|  |  |
| --- | --- |
| **Requirement** | **ID** |
| The pods will assist in their placement by outputting network connectivity information through a console interface.   1. The console interface will be provided through a USB access port. 2. Administrators will connect through a laptop and open a bash shell to log in. | **1** |
| Pod’s system will be password protected to limit access to administrators only. | **2** |
| The pods will have the ability to be environmentally aware to detect damage and alert administrators if they need repair or replacement.   1. Each pod will be equipped with an accelerometer to detect movement and seismic activity.    1. Data will be interrogated at a minimum of once every ten seconds 2. Each pod will be equipped with environment chips to detect the environment they’re placed in.    1. Contains a minimum of one luminosity and temperature sensor.    2. Luminosity sensor will be interrogated at a minimum of once every five minutes.    3. Temperature sensor will be interrogated at a minimum of once every five minutes. | **4** |
| Each pod will act as a repeater to extend coverage over an emergency area.   1. Each pod will be equipped with a Wi-Fi radio    1. The radio will meet at a minimum IEEE standard 802.11n    2. The radio configuration will have at a minimum a maximum data rate of 600 Mbps like what is provided by the IEEE standard 802.11n | **5** |
| The central server intranet connection will meet at a minimum IEEE standard 802.11n and will be password protected. | **6** |
| The pods will have the ability to display other pods that are within connectivity range to assist in placement.   1. The software will be executed through a bash shell. 2. The software will detect signal levels of pods within range and output them through a bash shell via USB. 3. The software will refresh the signal levels of the detected pods at a minimum of every 15 seconds. | **7** |
| The central server will have the ability to store sensor information.   1. The server will store sensor data into the database according to the sensor type.    1. Joined tables through a foreign key defined as the pod ID.    2. No null data will be stored for timestamps that don’t contain data for every sensor. 2. The server will only run MYSQL databases for uniformity. 3. Every data write will be done in one single query. | **8** |
| Every pod will have the ability to interrogate a sensor array connected to the GPIO of the raspberry pi.   1. Sensor interrogation will be executed by python scripts started when the pods are placed.    1. There will be a separate python program for each of the sensor array endpoints to interrogate.    2. Each timer that tracks the sampling rate of a sensor endpoint will run on a separate thread.    3. When a sensor array endpoint has been sampled, the value will be immediately sent with JSON encoding. | **9** |
| Every pod will have the ability to send sensor information to the central server.   1. The pods will use JSON to encode sensor data packets.    1. Each pod to server transmission will use on JSON hash with two keys; one for identification and one for sensor values.       1. The identification key will have on value containing an unique ID describing the pod.       2. The sensor values key will contain one value for each endpoint in the sensor array. 2. The pods will compute routing tables.    1. Each node in the cluster will run the wireless routing protocol (BABEL).    2. Each node in the cluster can reach each other despite being in different network subnets. | **10** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Expected Result | Outcome | Y | N | ID | Comments |
| User opens bash shell interface and configures it to connect to pod. Laptop is ready to connect. | Login worked and laptop connected. | x |  | (1)  (7) |  |
| User connects a USB to TTL Serial Cable from the laptop to the gpio pins on the pod. | Connection established. | x |  | (1)  (7) |  |
| User initiates communication with the pod by entering administrative credentials. | Login successful | x |  | (1)  (2) |  |
| User runs command **wavemon**  on the bash shell to view network information. | Wavemon opened in shell and shows current network information | x |  | (1)  (7) |  |
| User can see signal strength and network information from the main screen upon load of wavemon, user can view available networks by pressing f3. | All information is available | x |  | (1)  (7) |  |
| User can press the f7 key in wavemon to view the preferences for how often the signal strength is refreshed. It is listed as “Dynamic info updates”, user can set or change the setting here. | F7 works as expected | x |  | (7) |  |
