Capital One Commercial Tech Innovation Take-Home Test

5 August 2016. v1.2

Thank you for your interest in work at Capital One Commercial Tech Innovation. A major part of our initiative is to set an example for how software should be constructed, in both general application architecture and in the details of each data structure and function. Following this, we ask that you provide us with a solution fulfilling the below specification as a code sample. This should demonstrate your best effort and serve as an example of how you think an application should be designed. You are free to spend as much time as you like writing your solution and refining it to be your best work; you will not in any way be evaluated on how long you require to do this. You are also free to consult any resource, including people and documents on and off the Internet, for advice and examples. However, we ask that all of the code that you deliver (excluding your npm dependencies) be solely your independent work.

Happy coding!

Delivery

Please provide your solution, any accompanying documentation, and this document (so we know which version of the test you built) in a single directory named `<firstname>-<lastname>.c1-code-challenge` as a zip file named `<firstname>-<lastname>.c1-code-challenge.zip`. e.g.

```
grace-hopper.c1-code-challenge.zip
└── grace-hopper.c1-code-challenge
    ├── package.json
    ├── take-home-test-v1.2.pdf
    └── <your files>
```

Your solution must have a `package.json` file and *may* have dependencies on other libraries such as `underscore`. Please do not include your `node_modules` folder in your solution. If you have npm dependencies, we recommend using shrinkwrap to ensure consistency between your dependency tree versions and the reconstituted dependency tree for the test.

Your code, once unzipped, will be run with the following commands:

```
npm install
npm run prepare
npm start -- --host <host> --port <port>
```

Please see the npm documentation for how to hook into these commands.

The environment your server will run on is:

- OS X 10.11.x
- NodeJS 6.x
- npm 3.x

Because of OS-specific considerations, we recommend against using any node-gyp dependencies.

Automated tests are not required for your submission, but will be considered if they are provided. We know that writing a comprehensive test suite requires a considerable amount of time and effort, so we will not expect full code or even feature coverage, but rather will treat what you provide as a demonstration of the style you employ to test your code.

Your tests, if provided, will be run with the following command:

```
npm test
```

You may write your solution in whatever language you like, so long as it compiles to and runs on NodeJS. For instance, it is acceptable to write your solution in CoffeeScript or TypeScript and have a transpiler convert your code into JavaScript that is suitable for Node to run. However, you must include only original source code in your solution file; you may use the `prepare` npm script hook to transpile your code.

Your solution must not rely on external resources, such as databases, web services, or file storage, to run. It is expected that all of the state of your application resides in memory and all of its behavior is fulfilled by your source code and its code-dependencies.

Evaluation

Your solution will be evaluated broadly on the following points.

- Does it fully implement the specification?
- How tolerant is the solution of faulty input?
- Does the solution have any obvious security flaws?
- How well organized is the source code?
- Is there sufficient documentation for a reasonably experienced developer to understand how the code works? This may include some or all of:
  - Naming of constructs
  - Code comments
  - External documents
  - Tests
- How thorough are the tests for the features they verify?†
- How closely does the source code for the server and the unit tests† follow best practices?

† If provided. Automated unit tests are not required.

Specifications

Sarah is interested in tracking weather conditions in her garden. To accomplish this, she has set up some weather instruments connected to a RaspberryPi that makes HTTP calls from time to time with various metrics.

Assumptions

Because the server and the RaspberryPi are on Sarah's home network, assume that the callers are authorized implicitly; you don't need to worry about users, passwords, or authentication of any kind.

Sarah's RaspberryPi will always send request bodies in a valid JSON format and include the `Content-Type` header where appropriate. Similarly, her downstream applications read only valid JSON and send the `Accept` header where appropriate.

When your server starts it must have a clean state with no measurements in it. All data it records must reside solely in your server's process memory and will be lost when it is terminated.

