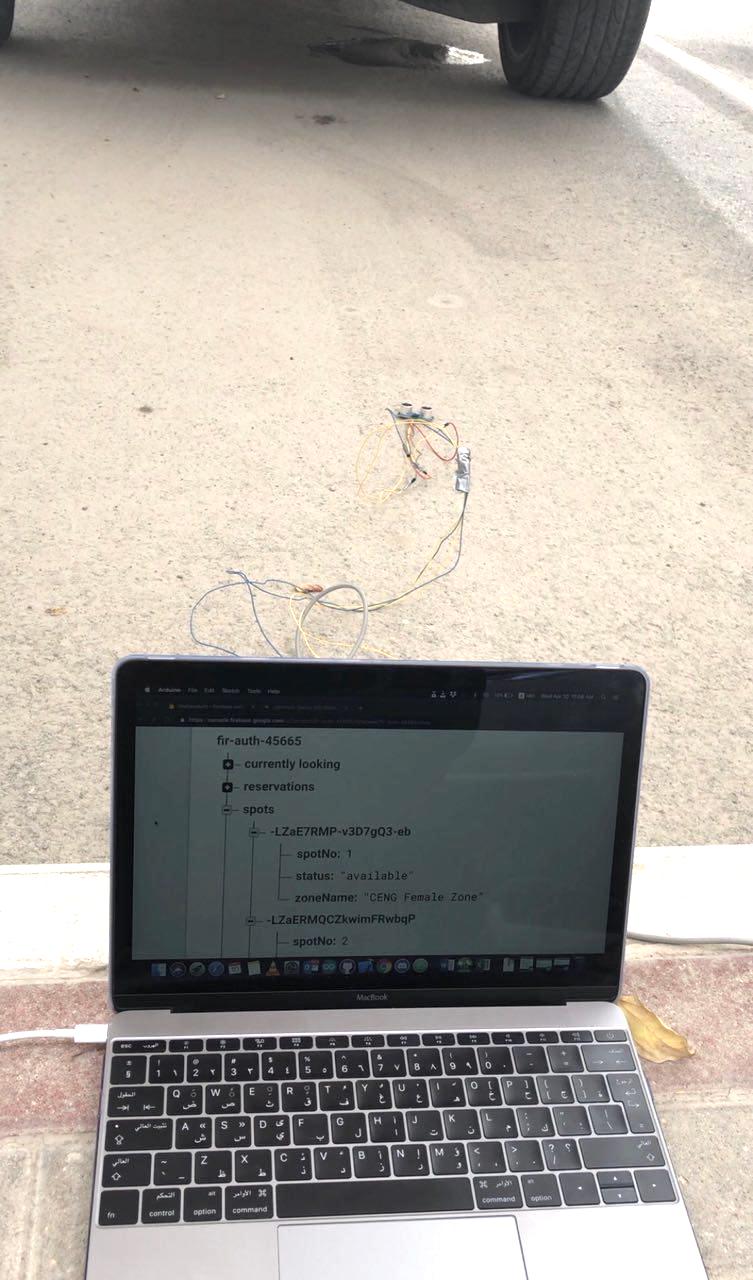
**Method of measuring time:**

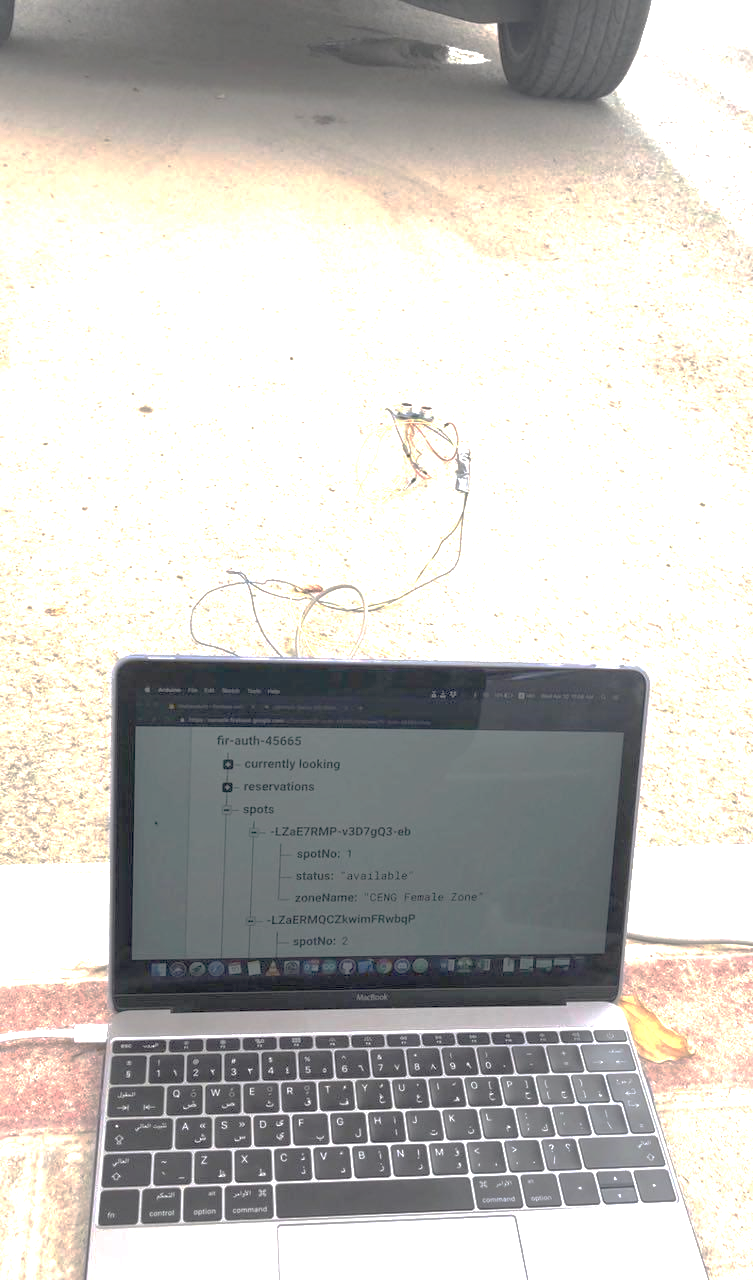
Using Postman as a testing tool, by firstly creating new routes for reading/writing without displaying pages content and applying GET/POST requests using Postman as it automatically calculates the time it took for the response to arrive.

