FORMAT

URL endpoint (domain name implied)(?) - **REQUEST TYPE**

:param: these are identifiers that are inserted into the URL

:request body: these are key-value pairs included in the request body

:header: these are key-value pairs in the request headers

:query: if '?' is included at the end of a URL, these optional query values can be added after the '?' in the URL. E.g. /api/nodes/prev_24h/72?timestamp=2017-07-29 21:23:08.986+00

Explanation of API function and return value

Brief overview of authentication layer

A user's username and password should only be sent to the server once per login session. That is, when the user logs in, their username and password are sent to the server for validation. Those credentials *should not* be sent to the server again until the current user logs out and a new user logs in (a "new user" could be the same user).

When the username and password are verified on the server, the server responds with an authentication token that is unique to the user that just logged in. All succeeding API requests made by that user, until they log out, must include this authentication token (called the api_token) in the request body. The api_token behaves as authentication *and* identification for the user at this point. In the documentation, when you see api_token as a request body requirement, it is referring to this authentication token.

Nodes & Readings

/api/nodes - **POST**

:request body: id (optional int), ipaddress (optional string)

:header: Authorization=api_token (string)

This endpoint creates a new node under the user for which the api_token is provided.

The *id* key may be provided in the body. If it is, the new node's id, i.e. its primary key in the database, will be set to the value for the key *id*. If the key *id* is not provided in the request body, the default value will be assigned to the new node's id (the server tracks the default value).

The *ipaddress* key takes a simple string value, and while it is not required to follow IP address format, and this value is not currently used, the value for the key *ipaddress* should look like an IP address (e.g. 1.1.1.1). If the key *ipaddress* is not provided in the request body, the default value will be assigned to the new node's id (the server tracks the default value).

This function creates a new node with the provided (or default) id and ipaddress, assigns it to the requesting user, and then the new node is sent as a JSON response back to the requesting client. Finally, a dummy reading with all 0 values is created for the new node.

This function also increments the nodeCount of the user with the provided api_token.

/api/nodes/:nid - **GET**
:param: nid (node id)
This endpoint gets the node with the id *nid*, as well as all readings that have ever been posted for this node. This information is returned as a JSON object and sent in the response from the server back to the requesting client. Note that the included readings are ordered by descending id, i.e. the latest reading is at the beginning of the list of readings, and the earliest reading is at the end of the list of readings.

/api/nodes/status? - **GET**
:query: nodes (comma-separated list of integer node IDs)
This endpoint is used to get the status of all nodes provided in the *nodes* query parameter list. It retrieves the latest reading posted for each node in the list. If that reading was posted earlier than now minus that node's interval value in minutes (stored in the DB), the node's status is "inactive". If the reading was posted the node's interval and any metric is out of range, but no metric is more than 5 points out of range, the status of that node is "warning". Finally, if the reading was posted within the node's interval and any reading is more than 5 points out of range the node's status is "bad". The return object is a map (aka js object) where each key is a node ID and the corresponding value is an object containing the latest posted reading and the status of the node with that ID.

/api/nodes/update? - **PUT**
:param: nid (node id)
:request body: ipaddress (optional string), name (optional string), tempMin (optional int), tempMax (optional int), humidityMin (optional int), humidityMax (optional int), moistureMin (optional int), moistureMax (optional int), sunlightMin (optional int), sunlightMax (optional int), groupName (optional string), interval (optional int), sensors (optional string)
:header: Authorization=api_token (string)
:query: nodes (comma-separated list of integer node IDs)
This endpoint updates the fields provided in the request body for all nodes with IDs provided in request.query.nodes. Whatever fields are not provided in the request body will not be altered. This endpoint was created to allow updating multiple nodes with the same values at one time. Note that this endpoint can still be used for a updating a single node by simply providing just one node's ID in request.query.nodes. A JSON list of updated nodes is returned in the response. Special Note: if the *name* key is not included, it is not altered. If the *name* key is included, but has an empty string as its value, the name of node with id *nid* will be set to NULL. Otherwise, a non-empty string value for the key *name* will set the node's name appropriately. The same goes for *groupName*.

/api/nodes/:nid - **DELETE**

:param: nid (node id)
:header: Authorization=api_token (string)
Delete the node with id *nid.* Note the deleting a node cascades to all of that node's readings (all of the deleted node's readings are also deleted). This function will also decrements the nodeCount of the user with the provided api_token. The response is 204 - no content.