| User exits wavemon to main terminal. | Main terminal now displayed | x |  | (1) |  |
| User will verify that pods have Wi-Fi dongle connected through USB port visually | Visually see connected wifi dongles | x |  | (5) |  |
| The Wi-Fi dongle meets IEEE standard 802.11n based on manufacturer's specifications | Documentation that came with Wi-Fi dongle shows standards have been met. | x |  | (5) |  |
| The radio configuration will have at a minimum a maximum data rate of 600 Mbps like what is provided by the IEEE standard 802.11n as stated in the manufacturer's specifications | Documentation that came with Wi-Fi dongle shows the speed of the radio and that it meets standards | x |  | (5) |  |
| User executes **sensorTest.py** program to view sensor data in the command line interface. | Script executed and all data is viewable | x |  | n/a |  |
| User looks at the second line of program output to see Lux sensor data. | Information is there | x |  | (4) |  |
| User looks directly to the right of the Lux data to see the amount of time until the sensor updates data. | Information is there | x |  | (4) |  |
| User looks at the third line of program output to see accelerometer data. | Information is there | x |  | (4) |  |
| User looks directly to the right of the accelerometer data to see the amount of time until the sensor updates data. | Information is there | x |  | (4) |  |
| User looks at the fourth line of program output to see temperature data in degrees Fahrenheit . | Information is there | x |  | (4) |  |
| User looks directly to the right of the accelerometer data to see the amount of time until the sensor updates. | Information is there | x |  | (4) |  |
| User quits sensorTest.py by pressing the ‘q’ key. | sensorTest.py succesfully quits and returns to terminal. | x |  | n/a |  |
| User runs client.py program. | Client.py succesfully opens. | x |  | n/a |  |
| User waits for data stream to begin. Once data stream begins user will be able to see the JSON encoding configuration. | Data prints out and JSON encoding is easy to see. | x |  | (9) (10) |  |
| User terminates the terminal session and removes connection from the pod. | Worked as expected | x |  | n/a |  |
| For the server: User will log-in by entering administrative credentials. | Login works as expected | x |  | (6) |  |
| The central server Wi-Fi network card meets the IEEE standard 802.11n qualifications based on manufacturer's specifications | Documentation that came with Wi-Fi network card shows standards have been met. | x |  | (6) |  |
| Open terminal on Ubuntu Server, Enter command: mysql -u leanuser -p, followed by the appropriate password when prompted. Prompt should now read “mysql>” | Terminal opened, commands worked as expected user entered password and mysql opened | x |  | (8) |  |
| Enter the command: USE leanSensors to select the correct database. Output should read Database changed. | Correct database selected and output is as expected | x |  | (8) |  |
| Enter the command: SELECT \* FROM pods. Output should show the records of all the pods recorded in the database. | Records of all pods shown | x |  | (8) |  |
| Enter Command: SELECT \* FROM temp. Output should show the records of the temperature readings | Records of all temp readings shown | x |  | (8) |  |
| Enter Command: SELECT \* FROM luminosity. Output should show the records of the luminosity readings | Records of all luminosity readings shown | x |  | (8) |  |
| Enter Command: SELECT \* FROM gps. Output should show the records of the gps readings | Records of all gps readings shown | x |  | (8) |  |
| Enter Command: SELECT \* FROM acceleration. Output should show the records of the acceleration readings | Records of all acceleration readings shown | x |  | (8) |  |
| In every node, enter the script:  sudo babeld -C 'redistribute metric 128' wlan0 &  Then, run the command:  ps aux | grep babeld | Babel daemon is show as running along with its process ID. | x |  | (10) |  |
| In every node run the command:  show ip route  Then, ping all listed reachable interfaces that are on a different subnet. If the node I’m curreltly logged on is in the subnet 192.168.1.0/24, ping all other interfaces in subnets different than the one I’m currently on. For example, if 192.168.10.1/24 is listed under show ip route; then, run:  ping 192.168.10.1 | Routes to all reachable subnets are shown in the routing table. Ping command returns an ICMP reply from interface. | x |  | (10) |  |