Sensorlab.io Javascript SDK Documentation

Sensorlab.io

Oct 15, 2018

Contents:

CHAPTER 1

How to use SDK

1.1 How to use SDK for REST API

1.1.1 Getting Started

This is documentation on Javascript SDK for http://sensorlab.io/ REST API.

This SDK can be easily used in NodeJS and as JS library to use in browser.

Installation

Install with npm:

```
$ npm i --save https://github.com/sensorlabio/sensorlabio-javascript-sdk
```

Use with NodeJS

Initialize inside your js code

```
import {SensorlabApi} from "sensorlabio-javascript-sdk";
let api = new SensorlabApi();
```

You can specify another url to REST API like this:

```
let api = new SensorlabApi('http://testing.sensorlab.io/api');
```

Note: There's no need to append */v1* to the url, endpoint methods will handle it.

You can also provide saved JWT using *setToken* method:

```
let api = new SensorlabApi('http://testing.sensorlab.io/api');
api.setToken(token);
```

Use with browsers

Download latest master release:

https://github.com/sensorlabio/sensorlabio-javascript-sdk/archive/master.zip

Unzip:

- *builds/index.min.js*

Use them inside your html code:

```
<script src="build/index.min.js"></script>
```

And inside your JS code:

```
<script type="text/javascript">
    var sdk = SensorlabSDK;
    var api = new sdk.SensorlabApi();
</script>
```

1.1.2 Response and error handling

Every endpoint method is async.

Methods await for request to API to completed and then returns Promise with result or throws exception on error.

Here's how you handle success results:

```
api.sensors.list(options)
         .then((sensors_response) => {
            console.log(sensors_response);
         });
```

Only HTTP status 200 is considered as success.

SDK handles few error types and throws these exceptions:

- *401 - ApiErrorUnauthorizedException*. SDK throws this exception if there's problem with authentication with token or authentication credentials are wrong for *token()* methods.
- *404 - ApiErrorNotFoundException*. SDK throws this exception if there's no object to return. This exception can appear if API's endpoint path is incorrect.
- *422 - ApiErrorValidationException*. SDK will throw this exception on validation errors. Some fields can be required or require special format. All those error will be available in the *errors* parameter of exception as array.
- *500 - ApiErrorInternalException*. If there are any problems with API this exception will be thrown.

Example:

```
[
  {
    "code": 1,
    "message": "Please, provide name field. This cannot be empty.",
```

(continues on next page)

(continued from previous page)

```
    "param": "name"
  }
]
```

In this example you see 3 fields:

- *code* - error code for error. You will see information on those codes further in endpoint methods how-to.
- *message* - error message. You can use those to display it to user.
- *param* - which field/param has this validation error.

If there's any other HTTP status SDK will throw *ApiErrorBasicException*.

All error exceptions extend *ApiErrorBasicException*.

There are 3 main parameters available for exception classes:

- *status* - HTTP status
- *message* - Error message
- *errors* - this field is mostly for 422 error exceptions.

You can handle errors like this:

```
api.sensors.list(options)
         .catch((exception) => {
            console.log(exception.status);
            console.log(exception.message);
            console.log(exception.errors);
         });
```

1.1.3 Authentication actions

User Authentication

You can authenticate user and get an authorization token:

```
let api = new SensorlabApi();
api.auth.user_token(email, password)
         .then((user) => {
            console.log(user);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success *User* model object will be returned in promise with generated jwt *token*:

```
let api = new SensorlabApi();
api.auth.user_token(email, password)
         .then((user) => {
            console.log(user.token);
         });
```

Output:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJfaWQiOiI1YWJhMTY1ZTQ4MmJlYzJkZjg4N2M2YTMiLCJpYXQiOjE1MjIxNDY0MTYsImV4cCI6MTUyMjIzMjgxNn0.
↪-6kJm1Rbd_SPbuwc6kg6FHuJnUii8FtKI9DXR0J5-Ig
```

Token will be automatically saved in SensorlabApi object and used for future requests. You can get token from User model or with method *SensorlabApi.getToken()* and save it for future requests.

Note: Not every endpoint is available for user token. You will find more information in the endpoint description itself.

Application Authentication

You can authenticate application and get an authorization token:

```
let api = new SensorlabApi();
api.auth.application_token(public_api_key, private_api_key)
         .then((application_token) => {
            console.log(application_token);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success *ApplicationToken* model object will be returned in promise with generated jwt *token*:

```
let api = new SensorlabApi();
api.auth.application_token(public_api_key, private_api_key)
         .then((application_token) => {
            console.log(application_token.token);
         });
```

Output:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJfaWQiOiI1YWJhMTY1ZTQ4MmJlYzJkZjg4N2M2YTMiLCJpYXQiOjE1MjIxNDY0MTYsImV4cCI6MTUyMjIzMjgxNn0.
↪-6kJm1Rbd_SPbuwc6kg6FHuJnUii8FtKI9DXR0J5-Ig
```

Token will be automatically saved in SensorlabApi object and used for future requests. You can get token from User model or with method *SensorlabApi.getToken()* and save it for future requests.

Note: Not every endpoint is available for application token. You will find more information in the endpoint description itself.

1.1.4 Profile actions

Get user profile

You can get user's profile information

```
let api = new SensorlabApi();
api.profile.get()
         .then((profile) => {
            console.log(profile);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success *Profile* model object will be returned in promise with available profile information:

```
let api = new SensorlabApi();
api.profile.get()
         .then((profile) => {
            console.log(profile.email);
         });
```

Output:

```
test@test.com
```

You can fetch profile with *User* model object that you got from authentication:

```
let api = new SensorlabApi();
api.profile.get()
         .then((user) => {
            user.profile().then((profile) => {
                console.log(profile.email);
            });
         });
```

Note: Available for:

- User token

Change user password

You can change password for authenticated users.