/api/nodes/:nid/new_reading - **POST**
:param: nid (node id)
:request body: humidity (int), sunlight (int), temperature (int), moisture (int), battery (optional int), postTime (optional timestamp)
This endpoint created a new reading for the node with id *nid*. The request body should include integer values for each of the keys *humidity, sunlight, temperature,* and *moisture.* The *battery* key is optional, and it takes an integer value but is set to NULL by default.
The *postTime* key is also optional, and when included with a timestamp value (e.g. `1970-01-01 00:00:01`), the createdAt value for the new reading will be set to the value for the *postTime* key. Note that the value for this key must be the UTC timezone timestamp. This feature was added so we could more accurately track the time at which a node generated/posted a new reading. Note that the updatedAt field for the new reading will maintain its default behavior.
The newly created reading is sent in the response as a JSON object to the requesting client.

Users

/api/users/login - **POST**
:request body: email (string), password (string)
This endpoint finds the user in the database with the email and password combination provided in the request body. If no such user is found, a response dictating that invalid credentials were provided is sent back to the requesting client. If a user is found, the password field of the response is set to null (to avoid leaking the password), and then the user and their nodes are converted into a JSON object and sent in the response to the requesting client. The nodes are ordered by ascending node id. The api_token will be available in this response.

/api/users - **POST**
:request body: username (string, optional), password (string), email (string), firstname (string), lastname (string)
This endpoint creates a new user and initializes his/her username, password, first and last name, and email to the values provided in the request body. The new user's nodeCount is also

initialized to 0. The user's password is nullified for the response object (not in the database!) and the user is then sent as a JSON object response to the requesting client.

/api/users/email - **POST**
:request body: username (string, optional), password (string), email (string), firstname (string), lastname (string)
This endpoint creates a new user and initializes his/her username, password, first and last name, and email to the values provided in the request body. The new user's nodeCount is also initialized to 0. A verification email is sent to *email*. The user is not officially created in the database until the verification link in the email that was sent is clicked. A JSON object is returned in the response that contains information about the email, i.e. from, to, rejected sends, successful sends, etc. This endpoint should be used to create a new user in production.

/api/users/verify? - **GET**
:query: id (string)
When the /api/users/email endpoint is used for creating a user, a URL to this endpoint is added in the email that is sent to the to-be-confirmed user's email address. The query contains a unique 15-digit identification string (*id*) that is used in the function that this endpoint calls to identify the user that is confirming their email, gather that user's information, and formally create that user in the DB. This endpoint should NOT be used as the others are; it is strictly used as a generated endpoint in verification emails. This API function also sets the api_token cookie in the front end and redirects to the user dashboard.

/api/users/recover_pwd - **POST**
:request body: email (string)
This endpoint looks up a user with email = *email*. If such a user is found, an email is sent to that user's email address with a URL that, when clicked, takes the user to a page where he/she can reset his/her password. This endpoint simply sends the email and does not have anything to do with resetting the password after that.

/api/reset_pwd? - **POST**
:query: id (string)
This endpoint behaves similarly to /api/users/verify. It is also similar to /api/users/verify in the sense that it should not be used as the other endpoints are; this endpoint is generated in an email by /api/users/recover_pwd. When the user clicks on the link in the email they receive (which points to this endpoint), this function identifies the requesting user by the value for *id*, effectively logs in that user by setting the api_token cookie in the frontend, and redirects the user to /reset_password - a page with a form for resetting the user's password (that page calls /api/users/update).

/api/users/update - **PUT**
:request body: username (optional string), password (optional string), firstname (optional string), lastname (optional string)
:header: Authorization=api_token (string)
This endpoint updates the provided fields for the user to which the provided api_token belongs. Any fields that are not provided are not altered. The updated user object is sent as a JSON object in the response to the requesting client, but the password is made null for security purposes.

/api/users/token - **PUT**
:header: Authorization=api_token (string)
This endpoint generates a new api_token for the user to which the provided api_token belongs. The response contains the updated user with a nullified password field as a JSON object.

/api/users/getuser - **GET**
:header: Authorization=api_token (string)
This endpoint takes the api_token provided as the key to the Authorization header and converts the user that corresponds to said api_token into a JSON object and sends it as a response to the requesting client. Note that the user's password is not included in the response.