Metrics

As time goes on, Sarah will buy new instruments to plug into her RaspberryPi. Furthermore, some of the instruments she has already installed sometimes malfunction and stop reporting metrics! To handle this, Sarah has programmed her RaspberryPi to be very fault-tolerant and send a measurement whether or not a given instrument has reported a metric. Her code will *always* report the time accurately and at proper intervals, but the other metrics may not always be reported.

The instruments plugged into the RaspberryPi will always report their metrics as floating-point numbers. This includes instruments that have not been plugged in yet.

On day one, Sarah has installed instruments that report the following metrics. **Keep in mind that she may install new ones in the future.**

Metric Name	Type	Example	Notes
timestamp	DateTime	"2015-09-01T16:00:00.000Z"	Always sent as an ISO-8061 string in UTC
temperature	float	22.4	in °C
dewPoint	float	18.6	in °C
precipitation	float	142.2	in mm
...etc	float	1234.56	Interpretation depends on instrument

REST API

The following is an overview of the REST endpoints your solution must expose.

Method	Path	Request Body	Response Body
`POST`	`/measurements`	`Measurement`	(none)
`GET`	`/measurements/:timestamp`	(none)	`Measurement`
`GET`	`/measurements/:date`	(none)	`Measurement[]`
`PUT`	`/measurements/:timestamp`	`Measurement`	(none)
`PATCH`	`/measurements/:timestamp`	`Measurement` (partial)	(none)
`DELETE`	`/measurements/:timestamp`	(none)	(none)
`GET`	`/stats`[1]	(none)	`Statistic[]`

[1] The `/stats` endpoint accepts query parameters to for its response. These parameters are:

Parameter	Indicates	Notes
`stat`	which statistic to compute	can be repeated for more than one statistic
`metric`	which metric to compute the statistics for	can be repeated for more than one metric
`fromDateTime`	the inclusive minimum date and time of the range	in UTC, ISO-8061 format
`toDateTime`	the exclusive maximum date and time of the range	in UTC, ISO-8061 format

Acceptance Tests

Your objective is to create a system of REST endpoints that implement the following Acceptance Tests (ATs).