```
let api = new SensorlabApi();
api.profile.change_password(old_password, new_password, new_password_check)
         .then((response) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success *ApiResponse* object will be returned in promise with available profile information:

```
let api = new SensorlabApi();
api.profile.change_password(old_password, new_password, new_password_check)
         .then((response) => {
            console.log(response.code);
```

(continues on next page)

(continued from previous page)

```
            console.log(response.status);
            console.log(response.success);
            console.log(response.message);
        });
```

Output:

```
100
200
true
New password is set for user.
```

Codes and messages for validation errors:

- *code=1 - Please, provide old password. This cannot be empty.*
- *code=2 - You must provide new password.*
- *code=3 - You must provide new password check.*
- *code=4 - Password is incorrect. Please provide you current password.*
- *code=5 - Both "new password" and "new password check" values must be equal.*

Note: Available for:

- User token

1.1.5 Applications Actions

Get applications

You can get list of applications:

```
let api = new SensorlabApi();
api.applications.list(options)
    .then((applications_response) => {
        console.log(applications_response);
    });
});
```

On success you will get promise with *ApplicationsResponse* object.

```
api.applications.list(options)
    .then((applications_response) => {
        console.log(applications_response.page);
        console.log(applications_response.count);
        console.log(applications_response.applications);
    });
});
```

Parameters of *ApplicationsResponse*:

- *page* - amount of pages available.
- *count* - amount of applications total.

- *applications* - an array of applications on this page. This is an array of *Applications* models objects.

You can provide *options* as an object with this parameters:

- *page* - by default will 1.
- *name* - search applications by name.
- ***sort* - sorting parameter, looks like "*fieldname,order*". Example: "*name,asc*"**

  Possible sort fields:

  - *name*
  - *created*

  Possible order types:

  - *asc*
  - *desc*

Note: Available for:

- User token

Get application by id

You can get application by its id:

```
let api = new SensorlabApi();
api.applications.get(application_id)
         .then((application) => {
            console.log(application);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success you will get promise with *Application* model object.

```
let api = new SensorlabApi();
api.applications.get(application_id)
         .then((application) => {
            console.log(application.id);
            console.log(application.name);
            console.log(application.description);
            console.log(application.created);
            console.log(application.public_api_key);
         });
```

Parameters of *Application*:

- *id* - id in the database.
- *name* - application name.
- *description* - application description.
- *created* - application's creation date.
- *public_api_key* - Public Api Key

Note: Available for:

- User token

Create application

This is how you can create new application:

```
let api = new SensorlabApi();
api.applications.create(name, description)
          .then((application) => {
             console.log(application);
          })
          .catch((response) => {
             console.log(response);
          });
```

On success you will get promise with *Application* model object.

```
let api = new SensorlabApi();
api.applications.create(name, description)
          .then((application) => {
             console.log(application.id);
             console.log(application.name);
             console.log(application.description);
             console.log(application.created);
             console.log(application.public_api_key);
             console.log(application.private_api_key);
          });
```

Parameters of *Application*:

- *id* - id in the database.
- *name* - application name.
- *description* - application description.
- *created* - application's creation date.
- *public_api_key* - Public Api Key
- *private_api_key* - Generated private API key.

Note: Please note, that Private API Key will appear only once after app creation. You will be able to generate new one though.

Codes and messages for validation errors:

- *code=1 - "Please, provide name field. This cannot be empty."*

Note: Available for:

- User token

Update application

This is how you can update applications:

```
let api = new SensorlabApi();
api.applications.update(application_id, name, description)
         .then((response) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

If success you will get promise with response type ApiResponse with *code=100*, *status=200* and *success=true*

```
let api = new SensorlabApi();
api.applications.update(application_id, name, description)
         .then((response) => {
            console.log(response.code);
            console.log(response.status);
            console.log(response.success);
            console.log(response.message);
         });
```

Codes and messages for validation errors:

- *code=1 - "Please, provide name field. This cannot be empty."*

Note: Available for:

- User token

Delete application

This is how you can delete applications:

```
let api = new SensorlabApi();
api.applications.delete(application_id)
         .then((response) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

If success you will get promise with response type ApiResponse with *code=100*, *status=200* and *success=true*

```
let api = new SensorlabApi();
api.applications.delete(application_id)
         .then((response) => {
            console.log(response.code);
            console.log(response.status);
            console.log(response.success);
            console.log(response.message);
         });
```

Note: Available for:

- User token
- Application token

Generate new Private Api Key for application

This is how you can generate new Private Api Key for application:

```
let api = new SensorlabApi();
api.applications.generate_private_api_key(application_id)
         .then((application) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success you will get promise with *Application* model object.

```
let api = new SensorlabApi();
api.applications.create(name, description)
         .then((application) => {
            console.log(application.id);
            console.log(application.name);
            console.log(application.description);
            console.log(application.created);
            console.log(application.public_api_key);
            console.log(application.private_api_key);
         });
```

Parameters of *Application*:

- *id* - id in the database.
- *name* - application name.
- *description* - application description.
- *created* - application's creation date.
- *public_api_key* - Public Api Key
- *private_api_key* - Generated private API key.

Note: Available for:

- User token

1.1.6 Sensors Actions

Get sensors list

You can get sensors list for authenticated user like this:

```
let api = new SensorlabApi();
api.sensors.list(options)
         .then((sensors_response) => {
            console.log(sensors_response);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success you will get promise with *SensorsResponse* object.

```
let api = new SensorlabApi();
api.sensors.list(options)
         .then((sensors_response) => {
            console.log(sensors_response.page);
            console.log(sensors_response.count);
            console.log(sensors_response.sensors);
         });
```

Parameters of *SensorsResponse*:

- *page* - amount of pages available
- *count* - amount of sensors
- *sensors* - an array of sensors on this page. This is an array of *Sensor* models objects.

Parameters of *Sensor*:

- *id* - id in the database.
- *imei* - Sensor's imei.
- *name* - Sensor's name.
- *application* - Application ID sensor is connected to.
- *batteryCharge* - Sensor's battery charge in percent.
- *isBatteryCharging* - Indicates if battery is charging or not.
- *isOnline* - Indicates if sensor is online and sending data or not.
- *is_public* - Show if sensor is public or not.

You can provide *options* as an object with this parameters:

- *page* - by default will 1.
- *name* - filter sensors by name.
- *id* - filter by ID.
- *imei* - filter by imei.
- *online_status* - pass "online" to search for online sensors or "offline" for offline sensors.
- *battery_charge_min* - filter sensors by battery charge.
- *battery_charge_max* - filter sensors by battery charge.
- ***sort* - sorting parameter, looks like "*fieldname,order*". Example: "*name,asc*"**

  Possible sort fields:

  - *name*

- *created*

Possible order types:

- *asc*
- *desc*

All options filter parameters can be used at the same time, but it will search with "AND" condition.

Note: Available for:

- User token
- Application token

Application token will have access only to sensors assigned to this application.

Get sensor by id

You can get sensor by its id:

```
let api = new SensorlabApi();
api.sensors.get(sensor_id)
          .then((sensor) => {
             console.log(sensor);
          })
          .catch((response) => {
             console.log(response);
          });
```

On success you will get promise with *Sensor* model object.