/api/users/delete - **DELETE**
:header: Authorization=api_token (string)
This endpoint deletes the user that corresponds to the provided api_token. The effect cascades into nodes, which then cascades into readings. The response is 204 - no content.

Notifications

/api/notifications? - **POST**
:request body: sensor (string), overMax | underMin (bool), reading (optional int)
:query: nodeId (int)
:header: Authorization=api_token (string)
This endpoint creates a new notification under a user and node. It shows that the latest reading for the *sensor* on node with id *nodeId* was either over the ideal range maximum or under the ideal range minimum. An optional reading value may be attached to the notification, so the notification may be interpreted as "Node with id *nodeId* posted a reading of *reading* for the sensor *sensor* that was over the ideal max / under the ideal min."

The value for sensor must be one of “temperature”, “humidity”, “moisture”, and “sunlight”.

The request body must contain either overMax or underMin, but both may be provided, so long as they are not equal. They will both default to false, however they cannot be equal, so excluding both will cause a post failure.
The newly created notification will be returned as a JSON object in the response.

/api/notifications/all - **GET**
:header: Authorization=api_token (string)
:query: timestamp (string), numDays (int)
This endpoint returns a list of all notifications, dismissed or undismissed, for a user. If included, the *timestamp* query serves as the end time for the query, and the *numDays* query tells the query to get notifications within numDays days of the end time. For example, if you wanted to get notifications that are posted before July 29, 2017 (at a given time), you could set the *timestamp* query element to ‘2017-07-29 21:23:08.986+00’. Note the timestamp format - if not in this format, the query will fail. If you wanted the three days of notifications up to and including July 29, 2017 (at a given time), you could set *timestamp* to ‘July 29, 2017 (at a given time)’ and *numDays* to ‘3’. Note that each query element can be used individually, but they are most useful when used together.

/api/notifications/undismissed - **GET**
:header: Authorization=api_token (string)
This endpoint returns a list of all undismissed notifications for a user. Dismissed notifications will not be included. If included, the *timestamp* query serves as the end time for the query, and the *numDays* query tells the query to get notifications within numDays days of the end time. For example, if you wanted to get notifications that are posted before July 29, 2017 (at a given time), you could set the *timestamp* query element to ‘2017-07-29 21:23:08.986+00’. Note the timestamp format - if not in this format, the query will fail. If you wanted the three days of notifications up to and including July 29, 2017 (at a given time), you could set *timestamp* to ‘July 29, 2017 (at a given time)’ and *numDays* to ‘3’. Note that each query element can be used individually, but they are most useful when used together.

/api/notifications/:nid - **PUT**
:request body: dismissed (bool)
:param: nid (integer id for the notification to update)
:header: Authorization=api_token (string)
This endpoint updates the dismissed field on the notification with id *nid* to *dismissed*. If *dismissed* is not included in the request body nothing is changed. The updated (or not updated) notification object is returned in the response.

/api/notifications? - **PUT**
:request body: dismissed (bool)
:header: Authorization=api_token (string)
:query: notifications (list of comma-separated integer notification IDs)
This endpoint updates the dismissed field on all notifications with id values in the list *notifications* to *dismissed*. If *dismissed* is not included in the request body then nothing is changed. The updated (or not updated) notification objects are returned as a list in the response.

/api/notifications/:nid/delete - **DELETE**
:param: nid (integer id for the notification to delete)
:header: Authorization=api_token (string)
This endpoint deletes the notification with id *nid* in the database, removing all record of its existence.

/api/notifications/delete_many? - **DELETE**
:query: notifications (comma-separated list of integer notification IDs)
:header: Authorization=api_token (string)
This endpoint deletes all notifications with specified IDs (in the *notifications* query object) in the database, removing all record of their existence.

Other

/api/nodes/:nid/latest_reading - **GET**
:param: nid (node id)
:header: Authorization=api_token (string)
This endpoint gets the single latest reading for the node with id *nid*. The reading object is sent back to the requesting client as a JSON object. The requesting user's api_token must be given as the key to the authorization header in the request.

/api/nodes/latest_readings/all - **GET**
:header: Authorization=api_token (string)
This endpoint gets the single latest reading for each of the nodes that belong to the user that corresponds to the provided api_token. The readings are ordered by increasing node id. This list of readings is sent back to the requesting client as a JSON object. The requesting user's api_token must be given as the key to the authorization header in the request.

