System and Unit Test Report
Smart-Irrigation Backend
Team: The Wubalubadubdubs
Nov 28th 2015

Sprint 1

A.) User Story 2 from Sprint 1 : As a developer, I want the data from the raspberry pi to be stored on a MySql database so I can easily query the data.

Scenario :

1.) Start SqlPullTest.pl with python SqlPull.py in same diectory.
    a.) Enter in a range of readings to request.
2.) User should see range of values reported from MySQL table.

Sprint 2

A.) As a developer, I want to be able to generate dummy data for testing so that I can continue development regardless of the PI sensor's state.

Scenario :

1.) Open RandCSV.cfg in the same directory as RandCSV.py (/smart-irrigation/Backend/util)
2.) Check field names, date formatting, and sensor ranges in RandCSV.cfg
    a.) Specify date strings under a single [Dates] header as "varname = 'dateformat' all on a single line
    b.) Specify all random integer ranges under a single [Integers] header as "varname = min,max"
3.) run "python RandCSV.py <out_csv_file> <num_rows>" in the
4.) CSV file will be written to the specified file

B.) As a frontend developer, I want to be able to access data through an API so that I can provide functionality to the user.

Scenario :

1.) Enter the /smart-irrigation/Backend/QA directory.
2.) Open 'testlist' and ensure the tests on the API you'd like to run are present.
    a.) Tests can be viewed in the /Tests subdirectory of QA.

3.) Execute TestHarness.py in the QA directory.
4.) terminal should report the queries performed through the API and their successes and failures.

Sprint 3

A.) As an administrator, I want to implement security into our API so that only the frontend has access to the information stored in the database.

Scenario :

1.) Enter the /smart-irrigation/Backend/QA directory
2.) Open the ‘testlist’ file.
    a.) Ensure the testlist contains the tests httpauthentication_bad and httpauthentication_good.
3.) Execute TestHarness.py in the QA directory.
    a.) Observe the terminal output for httpauthentication tests. good will use a correct username and password. bad will not.
    b.) good should result in observed output “All Hail Bob!”
    c.) bad should result a failure to login.
4.) Examine TestingLog_<yourdate>_at_<yourtime> and ensure all tests completed successfully.

B.) As an administrator, I want to implement security into our API so that only the frontend has access to the information on the database.

Scenario :

1.) Enter the /smart-irrigation/Backend/QA directory
2.) Open the ‘testlist’ file.
    a.) Ensure the testlist contains the tests httpauthentication_bad and httpauthentication_good.
3.) Execute TestHarness.py in the QA directory.
    b.) Observe the terminal output for httpauthentication tests. good will use a correct username and password. bad will not.
    c.) good should result in observed output “All Hail Bob!”
    d.) bad should result a failure to login.
4.) Examine TestingLog_<yourdate>_at_<yourtime> and ensure all tests completed successfully.

Testing

pullavg - Tests the averaging function between dates.

Valid Class: {(validField+, validTable, validDateStart, validDateEnd)}

- The field must be in configAPI.ini
- Table must be in configAPI.ini
- Start date and end date must be ordered and formatted properly

Invalid Class: { (field, table, start, end) s.t criteria for valid class are not met }

Test for Valid Class: curl call containing valid fields. Returns an average value.
Test for invalid class: None

pullrange - Tests for pulling fields between a range of dates

Valid Class: {(validField+, validTable, validDateStart, validDateEnd)}

-Identical to requirements for pullavg

Invalid Class: {(field, table, start, end) s.t >= 1 criteria for valid class are not met }

Test for Valid Class: curl with valid fields. Returns list of JSON objects between between the specified range. Uses timestamp for date range

Test for invalid class: **badrange,** same command but with dates that are not properly formatted. Returns an 'Invalid Dates' message in a JSON object.

pullkeys - Gets values, specified by start and end, for a particular value (key)

Valid Class: {(validField+, validTable, validDateStart, validDateEnd, validKey)}

- The field must be in configAPI.ini
- Table must be in configAPI.ini
- Start date and end date must be ordered and formatted properly
- Key must be field in configAPI.ini

Invalid Class: {(field, table, start, end,key) s.t >= 1 invalid criteria}

Test for valid class: curl command with key specified, all valid fields. Returns range of JSON objects.

Test for invalid class: **badkey,** use invalid key "meow". Returns "Invalid Key" message in JSON object

pullforecast - Pulls the most recent forecast added to the weather table

Valid Class: {(validWeatherTable)}
Invalid Class { table : st table is not in configWeather.ini}

Test for valid class: curl command, returns most recent addition to table

Test for invalid class: **badforecast**, uses invalid table in call, corresponding message

postwater - Adds a watering entry to table and returns the range of values between the entry values.

Valid Class: {(validDuration, validStart, validEnd, gallons, validTable)}
Invalid class: Invalid fields

Test for valid class: Post an entry with curl command, then use mysql command line to pull the most recent value for the gallons per minute entered.
Test for invalid class: None

Testing was conducted by everyone, all using TestHarness.py file to run the tests.