```
let api = new SensorlabApi();
api.sensors.get(sensor_id)
          .then((sensor) => {
             console.log(sensors_response.id);
             console.log(sensors_response.imei);
             console.log(sensors_response.name);
          });
```

Parameters of *Sensor*:

- *id* - id in the database.
- *imei* - Sensor's imei.
- *name* - Sensor's name.
- *application* - Application ID sensor is connected to.
- *batteryCharge* - Sensor's battery charge in percent.
- *isBatteryCharging* - Indicates if battery is charging or not.
- *isOnline* - Indicates if sensor is online and sending data or not.

Note: Available for:

- User token

- Application token

Application token will have access only to sensors assigned to this application.

Update sensor

This is how you can update sensors:

```
let api = new SensorlabApi();
api.sensors.update(sensor_id, name, application)
          .then((response) => {
             console.log(response);
          })
          .catch((response) => {
             console.log(response);
          });
```

If success you will get promise with response type ApiResponse with *code=100*, *status=200* and *success=true*

```
let api = new SensorlabApi();
api.sensors.update(sensor_id, name, application)
          .then((response) => {
             console.log(response.code);
             console.log(response.status);
             console.log(response.success);
             console.log(response.message);
          });
```

Codes and messages for validation errors:

- code=1 - field=name - Please, provide name field. This cannot be empty
- code=2 - field=application - *application* must be correct UUID format
- code=3 - field=is_public - *is_public* must be correct boolean format

Note: Available for:

- User token
- Application token

Application token will have access only to sensors assigned to this application.

Get measurements for sensor

Every *Sensor* model has access to the measurement methods. You can get list of measurements for specified sensor:

```
let api = new SensorlabApi();
api.sensors.one(sensor_id)
          .then((sensor) => {
             sensor.measurements.list(options).then((measurements_result) => {
                 console.log(measurements_result);
             });
          });
```

On success you will get promise with *MeasurementsResponse* object.

```
sensor.measurements.list(options)
         .then((measurements_response) => {
            console.log(measurements_response.page);
            console.log(measurements_response.count);
            console.log(measurements_response.measurements);
         });
```

Parameters of *MeasurementsResponse*:

- *next* - next measurement id for pagination.
- *prev* - previous measurement id for pagination.
- *measurements* - an array of measurements on this page. This is an array of *Measurements* models objects.
- *timestamp_start* - filter by timestamp range.
- *timestamp_stop* - filter by timestamp range.

You can provide *options* as an object with this parameters:

- *next* - by default will null.
- *type* - filter by measurement type.
- *sensor_id* - filter by sensor id.

Codes and messages for validation errors:

- code=2 - field=sensor_id - This is not correct id format
- code=3 - field=next - This is not correct id format
- code=4 - field=timestamp_start - *timestamp_start* should be correct unix timestamp format
- code=5 - field=timestamp_stop - *timestamp_stop* should be correct unix timestamp format
- code=6 - field=timestamp_stop - *timestamp_stop* should be more or equal *timestamp_start*

Note: Available for:

- User token
- Application token

Application token will have access only to measurements of sensors assigned to this application.

Get last measurement for sensor

Every *Sensor* model has access to the measurement methods. You can get last measurement for specified sensor:

```
let api = new SensorlabApi();
api.sensors.one(sensor_id)
         .then((sensor) => {
            sensor.measurements.last(options).then((measurement) => {
                console.log(measurement);
            });
         });
```

On success you will get promise with *Measurement* model object.

```
sensor.measurements.last(options)
         .then((measurement) => {
            console.log(measurement.type);
            console.log(measurement.value);
            console.log(measurement.timestamp);
         });
```

Parameters of *Measurement* model object:

- *type* - measurement type
- *value* - value array
- *timestamp* - timestamp measurement was created

You can provide *options* as an object with this parameters:

- *type* - filter by measurement type.

Note: Available for:

- User token
- Application token

Application token will have access only to measurements of sensors assigned to this application.

1.1.7 Measurement Actions

Get measurements

You can get measurments without being attached to sensor:

```
let api = new SensorlabApi();
api.measurements.list(options)
    .then((measurements_response) => {
        console.log(measurements_response);
    });
});
```

On success you will get promise with *MeasurementsResponse* object.

```
api.measurements.list(options)
    .then((measurements_response) => {
        console.log(measurements_response.next);
        console.log(measurements_response.prev);
        console.log(measurements_response.measurements);
    });
});
```

Parameters of *MeasurementsResponse*:

- *next* - next measurement id for pagination.
- *prev* - previous measurement id for pagination.
- *measurements* - an array of measurements on this page. This is an array of *Measurements* models objects.
- *timestamp_start* - filter by timestamp range.

- *timestamp_stop* - filter by timestamp range.

You can provide *options* as an object with this parameters:

- *next* - by default will null.
- *type* - filter by measurement type.
- *sensor* - filter by sensor id.
- *timestamp_start* - filter by timestamp range - range start.
- *timestamp_stop* - filter by timestamp range - range stop.

Codes and messages for validation errors:

- code=1 - field=sensor_id - Please, provide sensor field. This cannot be empty
- code=2 - field=sensor_id - This is not correct id format
- code=3 - field=next - This is not correct id format
- code=4 - field=timestamp_start - *timestamp_start* should be correct unix timestamp format
- code=5 - field=timestamp_stop - *timestamp_stop* should be correct unix timestamp format
- code=6 - field=timestamp_stop - *timestamp_stop* should be more or equal *timestamp_start*

Note: Available for:

- User token
- Application token

Application token will have access only to measurements of sensors assigned to this application.

Get last measurement

You can get last measurment without being attached to sensor:

```
let api = new SensorlabApi();
api.measurements.last(options).then((measurement) => {
    console.log(measurement);
});
```

On success you will get promise with *Measurement* model object.