/api/nodes/prev_24h/:nid? - **GET**
:param: nid (node id)
:header: Authorization=api_token (string)
:query: timestamp (optional string)
This endpoint gathers the last 24 hrs of readings for the node with id *nid*. The readings are put into a list with the latest readings at the end of the list and earliest readings at the beginning of the list. The list is sent as a JSON response to the requesting client.
By default the function looks at the last 24 hours from the time of the function call. The requesting user's api_token must be given as the key to the authorization header in the request.

A user can also provide a *timestamp* key-value pair in the request query to look at the 24 hours previous to a specific time. For example, if I want to get readings within the last 24 hrs of 2017-06-02 01:26:19.768+00, I can provide this value for the *timestamp* key in the request query. If the *timestamp* key is not included in the request query, the default action (to start at the time of the request) is taken.
Special Note: the value for the *timestamp* key **must** take the following format -

YYYY-mm-dd hh:MM:ss.SSS
Where m = month, M = minutes, s = seconds, and S = milliseconds
E.g. 2017-06-02 01:26:19.768
Timezone offset may also be included, but it will be ignored
E.g. 2017-06-02 01:26:19.768+00 (this is how timestamps are stored in our DB)

/api/nodes/prev_xh/:nid? - **GET**
:param: nid (node id)
:header: Authorization=api_token (string)
:query: timestamp (optional string), hours (required, non-negative int)
This endpoint gathers and returns all readings within the last *hours* hrs for the node with id *nid*. The readings are put into a list with the latest readings at the end of the list and earliest readings at the beginning of the list. The list is sent as a JSON response to the requesting client.
The requesting user's api_token must be given as the key to the authorization header in the request.

A user can also provide a *timestamp* key-value pair in the request query to look at the *hours* hours previous to a specific time. For example, if I want to get readings within the last *hours* hrs of 2017-06-02 01:26:19.768+00, I can provide this value for the *timestamp* key in the request query. If the *timestamp* key is not included in the request query, the default action (to start at the time of the request) is taken.
Special Note: the value for the *timestamp* key **must** take the following format -

YYYY-mm-dd hh:MM:ss.SSS
Where m = month, M = minutes, s = seconds, and S = milliseconds
E.g. 2017-06-02 01:26:19.768
Timezone offset may also be included, but it will be ignored
E.g. 2017-06-02 01:26:19.768+00 (this is how timestamps are stored in our DB)

Deprecated

/api/nodes - **POST (use Authorization header instead of request body for api_token)**
:request body: api_token (string), id (optional int), ipaddress (string)
This endpoint creates a new node under the user for which the api_token is provided.
The *id* key may be provided in the body. If it is, the new node's id, i.e. its primary key in the database, will be set to the value for the key *id*. If the key *id* is not provided in the request body, the default value will be assigned to the new node's id (the server tracks the default value).
The *ipaddress* key takes a simple string value, and while it is not required to follow IP address format, and this value is not currently used, the value for the key *ipaddress* should look like an IP address (e.g. 1.1.1.1).
This function creates a new node with the provided (or default) id and ipaddress, assigns it to the requesting user, and then the new node is sent as a JSON response back to the requesting client.
Finally, a dummy reading with all 0 values is created for the new node.
This function also increments the nodeCount of the user with the provided api_token.

/api/nodes/:nid - **PUT (use /api/nodes/update?)**
:param: nid (node id)
:request body: api_token (string), ipaddress (optional string), name (optional string), tempMin (optional int), tempMax (optional int), humidityMin (optional int), humidityMax (optional int), moistureMin (optional int), moistureMax (optional int), sunlightMin (optional int), sunlightMax (optional int), groupName (optional string)
This endpoint updates the fields provided in the request body for the node with id *nid*. Whatever fields are not provided in the request body will not be altered. Special Note: if the *name* key is not included, it is not altered. If the *name* key is included, but has an empty string as its value, the name of node with id *nid* will be set to NULL. Otherwise, a non-empty string value for the key *name* will set the node's name appropriately. The same goes for *groupName*.