For each scenario, your server will be started, prepared according to the `Background` and `Given` steps, tested according to `When` and `Then` steps, and then shut down. Therefore, each scenario will test from a clean slate.

```
Feature: Add a measurement
  In order to have source information to examine later
  I want to be able to capture a measurement of several metrics at a specific time

  Scenario: Add a measurement with valid (numeric) values
    # POST /measurements
    When I submit a new measurement as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
    Then the response has a status code of 201
    And the Location header has the path "/measurements/2015-09-01T16:00:00.000Z"

  Scenario: Cannot add a measurement with invalid values
    # POST /measurements
    When I submit a new measurement as follows:
      | timestamp                  | temperature    | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | "not a number" | 16.7     | 0             |
    Then the response has a status code of 400

  Scenario: Cannot add a measurement without a timestamp
    # POST /measurements
    When I submit a new measurement as follows:
      | temperature | dewPoint | precipitation |
      | 27.1        | 20       | 0             |
    Then the response has a status code of 400

Feature: Get a measurement
  In order to learn what weather conditions were like at a specific time
  I want to be able to retrieve a measurement of several metrics at that time

  Background:
   # POST /measurements
   Given I have submitted new measurements as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
      | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |
      | "2015-09-01T16:20:00.000Z" | 27.5        | 17.1     | 0             |
      | "2015-09-01T16:30:00.000Z" | 27.4        | 17.3     | 0             |
      | "2015-09-01T16:40:00.000Z" | 27.2        | 17.2     | 0             |
      | "2015-09-02T16:00:00.000Z" | 28.1        | 18.3     | 0             |

  Scenario: Get a specific measurement
    # GET /measurements/2015-09-01T16:20:00.000Z
    When I get a measurement for "2015-09-01T16:20:00.000Z"
    Then the response has a status code of 200
    And the response body is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:20:00.000Z" | 27.5        | 17.1     | 0             |

  Scenario: Get a measurement that does not exist
    # GET /measurements/2015-09-01T16:50:00.000Z
    When I get a measurement for "2015-09-01T16:50:00.000Z"
    Then the response has a status code of 404

  Scenario: Get measurements from a day
    # GET /measurements/2015-09-01
    When I get measurements for "2015-09-01"
    Then the response has a status code of 200
    And the response body is an array of:
       | timestamp                  | temperature | dewPoint | precipitation |
       | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
       | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |
       | "2015-09-01T16:20:00.000Z" | 27.5        | 17.1     | 0             |
       | "2015-09-01T16:30:00.000Z" | 27.4        | 17.3     | 0             |
       | "2015-09-01T16:40:00.000Z" | 27.2        | 17.2     | 0             |

  Scenario: Get measurement from a day where no measurements were taken.
    # GET /measurements/:date
    When I get measurements for "2015-09-03"
    Then the response has a status code of 404

Feature: Update a measurement
  In order to correct the record of weather conditions
  I want to be able to update a measurement taken at a specific time

  Background:
    # POST /measurements
    Given I have submitted new measurements as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
      | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |

  Scenario: Replace a measurement with valid (numeric) values
    # PUT /measurements/2015-09-01T16:00:00.000Z
    When I replace the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 15.2          |
    Then the response has a status code of 204
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 15.2          |

  Scenario: Replace a measurement with invalid values
    # PUT /measurements/2015-09-01T16:00:00.000Z
    When I replace the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | temperature    | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | "not a number" | 16.7     | 15.2          |
    Then the response has a status code of 400
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp              | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |

  Scenario: Replace a measurement with mismatched timestamps
    # PUT /measurements/2015-09-01T16:00:00.000Z
    When I replace the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-02T16:00:00.000Z" | 27.1        | 16.7     | 15.2          |
    Then the response has a status code of 409
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp              | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |

  Scenario: Replace a measurement that does not exist
    # PUT /measurements/2015-09-02T16:00:00.000Z
    When I replace the measurement for "2015-09-02T16:00:00.000Z" as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-02T16:00:00.000Z" | 27.1        | 16.7     | 15.2          |
    Then the response has a status code of 404

  Scenario: Update metrics of a measurement with valid (numeric) values
    # PATCH /measurements/2015-09-01T16:00:00.000Z
    When I update the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | precipitation |
      | "2015-09-01T16:00:00.000Z" | 12.3          |
    Then the response has a status code of 204
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 12.3          |

  Scenario: Update metrics of a measurement with invalid values
    # PATCH /measurements/2015-09-01T16:00:00.000Z
    When I update the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | precipitation  |