```
let api = new SensorlabApi();
api.measurements.last(options)
         .then((measurement) => {
            console.log(measurement.type);
            console.log(measurement.value);
            console.log(measurement.timestamp);
         });
```

Parameters of *Measurement* model object:

- *type* - measurement type
- *value* - value array
- *timestamp* - timestamp measurement was created

You can provide *options* as an object with this parameters:

- *sensor* - filter by sensor.
- *type* - filter by measurement type.

Codes and messages for validation errors:

- *code=1 - field=sensor_id - Please, provide sensor field. This cannot be empty.*.
- *code=2 - field=sensor_id - This is not correct id format.*.

Note: Available for:

- User token
- Application token

Application token will have access only to measurements of sensors assigned to this application.

1.1.8 Public Sensors Measurements Actions

Get measurements for public sensor

You can get access to *Measurements* for public *Sensor* using only Public Api Key from your *Application*.

On success you will get promise with *MeasurementsResponse* object.

```
sensor.public.list(public_api_key, options)
         .then((measurements_response) => {
            console.log(measurements_response.page);
            console.log(measurements_response.count);
            console.log(measurements_response.measurements);
         });
```

Parameters of *MeasurementsResponse*:

- *next* - next measurement id for pagination.
- *prev* - previous measurement id for pagination.
- *measurements* - an array of measurements on this page. This is an array of *Measurements* models objects.
- *timestamp_start* - filter by timestamp range.
- *timestamp_stop* - filter by timestamp range.

You can provide *options* as an object with this parameters:

- *next* - by default will null.
- *type* - filter by measurement type.
- *sensor_id* - filter by sensor id.

Codes and messages for validation errors:

- code=2 - field=sensor_id - This is not correct id format
- code=3 - field=next - This is not correct id format
- code=4 - field=timestamp_start - *timestamp_start* should be correct unix timestamp format
- code=5 - field=timestamp_stop - *timestamp_stop* should be correct unix timestamp format

- code=6 - field=timestamp_stop - *timestamp_stop* should be more or equal *timestamp_start*

Get last measurement for public sensor

You can get access to *Measurements* for public *Sensor* using only Public Api Key from your *Application*.

On success you will get promise with *Measurement* model object.

```
sensor.public.last(public_api_key, options)
          .then((measurement) => {
             console.log(measurement.type);
             console.log(measurement.value);
             console.log(measurement.timestamp);
          });
```

Parameters of *Measurement* model object:

- *type* - measurement type
- *value* - value array
- *timestamp* - timestamp measurement was created

You can provide *options* as an object with this parameters:

- *type* - filter by measurement type.

1.1.9 Sensor Alerts Actions

Get sensor alerts configurations

You can get list of sensor alerts configuration:

```
let api = new SensorlabApi();
api.sensor_alerts.list(sensor)
    .then((sensor_alerts_response) => {
        console.log(sensor_alerts_response);
    });
});
```

On success you will get promise with *SensorAlertsResponse* object.

```
api.sensor_alerts.list(options)
    .then((sensor_alerts_response) => {
        console.log(sensor_alerts_response.sensor_alerts);
    });
});
```

Method parameters:

- *sensor* - Sensor's ID

Parameters of *SensorAlertsResponse*:

- *sensor_alerts* - an array of sensor alerts. This is an array of *SensorAlerts* models objects.

Note: Available for:

- User token

- Application token

Get sensor alert configuration by id

You can get sensor alert configuration by its id:

```
let api = new SensorlabApi();
api.sensor_alerts.get(sensor, alert)
         .then((sensor_alert) => {
            console.log(sensor_alert);
         })
         .catch((response) => {
            console.log(response);
         });
```

On success you will get promise with *SensorAlert* model object.

```
let api = new SensorlabApi();
api.sensor_alerts.get(sensor, alert)
         .then((sensor_alert) => {
            console.log(sensor_alert.id);
            console.log(sensor_alert.threshold_type);
            console.log(sensor_alert.measurement_type);
            console.log(sensor_alert.threshold_value);
         });
```

Method parameters:

- *sensor* - Sensor's ID
- *alert* - Alert's ID

Parameters of *SensorAlert*:

- *id* - id in the database.
- *threshold_type* - threshold type.
- *measurement_type* - measurement type.
- *threshold_value* - threshold value.

Note: Available for:

- User token
- Application token

Create sensor alert configuration

This is how you can create new sensor alert configuration:

```
let api = new SensorlabApi();
api.sensor_alerts.create(sensor, threshold_type, measurement_type, threshold_value)
         .then((sensor_alert) => {
            console.log(sensor_alert);
```

(continues on next page)

(continued from previous page)

```
        })
        .catch((response) => {
           console.log(response);
        });
```

On success you will get promise with *SensorAlert* model object.

```
let api = new SensorlabApi();
api.sensor_alerts.create(sensor, threshold_type, measurement_type, threshold_value)
        .then((sensor_alert) => {
           console.log(sensor_alert.id);
           console.log(sensor_alert.threshold_type);
           console.log(sensor_alert.measurement_type);
           console.log(sensor_alert.threshold_value);
        });
```

Method parameters:

- *sensor* - Sensor's ID
- *threshold_type* - threshold type.
- *measurement_type* - measurement type.
- *threshold_value* - threshold value.

Parameters of *SensorAlert*:

- *id* - id in the database.
- *threshold_type* - threshold type.
- *measurement_type* - measurement type.
- *threshold_value* - threshold value.

Codes and messages for validation errors:

- code=2 - field=sensor - *sensor* field should be a correct UUID format
- code=3 - field=threshold_type - *threshold_type* is required
- code=4 - field=threshold_type - *threshold_type* must be *min*, *max* or *loc*
- code=5 - field=measurement_type - *measurement_type* is required
- code=6 - field=threshold_value - *threshold_value* is required
- code=7 - field=threshold_value - *threshold_value* for *threshold_type* "min" must be a correct float
- code=8 - field=threshold_value - *threshold_value* for *threshold_type* "max" must be a correct float
- code=9 - field=threshold_value - *threshold_value* for *threshold_type* "loc" must be a correct JSON object with *lat*, *lng* and *radius* parameters
- code=10 - field=threshold_value - *threshold_value.lat* must be correct latitude
- code=11 - field=threshold_value - *threshold_value.lng* must be correct longitude
- code=12 - field=threshold_value - *threshold_value.radius* must be correct positive integer

---

Note: Available for:

- User token

- Application token

Update sensor alert configuration

This is how you can update new sensor alert configuration:

```
let api = new SensorlabApi();
api.sensor_alerts.update(sensor, alert, threshold_type, measurement_type, threshold_
↪value)
         .then((response) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

If success you will get promise with response type ApiResponse with *code=100*, *status=200* and *success=true*

```
let api = new SensorlabApi();
api.sensor_alerts.update(sensor, alert, threshold_type, measurement_type, threshold_
↪value)
         .then((response) => {
            console.log(response.code);
            console.log(response.status);
            console.log(response.success);
            console.log(response.message);
         });
```

Method parameters:

- *sensor* - Sensor's ID
- *alert* - Alert's ID.
- *threshold_type* - threshold type.
- *measurement_type* - measurement type.
- *threshold_value* - threshold value.

Codes and messages for validation errors:

- code=2 - field=sensor - *sensor* field should be a correct UUID format
- code=4 - field=id - *id* field should be a correct UUID format
- code=5 - field=threshold_type - *threshold_type* is required
- code=6 - field=threshold_type - *threshold_type* must be *min*, *max* or *loc*
- code=7 - field=measurement_type - *measurement_type* is required
- code=8 - field=threshold_value - *threshold_value* is required
- code=9 - field=threshold_value - *threshold_value* for *threshold_type* "min" must be a correct float
- code=10 - field=threshold_value - *threshold_value* for *threshold_type* "max" must be a correct float
- code=11 - field=threshold_value - *threshold_value* for *threshold_type* "loc" must be a correct JSON object with *lat*, *lng* and *radius* parameters
- code=12 - field=threshold_value - *threshold_value.lat* must be correct latitude

- code=13 - field=threshold_value - *threshold_value.lng* must be correct longitude
- code=14 - field=threshold_value - *threshold_value.radius* must be correct positive integer

---

Note: Available for:

- User token
- Application token

Delete sensor alert configuration

This is how you can delete sensor alert configuration:

```
let api = new SensorlabApi();
api.sensor_alert.delete(sensor, alert)
         .then((response) => {
            console.log(response);
         })
         .catch((response) => {
            console.log(response);
         });
```

If success you will get promise with response type *ApiResponse* with *code=100*, *status=200* and *success=true*

```
let api = new SensorlabApi();
api.sensor_alert.delete(sensor, alert)
         .then((response) => {
            console.log(response.code);
            console.log(response.status);
            console.log(response.success);
            console.log(response.message);
         });
```

Codes and messages for validation errors:

- code=2 - field=sensor - *sensor* field should be a correct UUID format
- code=4 - field=id - *id* field should be a correct UUID format

---

Note: Available for:

- User token
- Application token

Get last generated alerts for sensor

You can get list of last sensor alerts:

```
let api = new SensorlabApi();
api.alerts.last(sensor)
    .then((alerts_response) => {
        console.log(alerts_response);
```

(continues on next page)

(continued from previous page)

```
    });
});
```

On success you will get promise with *AlertsResponse* object.

```
api.alerts.last(options)
    .then((alerts_response) => {
        console.log(alerts_response.alerts);
    });
});
```

Method parameters:

- *sensor* - Sensor's ID

Parameters of *AlertsResponse*:

- *alerts* - an array of alerts on this page. This is an array of *Alerts* models objects.

Note: Available for:

- User token
- Application token

1.2 How to use SDK for Websockets

1.2.1 Getting Started

This is documentation on Javascript SDK for http://sensorlab.io/ Websockets.

This SDK can be easily used in NodeJS and as JS library to use in browser.

Websockets are implemented based on socket.io

With this implementation you can receive measurements and alerts from sensors in real-time.

Installation

Install with npm:

```
$ npm i --save https://github.com/sensorlabio/sensorlabio-javascript-sdk
```

Use with NodeJS

Initialize inside your js code

```
import {SensorlabWebsockets} from "sensorlabio-javascript-sdk";
let measurements_ws = new ws.measurements();
let alerts_ws = new ws.alerts();
let public_ws = new ws.public();
```

You can specify another url to websockets:

```
let measurements_ws = new ws.measurements('http://testing.sensorlab.io:8080/');
```

For React you must use another build for web:

import {SensorlabWebsockets} from "sensorlabio-javascript-sdk/build/web.min";

Use with browsers

Download latest master release:

https://github.com/sensorlabio/sensorlabio-javascript-sdk/archive/master.zip

Unzip:

- *builds/web.min.js*

Use them inside your html code:

```
<script src="build/web.min.js"></script>
```

And inside your JS code:

```
<script type="text/javascript">
    var sdk = SensorlabSDK;
    var ws = new sdk.SensorlabWebsockets();
</script>
```

1.2.2 Measurements

You can receive measurements using websockets.

In order to connect to websockets you should generate correct JWT.

```
let api = new SensorlabApi();
api.auth.application_token(public_api_key, private_api_key)
         .then((application_token) => {
            console.log(application_token);
         })
         .catch((response) => {
            console.log(response);
         });
```

When you have token you can connect to /measurements namespace:

```
var ws = new sdk.SensorlabWebsockets();
let measurements_ws = new ws.measurements();

measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
}).catch((error) => {
    console.log(error);
});
```

If there are problems with connection or authentication promise will throw an exception.

```
measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
}).catch((error) => {
    console.log(error.code);
    console.log(error.message);
});
```

In case authentication problems exception code will be "401" with message "Authentication error. Please check your token."

If connection problem - SDK will throw exception with code "0" and message "Connection error"

When connected, you can join sensor rooms. There are 2 types of rooms:

- measurements for sensor
- measurements by type for sensor

Connect sensor room:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
    measurements_ws.joinSensor(sensor);
});
```

Connect to sensor/type room:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
    measurements_ws.joinSensor(sensor, 'TMP');
});
```

If user/application with this token doesn't have rights to the sensor, a "sensor/access_denied" message will be emitted.