/api/nodes/:nid - **PUT (use /api/nodes/update?)**
:param: nid (node id)

:request body: ipaddress (optional string), name (optional string), tempMin (optional int), tempMax (optional int), humidityMin (optional int), humidityMax (optional int), moistureMin (optional int), moistureMax (optional int), sunlightMin (optional int), sunlightMax (optional int), groupName (optional string)
:header: Authorization=api_token (string)
This endpoint updates the fields provided in the request body for the node with id *nid*. Whatever fields are not provided in the request body will not be altered. Special Note: if the *name* key is not included, it is not altered. If the *name* key is included, but has an empty string as its value, the name of node with id *nid* will be set to NULL. Otherwise, a non-empty string value for the key *name* will set the node's name appropriately. The same goes for *groupName*.

/api/nodes/:nid - **DELETE (use Authorization header instead of request body for api_token)**
:param: nid (node id)
:request body: api_token (string)
Delete the node with id *nid.* Note the deleting a node cascades to all of that node's readings (all of the deleted node's readings are also deleted). This function will also decrements the nodeCount of the user with the provided api_token. The response is 204 - no content.

/api/users/update - **PUT (use Authorization header instead of request body for api_token)**
:request body: api_token (string), username (optional string), password (optional string)
This endpoint updates the provided fields for the user to which the provided api_token belongs. Any fields that are not provided are not altered. The updated user object is sent as a JSON object in the response to the requesting client, but the password is made null for security purposes.

/api/users/token - **PUT (use Authorization header instead of request body for api_token)**
:request body: api_token (string)
This endpoint generates a new api_token for the user to which the provided api_token belongs. The response contains the updated user with a nullified password field as a JSON object.

/api/users/delete - **DELETE (use Authorization header instead of request body for api_token)**
:request body: api_token (string)
This endpoint deletes the user that corresponds to the provided api_token. The effect cascades into nodes, which then cascades into readings. The response is 204 - no content.

/api/users/getuser - **POST (replaced by GET with Authorization header)**
:request body: api_token (string)

This endpoint converts the user that corresponds to the api_token provided in the request body into a JSON object and sends it as a response to the requesting client. Note that the user's password is not included in the response.

/api/nodes/:nid/latest_reading - **POST (replaced by GET with Authorization header)**
:param: nid (node id)
:request body: api_token (string)
This endpoint gets the single latest reading for the node with id *nid*. The reading object is sent back to the requesting client as a JSON object.

/api/nodes/latest_readings/all - **POST (replaced by GET with Authorization header)**
:request body: api_token (string)
This endpoint gets the single latest reading for each of the nodes that belong to the user that corresponds to the provided api_token. The readings are ordered by increasing node id. This list of readings is sent back to the requesting client as a JSON object.

/api/nodes/prev_24h/:nid? - **POST (replaced by GET with Authorization header)**
:param: nid (node id)
:request body: api_token (string), timestamp (optional string)
This endpoint gathers the last 24 hrs of readings for the node with id *nid*. The readings are put into a list with the latest readings at the end of the list and earliest readings at the beginning of the list. The list is sent as a JSON response to the requesting client.
By default the function looks at the last 24 hours from the time of the function call. A user can also provide a *timestamp* key-value pair in the request body to look at the 24 hours previous to a specific time. For example, if I want to get readings within the last 24 hrs of 2017-06-02 01:26:19.768+00, I can provide this value for the *timestamp* key in the request body. If the *timestamp* key is not included in the request body, the default action (to start at the time of the request) is taken.
Special Note: the value for the *timestamp* key **must** take the following format -

YYYY-mm-dd hh:MM:ss.SSS
Where m = month, M = minutes, s = seconds, and S = milliseconds
E.g. 2017-06-02 01:26:19.768
Timezone offset may also be included, but it will be ignored
E.g. 2017-06-02 01:26:19.768+00 (this is how timestamps are stored in our DB)

API functions to develop

- We need an API function that returns all readings posted for a node in the last week. These readings will likely need to be aggregated. I'm thinking a min, max, and median value for each of the metrics for each day. For example, return object =

```
{
  7 days ago: { min: x, max: y, median: z},
  6 days ago: { min: x, max: y, median: z},
  5 days ago: { min: x, max: y, median: z},
  4 days ago: { min: x, max: y, median: z},
  3 days ago: { min: x, max: y, median: z},
  2 days ago: { min: x, max: y, median: z},
  1 day ago: { min: x, max: y, median: z}
}
```