**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”.

****



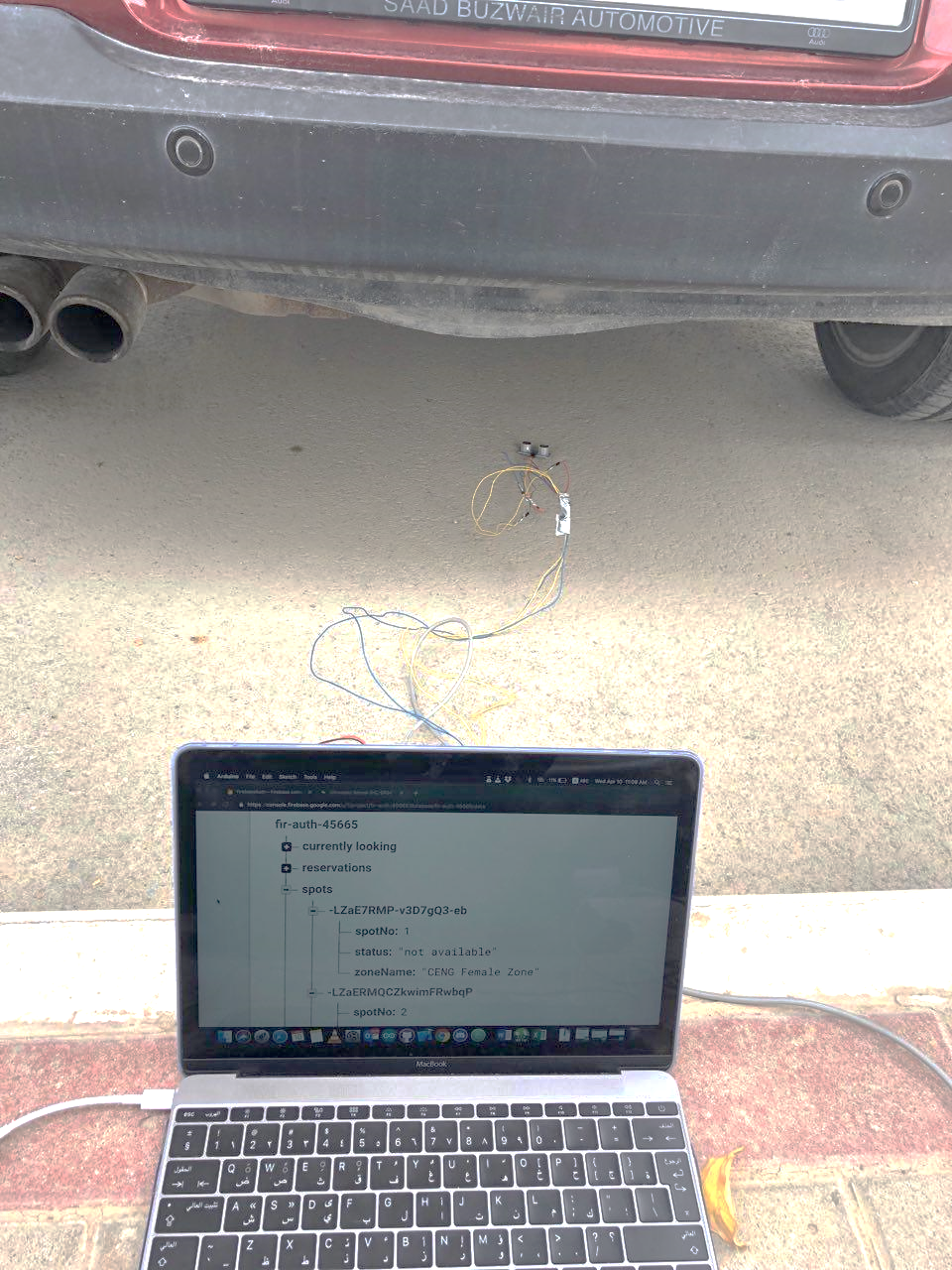
****

Figure Status "available"

Figure Status "not available"

Furthermore, when the system is implemented in real life, the sensor installation positions must be studied to pick the best position with the environment taken into consideration. Table 6.x shows the different positions a sensor could be placed in real life along with its pros and cons.

Possible installation positions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Position in a parking spot | Distance limit | Where | Pros | Cons |
| Above | Shortest possible car height | Indoors and outdoors | - Sensor protected from rain and dirt.  - Less susceptible to breaking | - Needs a parking shade to place the sensor under it |
| Below | Highest possible car height | Indoors | - Cheapest | - Easily subjected to dirt if outdoors  - Easily breakable (Sensor needs some kind of shield which is costly) |
| In front | Farthest possible distance a car could park | Indoors and outdoors |  | -Cars may bump the sensor (Sensor needs some kind of shield which is costly)  -needs a stand (Costly)  -Most expensive |

For indoors parking lots, the most suitable sensor position is above the parking spot because the sensor will be less susceptible to breaking and the parking spot will always have a ceiling to place the sensor under it. On the other hand, outdoors parking lots best position can either be above or below the parking spot depending on the environment. Placing the sensor above the parking has a huge benefit compared to below the parking spot, as the sensor will be protected from rain and dirt. However, if the parking lot does not have a shade that is already implemented, then placing the sensor above can be expensive compared to placing it below the parking spot. With that said, installing the sensor below a parking spot will require a shield to protect the sensor from being bumped by cars or covered with dirt.

**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**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Met/Not Met** | **How the design was met?** |
| Availability | The system should always be available and real time. |  | The android application and website can be available at all times given that the devices accessing the application or website is connected to the internet. Also, for the system to work, NodeMCU must be connected to the internet.  The system is real time by using Firebase Realtime Database that allows any change in the sensors to be immediately updated in the database, which in turn automatically displays the changes in the android application and website without the need to refresh the page |