You can catch this message with listener:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedMeasurements(sensor, message) {
    console.log('access denied', sensor, message);
}

measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
    measurements_ws.joinSensor(sensor, 'TMP');
    measurements_ws.onAccessDenied(onAccessDeniedMeasurements);
});
```

If there are no problems - you can connect listeners to rooms:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedMeasurements(sensor, message) {
    console.log('access denied', sensor, message);
}
```

(continues on next page)

(continued from previous page)

```
function getMeasurements(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.type);
        console.log(measurement.value);
    });
}

function getMeasurementsTMP(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.value);
    });
}

measurements_ws.connect(token).then(() => {
    console.log('Connected and authenticated to measurements');
    measurements_ws.joinSensor(sensor);
    measurements_ws.onMeasurements(sensor, null, getMeasurements);
    measurements_ws.onMeasurements(sensor, 'TMP', getMeasurementsTMP);
    measurements_ws.onAccessDenied(onAccessDeniedMeasurements);
});
```

You can connect any amount of listeners to each room.

Do disable listener, use *offMeasurements* method:

```
function getMeasurements(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.type);
        console.log(measurement.value);
    });
}

measurements_ws.offMeasurements(sensor, null, getMeasurements);
```

You can leave sensor room like this:

```
measurements_ws.leaveSensor(sensor, 'TMP');
```

To leave all rooms run this code:

```
measurements_ws.leaveAll();
```

Note: Leaving rooms will stop data to be emitted, but listeners will still be connected.

1.2.3 Alerts

You can receive alerts using websockets.

In order to connect to websockets you should generate correct JWT.

```
let api = new SensorlabApi();
api.auth.application_token(public_api_key, private_api_key)
         .then((application_token) => {
            console.log(application_token);
         })
         .catch((response) => {
            console.log(response);
         });
```

When you have token you can connect to /alerts namespace:

```
var ws = new sdk.SensorlabWebsockets();
let alerts_ws = new ws.alerts();

alerts_ws.connect(token).then(() => {
    console.log('Connected and authenticated to alerts');
}).catch((error) => {
    console.log(error);
});
```

If there are problems with connection or authentication promise will throw an exception.

```
alerts_ws.connect(token).then(() => {
    console.log('Connected and authenticated to alerts');
}).catch((error) => {
    console.log(error.code);
    console.log(error.message);
});
```

In case authentication problems exception code will be “401” with message “Authentication error. Please check your token.”

If connection problem - SDK will throw exception with code “0” and message “Connection error”

When connected, you can join sensor rooms.

Connect sensor room:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

alerts_ws.connect(token).then(() => {
    console.log('Connected and authenticated to alerts');
    alerts_ws.joinSensor(sensor);
});
```

If user/application with this token doesn’t have rights to the sensor, a sensor/access_denied message will be emitted.

You can catch this message with listener:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedAlerts(sensor, message) {
    console.log('access denied', sensor, message);
}

alerts_ws.connect(token).then(() => {
    console.log('Connected and authenticated to alerts');
    alerts_ws.joinSensor(sensor, 'TMP');
```

(continues on next page)

(continued from previous page)

```
    alerts_ws.onAccessDenied(onAccessDeniedAlerts);
});
```

If there are no problems - you can connect listeners to rooms:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedAlerts(sensor, message) {
    console.log('access denied', sensor, message);
}

function getAlerts(alerts) {
    alerts.forEach((alert) => {
        console.log(alert.measurement.timestamp);
        console.log(alert.measurement.type);
        console.log(alert.measurement.value);

        console.log(alert.threshold.measurement_type);
        console.log(alert.threshold.threshold_type);
        console.log(alert.threshold.threshold_value);
    });
}

alerts_ws.connect(token).then(() => {
    console.log('Connected and authenticated to alerts');
    alerts_ws.joinSensor(sensor);
    alerts_ws.onAlerts(sensor, getAlerts);
    alerts_ws.onAccessDenied(onAccessDeniedAlerts);
});
```

You can connect any amount of listeners to each room.

Do disable listener, use *offAlerts* method:

```
function getAlerts(alerts) {
    alerts.forEach((alert) => {
        console.log(alert.measurement.timestamp);
        console.log(alert.measurement.type);
        console.log(alert.measurement.value);

        console.log(alert.threshold.measurement_type);
        console.log(alert.threshold.threshold_type);
        console.log(alert.threshold.threshold_value);
    });
}

alerts_ws.offAlerts(sensor, getAlerts);
```

You can leave sensor room like this:

```
alerts_ws.leaveSensor(sensor);
```

To leave all rooms run this code:

```
alerts_ws.leaveAll();
```

Note: Leaving rooms will stop data to be emitted, but listeners will still be connected.

1.2.4 Public Measurements

You can receive measurements from public sensors using websockets.

In order to connect to websockets you should get public api key for your application.

With public api key you can connect to /public namespace:

```
var ws = new sdk.SensorlabWebsockets();
let public_ws = new ws.public();

public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
}).catch((error) => {
    console.log(error);
});
```

If there are problems with connection or authentication promise will throw an exception.

```
public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
}).catch((error) => {
    console.log(error.code);
    console.log(error.message);
});
```

In case authentication problems exception code will be “401” with message “Authentication error. Please check your token.”

If connection problem - SDK will throw exception with code “0” and message “Connection error”

When connected, you can join sensor rooms. There are 2 types of rooms:

- measurements for sensor
- measurements by type for sensor

Connect sensor room:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
    public_ws.joinSensor(sensor);
});
```

Connect to sensor/type room:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
    public_ws.joinSensor(sensor, 'TMP');
});
```

If user/application with this token doesn't have rights to the sensor and sensor is not public - a sensor/access_denied message will be emitted.

You can catch this message with listener:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedPublicMeasurements(sensor, message) {
    console.log('access denied', sensor, message);
}

public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
    public_ws.joinSensor(sensor, 'TMP');
    public_ws.onAccessDenied(onAccessDeniedPublicMeasurements);
});
```

If there are no problems - you can connect listeners to rooms:

```
let sensor = '7e9fb090-c717-11e8-b4d3-c587309ce935';

function onAccessDeniedPublicMeasurements(sensor, message) {
    console.log('access denied', sensor, message);
}

function getMeasurements(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.type);
        console.log(measurement.value);
    });
}

function getMeasurementsTMP(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.value);
    });
}

public_ws.connect(token).then(() => {
    console.log('Connected and authenticated to public');
    public_ws.joinSensor(sensor);
    public_ws.onMeasurements(sensor, null, getMeasurements);
    public_ws.onMeasurements(sensor, 'TMP', getMeasurementsTMP);
    public_ws.onAccessDenied(onAccessDeniedPublicMeasurements);
});
```

You can connect any amount of listeners to each room.