      | "2015-09-01T16:00:00.000Z" | "not a number" |
    Then the response has a status code of 400
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |

  Scenario: Update metrics of a measurement with mismatched timestamps
    # PATCH /measurements/2015-09-01T16:00:00.000Z
    When I update the measurement for "2015-09-01T16:00:00.000Z" as follows:
      | timestamp                  | precipitation |
      | "2015-09-02T16:00:00.000Z" | 12.3          |
    Then the response has a status code of 409
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |

  Scenario: Update metrics of a measurement that does not exist
    # PATCH /measurements/2015-09-02T16:00:00.000Z
    When I update the measurement for "2015-09-02T16:00:00.000Z" as follows:
      | timestamp                  | precipitation |
      | "2015-09-02T16:00:00.000Z" | 12.3          |
    Then the response has a status code of 404

Feature: Delete a measurement
  In order to remove incorrect measurements
  I want to be able to delete a measurement taken at a specific time

  Background:
    Given I have submitted new measurements as follows:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
      | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |

  Scenario: Delete a specific measurement
    # DELETE /measurements/2015-09-01T16:00:00.000Z
    When I delete the measurement for "2015-09-01T16:00:00.000Z"
    Then the response has a status code of 204
    And the measurement for "2015-09-01T16:00:00.000Z" does not exist
    But the measurement for "2015-09-01T16:10:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |

  Scenario: Delete a measurement that does not exist
    # DELETE /measurements/2015-09-01T16:20:00.000Z
    When I delete the measurement for "2015-09-01T16:20:00.000Z"
    Then the response has a status code of 404
    And the measurement for "2015-09-01T16:00:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.7     | 0             |
    And the measurement for "2015-09-01T16:10:00.000Z" is:
      | timestamp                  | temperature | dewPoint | precipitation |
      | "2015-09-01T16:10:00.000Z" | 27.3        | 16.9     | 0             |

Feature: Get measurement statistics
  In order to understand trends of measurements
  I want to be able to get statistics over specified periods of time

  Background:
    Given I have submitted new measurements as follows:
      | timestamp                  | temperature | dewPoint |
      | "2015-09-01T16:00:00.000Z" | 27.1        | 16.9     |
      | "2015-09-01T16:10:00.000Z" | 27.3        |          |
      | "2015-09-01T16:20:00.000Z" | 27.5        | 17.1     |
      | "2015-09-01T16:30:00.000Z" | 27.4        | 17.3     |
      | "2015-09-01T16:40:00.000Z" | 27.2        |          |
      | "2015-09-01T17:00:00.000Z" | 28.1        | 18.3     |

  Scenario: Get stats for a well-reported metric
    # GET /stats?<params...>
    When I get stats with parameters:
      | param        | value                    |
      | stat         | min                      |
      | stat         | max                      |
      | stat         | average                  |
      | metric       | temperature              |
      | fromDateTime | 2015-09-01T16:00:00.000Z |
      | toDateTime   | 2015-09-01T17:00:00.000Z |
    Then the response has a status code of 200
    And the response body is an array of:
      | metric        | stat      | value |
      | "temperature" | "min"     | 27.1  |
      | "temperature" | "max"     | 27.5  |
      | "temperature" | "average" | 27.3  |

  Scenario: Get stats for a sparsely reported metric
     # GET /stats?<params...>
    When I get stats with parameters:
      | param        | value                    |
      | stat         | min                      |
      | stat         | max                      |
      | stat         | average                  |
      | metric       | dewPoint                 |
      | fromDateTime | 2015-09-01T16:00:00.000Z |
      | toDateTime   | 2015-09-01T17:00:00.000Z |
    Then the response has a status code of 200
    And the response body is an array of:
      | metric     | stat      | value |
      | "dewPoint" | "min"     | 16.9  |
      | "dewPoint" | "max"     | 17.3  |
      | "dewPoint" | "average" | 17.1  |

  Scenario: Get stats for a metric that has never been reported
     # GET /stats?<params...>
    When I get stats with parameters:
      | param        | value                    |
      | stat         | min                      |
      | stat         | max                      |
      | stat         | average                  |
      | metric       | precipitation            |
      | fromDateTime | 2015-09-01T16:00:00.000Z |
      | toDateTime   | 2015-09-01T17:00:00.000Z |
    Then the response has a status code of 200
    And the response body is an empty array

  Scenario: Get stats for more than one metric
    # GET /stats?<params...>
    When I get stats with parameters:
      | param        | value                    |
      | stat         | min                      |
      | stat         | max                      |
      | stat         | average                  |
      | metric       | temperature              |
      | metric       | dewPoint                 |
      | metric       | precipitation            |
      | fromDateTime | 2015-09-01T16:00:00.000Z |
      | toDateTime   | 2015-09-01T17:00:00.000Z |
    Then the response has a status code of 200
    And the response body is an array of:
      | metric        | stat      | value |
      | "temperature" | "min"     | 27.1  |
      | "temperature" | "max"     | 27.5  |
      | "temperature" | "average" | 27.3  |
      | "dewPoint"    | "min"     | 16.9  |
      | "dewPoint"    | "max"     | 17.3  |
      | "dewPoint"    | "average" | 17.1  |
```