| Performance | The system should complete the required functionalities within a short response time, providing accurate readings from sensors and efficient information from Android application and website. |  | Refer to section |
| Reliability | No data loss is allowed. |  | NodeMCU, Arduino, Android and the website do not save any data locally and automatically save data in the Firebase, hence if any malfunction occurs, the data will not be lost. |
| Connectivity | NodeMCU needs to be connected to the internet to collect updated data from the sensors through the Arduino board then save it in the Firebase. In addition, the mobile application and website needs to be connected to the internet to get the data from the Firebase |  | Refer to section |
| Scalability | The system can support the addition of sensors and components as well as having more users and the cloud can be upgraded accordingly. |  | ParQU can handle more VIP users with more RFID tags. Additionally, zones can be added with more components (In our prototype, a switch button is used to illustrate the scalability of our system, as explained in section (-)). All of the previous additions require more database storage, the Firebase Realtime Database allows us to store up to 1GB for free (Free Plan). Any more storage would require us to upgrade to higher plans with a specific pricing. (reference) |
| Mobility | The system can be accessed from many different platforms. |  | Website can be accessible from any type of devices and platforms such as iOS, Android and Windows, etc. Android applications can be accessible from Android devices only whether it was a mobile or tablet. |
| Power | Power source needed for:  Motor: 3-7V  Sensor: 5V  Arduino: 7-12V |  | Arduino is supplied by a 9V battery. Sensors and Servo Motor are both supplied by Arduino |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quality | Name | Description | Met/Not Met | Verification |
| Economic | Design Cost | The prototype uses high quality component with affordable prices |  | The whole prototype costed us on average 550 QR (Prototype consists of two zones, each zone has four parking spots) |
| Social | Usability | A normal user with minimal software knowledge should be able to use the mobile application and the website with ease |  | Refer to section 6.3 Acceptance testing |
| Sustainability | Maintenance | The system components should be easy to replace, remove and implement. |  | The components could easily be replaced by uploading our code to a new component. |
| Quality | Performance | The system should provide efficient information and accurate readings from the parking area. |  |  |