Do disable listener, use *offMeasurements* method:

```
function getMeasurements(measurements) {
    measurements.forEach((measurement) => {
        console.log(measurement.timestamp);
        console.log(measurement.type);
        console.log(measurement.value);
    });
}
```

(continues on next page)

(continued from previous page)

```
public_ws.offMeasurements(sensor, null, getMeasurements);
```

You can leave sensor room like this:

```
public_ws.leaveSensor(sensor, 'TMP');
```

To leave all rooms run this code:

```
public_ws.leaveAll();
```

Note: Leaving rooms will stop data to be emitted, but listeners will still be connected.

CHAPTER 2

Specifications

2.1 REST API SDK classes

2.1.1 SensorlabApi class

class SensorlabApi()

Main class that connects all the endpoints.

SensorlabApi.**users**
type: UsersEndpoints

Users endpoint.

SensorlabApi.**auth**
type: AuthEndpoints

Auth endpoints.

SensorlabApi.**profile**
type: ProfileEndpoints

Profile endpoints.

SensorlabApi.**sensors**
type: SensorsEndpoints

Sensor endpoint.

SensorlabApi.**measurements**
type: MeasurementsEndpoints

Measurements endpoints.

SensorlabApi.**jwt_token**
type: string

Saved JWT token

SensorlabApi.**setToken**(*jwt_token*)
Set authentication jwt token.

Arguments

- **jwt_token** (*string*) – saved token

SensorlabApi.**getToken**()
Get authentication jwt token.

Returns **string** –

SensorlabApi.**_makeApiRequest**(*endpoint_url*, *method*, *data*, *jwt_token*)
Make requests to API with axios.

Arguments

- **endpoint_url** (*string*) – endpoint to send request to
- **method** (*string*) – method type
- **data** (*object*) – data to send with request
- **jwt_token** (*boolean*) – use request with token or not

Returns **Promise.<(*|InterceptorManager|AxiosResponse|AxiosInterceptorManager.<AxiosResponse>|Object)>** –

SensorlabApi.**_prepareApiResponse**(*response*, *success*)
Create ApiResponse from axios response data.

Arguments

- **response** (*object*) – response from axios
- **success** (*function*) – callback

Throws ***ApiResponse*** –

Returns **ApiResponse** –

2.1.2 Endpoint classes

AuthEndpoint class

class AuthEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

AuthEndpoints.**user_token**(*email*, *password*)
Authorize with email and password and get authentication token.

Arguments

- **email** (*string*) – user's email
- **password** (*string*) – user's password

Returns **Promise.<ApiResponse>** –

AuthEndpoints.**application_token**(*public_api_key*, *private_api_key*)
Authenticate application by public and private api keys and get JWT token.

Arguments

- **public_api_key** (*string*) – Application's public api key
- **private_api_key** (*string*) – Application's private api key

Returns **Promise.<ApiResponse>** –

ProfileEndpoint class

class ProfileEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

ProfileEndpoints.**get**()
Get user's profile.

Returns **Promise.<ApiResponse>** –

ProfileEndpoints.**change_password**(*old_password*, *new_password*, *new_password_check*)
Change password.

Arguments

- **old_password** (*string*) – current (old) user's password
- **new_password** (*string*) – new password to set
- **new_password_check** (*string*) – new password check

Returns **Promise.<ApiResponse>** –

ApplicationsEndpoint class

class ApplicationsEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

ApplicationsEndpoints.**list**(*options*)
Get applications list

Arguments

- **options** (*object*) – method options
- **options.page** (*number*) – page number to display. Default is *1*.
- **options.name** (*string*) – search by name.
- **options.sort** (*string*) – sorting parameter

Returns **Promise.<ApiResponse>** –

ApplicationsEndpoints.**get**(*application_id*)
Get application by id

Arguments

- **application_id** (*string*) – application_id

Returns **Promise.<ApiResponse>** –

ApplicationsEndpoints.**create**(*name*, *description*)
Create application.

Arguments

- **name** (*string*) – application's name
- **description** (*string*) – users's description

ApplicationsEndpoints.**update**(*application_id*, *name*, *description*)
Update application.

Arguments

- **application_id** (*string*) – id of application to update
- **name** (*string*) – application's name
- **description** (*string*) – users's description

ApplicationsEndpoints.**delete**(*application_id*)
Delete application by id.

Arguments

- **application_id** (*string*) – application_id

Returns **Promise.<ApiResponse>** –

ApplicationsEndpoints.**get_self**()
Get application for application token.

Returns **Promise.<ApiResponse>** –

ApplicationsEndpoints.**update_self**(*name*, *description*)
Update application for application token.

Arguments

- **name** (*string*) – application's name
- **description** (*string*) – users's description

SensorsEndpoint class

class SensorsEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

SensorsEndpoints.**list**(*options*)
Get sensors list.

Arguments

- **options** (*object*) – method options
- **options.page** (*number*) – page number to display. Default is *1*.
- **options.name** (*string*) – filter by name.
- **options.id** (*string*) – filter by id.
- **options.imei** (*string*) – filter by imei.
- **options.sort** (*string*) – sorting parameter
- **options.online_status** (*string*) – pass "online" to search for online sensors or "offline" for offline sensors.

- **options.battery_charge_min** (*string*) – filter sensors by battery charge
- **options.battery_charge_max** (*string*) – filter sensors by battery charge

Returns Promise.<ApiResponse> –

SensorsEndpoints.**update**(*sensor_id*, *name*, *application*, *is_public*)
Update sensors.

