The home automation system that we are building has a few central components with several supplementary features. The central software components include the code for the WiFi module, the user-web interface, the database, the server and the communication between all components of the system. These parts are essential because they are interdependent. The failure of any one of these features causes the whole system to fail. Supplementary features include light control, music control, monitoring of door locks, and system control using a mobile application. These features are supplementary because they can fail without affecting other parts of the home automation system. It is important to keep in mind that even though we call these features supplementary, this does not mean these features are unimportant.

Following this distinction, the way we design the acceptance tests for our home automation system follows two phases of “Bottom Up” integration testing. In the first phase, we perform unit tests on the central components. Once the system passes the unit tests, individual components are put into small groups and tested together. We continue combining groups until we test each central component as a whole and finally perform integration testing as we test the components together. After the central components have passed the integration test phase, we begin the second phase. This phase begins with unit tests on the supplementary features. Following the same procedure that was used to test the central components, these supplementary features are tested from the bottom upwards. After each supplementary feature has passed its own “Bottom Up” testing, it is tested with the system of central components in integration testing. Although it was previously stated that the second phase of testing starts after the first phase, we can perform the unit tests for both phases concurrently.

The section below shows plans for unit tests of each class shown in the class diagram. We start by isolating individual functions in a class for testing.

**Unit Tests**

**DeviceController**

addDevice(deviceID); removeDevice(deviceID)

Success:

* The user sends a request to add or remove a device to his or her account, a device ID unique within the user’s account is passed to the function and the function sends the request to the server via communications. In the database, the device is added to or removed from the account.

Failure:

* The user sends a request to add or remove a device from his account and the request is not received by the server.
* The user sends a request to add or remove a device from his account and the device ID generated is not unique within the user’s account.
* The user sends a request to add or remove a device from his account, server receives the request, but the device ID is not stored in the database.

updateDeviceStatus(deviceID, deviceStatus)

Success:

* The device controller sends a message to the server via communications. This message asks the database to update the status of the specific user’s device.

Failure:

* Some or all of the message sent is lost during communications.
* The message is sent and received, but the wrong user account is updated.
* The message is sent and received, but the wrong device is updated.
* The message is not sent.

getDeviceStatus(deviceID)

Success:

* The device controller sends a message to the server via communications. This message requests from the database the status of a device unique to a user’s account.

Failure:

* Some or all of the message sent is lost during communications.
* The message is sent and received, but the wrong user account is accessed in the database.
* The message is sent and received, but the wrong device is accessed in the database.
* The message is not sent.

requestConnection(deviceID)

Success:

* The device controller receives a connection request from the Arduino via the device interface module. The controller then sends the request to server via communications and waits for a boolean reply. The function returns the boolean reply.

Failure:

* The controller receives a connection request, but does not send it.
* The controller sends a connection request when no request was received.
* The controller receives and sends a connection request to server, but never receives a reply and is stuck waiting.
* The reply is not boolean.

**DeviceInterfaceModule**

turnOn()

Success:

* T

Failure:

* T