Appendix

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** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post User**  **(113 Bytes)** | **Get User**  **(113 Bytes)** |
| 1 | 0.011711 s | 0.007428 s |
| 2 | 0.011810 s | 0.007435 s |
| 3 | 0.011822 s | 0.007423 s |
| 4 | 0.011734 s | 0.007430 s |
| Average | **0.011777 s** | **0.007429 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Currently Looking**  **(52 Bytes)** | **Get Currently Looking**  **(52 Bytes)** |
| 1 | 0.011628 s | 0.007298 s |
| 2 | 0.011623 s | 0.007295 s |
| 3 | 0.011625 s | 0.007294 s |
| 4 | 0.011630 s | 0.007299 s |
| Average | **0.011627 s** | **0.007296 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Reservation**  **(177 Bytes)** | **Get Reservation**  **(177 Bytes)** |
| 1 | 0.012106 s | 0.007584 s |
| 2 | 0.012110 s | 0.007590 s |
| 3 | 0.012109 s | 0.007588 s |
| 4 | 0.012115 s | 0.007583 s |
| Average | **0.012112 s** | **0.007586 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Zone**  **(8606 Bytes)** | **Get Zone**  **(8606 Bytes)** |
| 1 | 0.017234 s | 0.008668 s |
| 2 | 0.017236 s | 0.008664 s |
| 3 | 0.017239 s | 0.008663 s |
| 4 | 0.017235 s | 0.008666 s |
| Average | **0.017236 s** | **0.008666 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Spot**  **(49 Bytes)** | **Get Spot**  **(49 Bytes)** |
| 1 | 0.011525 s | 0.007122 s |
| 2 | 0.011524 s | 0.007121 s |
| 3 | 0.011526 s | 0.007120 s |
| 4 | 0.011528 s | 0.007124 s |
| Average | **0.011526 s** | **0.007123 s** |

|  |  |  |
| --- | --- | --- |
| *Website connection time to* | | |
| Trials | **Post Reservation**  **(177 Bytes)** | **Get Reservation**  **(177 Bytes)** |
| 1 | 0.490 s | 0.411 s |
| 2 | 0.445 s | 0.420 s |
| 3 | 0.480 s | 0.451 s |
| 4 | 0.482 s | 0.423 s |
| 5 | 0.522 s | 0.416 s |
| Average | **0.4838 s** | **0.4242 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Currently Looking**  **(52 Bytes)** | **Get Currently Looking**  **(52 Bytes)** |
| 1 | 0.383 s | 0.370 s |
| 2 | 0.370 s | 0.386 s |
| 3 | 0.313 s | 0.406 s |
| 4 | 0.410 s | 0.360 s |
| 5 | 0.436 s | 0.323 s |
| Average | **0.3824 s** | **0.3690 s** |

|  |  |  |
| --- | --- | --- |
| *Website connection time to* | | |
| Trials | **Post User**  **(113 Bytes)** | **Get User**  **(113 Bytes)** |
| 1 | 0.489 s | 0.418 s |
| 2 | 0.409 s | 0.373 s |
| 3 | 0.456 s | 0.436 s |
| 4 | 0.488 s | 0.369 s |
| 5 | 0.414 s | 0.428 s |
| Average | **0.4512 s** | **0.4048 s** |

|  |  |  |
| --- | --- | --- |
| *Application connection time to* | | |
| Trials | **Post Zone**  **(8606 Bytes)** | **Get Zone**  **(8606 Bytes)** |
| 1 | 0.662 s | 0.722 s |
| 2 | 0.666 s | 0.716 |
| 3 | 0.656 s | 0.652 |
| 4 | 0.674 s | 0.712 |
| 5 | 0.712 s | 0.698 |
|  | 0.691 s | 0.654 |
| Average | **0. 8122 s** | **0.8303 s** |

|  |  |  |
| --- | --- | --- |
| *Website connection time to* | | |
| Trials | **Post spot**  **(49 Bytes)** | **Get Spot**  **(49 Bytes)** |
| 1 | 0.362 s | 0.349 s |
| 2 | 0.287 s | 0.320 s |
| 3 | 0.342s | 0.324 s |
| 4 | 0.317s | 0.328 s |
| 5 | 0.345 s | 0.328 s |
| Average | **0.3306 s** | **0.3298 s** |