Arguments

- **sensor_id** (*string*) – id of sensor to update
- **name** (*string*) – sensor's name
- **application** (*string*) – application id to assign to
- **is_public** (*boolean*) – set sensor private or public

MeasurementsEndpoint class

class MeasurementsEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

MeasurementsEndpoints.**list**(*options*)
Get sensors list

Arguments

- **options** (*object*) – method options
- **options.page** (*number*) – page number to display. Default is *1*.
- **options.sensor** (*string*) – show measurements for sensor with specified id.
- **options.type** (*string*) – get measurements with specified type only.
- **options.sort** (*string*) – sorting parameter

Returns Promise.<ApiResponse> –

MeasurementsEndpoints.**last**(*options*, *type*)
Get sensor by id

Arguments

- **options** (*object*) – method options
- **options.sensor_id** (*string*) – get measurement for sensor with specified id.
- **type** (*string*) – get measurement with specified type only.

Returns Promise.<ApiResponse> –

PublicEndpoint class

class PublicEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

PublicEndpoints.**list**(*public_api_key*, *options*)
Get sensors list

Arguments

- **public_api_key** (*string*) –
- **options** (*object*) – method options
- **options.page** (*number*) – page number to display. Default is *1*.
- **options.sensor** (*string*) – show measurements for sensor with specified id.
- **options.type** (*string*) – get measurements with specified type only.
- **options.sort** (*string*) – sorting parameter

Returns **Promise.<ApiResponse>** –

PublicEndpoints.**last**(*public_api_key*, *options*, *type*)
Get sensor by id

Arguments

- **public_api_key** (*string*) –
- **options** (*object*) – method options
- **options.sensor_id** (*string*) – get measurement for sensor with specified id.
- **type** (*string*) – get measurement with specified type only.

Returns **Promise.<ApiResponse>** –

SensorAlertsEndpoints class

class SensorAlertsEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

SensorAlertsEndpoints.**list**(*sensor*)
Get sensor alerts list

Arguments

- **sensor** (*string*) – Sensor ID.

Returns **Promise.<ApiResponse>** –

SensorAlertsEndpoints.**get**(*sensor*, *alert*)
Get sensor alert by id

Arguments

- **sensor** (*string*) – Sensor ID
- **alert** (*string*) – Sensor Alert ID

Returns **Promise.<ApiResponse>** –

SensorAlertsEndpoints.**create**(*threshold_type*, *measurement_type*, *threshold_value*)
Create sensor alert.

Arguments

- **threshold_type** (*string*) – Threshold type

- **measurement_type** (*string*) – Measurement type
- **threshold_value** (*mixed*) – Threshold value

SensorAlertsEndpoints.**update**(*sensor*, *alert*, *threshold_type*, *measurement_type*, *threshold_value*)

Update sensor alert.

Arguments

- **sensor** (*string*) – Sensor ID
- **alert** (*string*) – Sensor Alert ID
- **threshold_type** (*string*) – Threshold type
- **measurement_type** (*string*) – Measurement type
- **threshold_value** (*string|number|Object*) – Threshold value

SensorAlertsEndpoints.**delete**(*sensor*, *alert*)

Delete sensor alert.

Arguments

- **sensor** (*string*) – Sensor ID
- **alert** (*string*) – Sensor Alert ID

Returns **Promise.<ApiResponse>** –

AlertsEndpoints class

class AlertsEndpoints(*api*)

Arguments

- **api** (SensorlabApi) – parent api

AlertsEndpoints.**list**(*sensor*)

Get last alerts for sensor

Arguments

- **sensor** (*string*) – Sensor ID.

Returns **Promise.<ApiResponse>** –

2.1.3 Models classes

User class

class User(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

User.**token**

type: string

Authorization token.

`User.profile()`
Get user's profile.

Returns **Promise.<ApiResponse>** –

ApplicationToken class

class ApplicationToken(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

`ApplicationToken.token`
type: string

Authorization token.

Profile class

class Profile(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

`Profile.email`
type: string

User's email

`Profile.change_password`(*old_password*, *new_password*, *new_password_check*)
Change password for profile/user.

Arguments

- **old_password** –
- **new_password** –
- **new_password_check** –

Returns **Promise.<ApiResponse>** –

Applications class

class Application(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

`Application.id`
type: string

Application ID.

Application.**name**
type: string

Application name.

Application.**description**
type: string

Application name.

Application.**created**
type: *

Date/time created.

Application.**public_api_key**
type: string

Public api key for application.

Application.**private_api_key**
type: string

Private Api Key for created application.

Sensor class

class Sensor(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

Sensor.**id**
type: string

Sensor ID.

Sensor.**imei**
type: string

Sensor IMEI.

Sensor.**name**
type: string

Sensor name.

Sensor.**application**
type: string

Application ID.

Sensor.**created**
type: *

Date/time created.

Sensor.**batteryCharge**
type: number

Battery charge.

Sensor.**isBatteryCharging**
type: boolean

Is battery charging?

Sensor.**isOnline**
type: boolean

Is sensor connected and online?

Sensor.**measurements**
type: Object

Access to measurments methods for current sensor.

Sensor.**is_public**
type: boolean

Is sensor public?

Measurement class

class Measurement(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

Measurement.**type**
type: string

Measurement type.

Measurement.**value**
type: array

Measurement values array.

Measurement.**created**
type: Date

Created date.

Alert class

class Alert(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

Alert.**measurement**
type: object

Affected measurement

Alert.**threshold**
type: object

Threshold

Sensor alert class

class SensorAlert(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

SensorAlert.**id**
type: string

Sensor alert ID

SensorAlert.**threshold_type**
type: string

Threshold type

SensorAlert.**measurement_type**
type: string

Measurement type

SensorAlert.**threshold_value**
type: *

Threshold value

2.1.4 Response classes

Default ApiResponse class

class ApiResponse(*success*, *status*, *code*, *message*, *token*)

Arguments

- **success** (*boolean*) –
- **status** (*number*) –
- **code** (*number*) –
- **message** (*string*) –
- **token** (*string*) –

ApiResponse.**success**
type: boolean

Response was success or returned with errors

ApiResponse.**status**
type: number

HTTP status of response

ApiResponse.**code**
type: number

Response/error code

ApiResponse.**message**
type: string

Response/error message

ApiResponse.**errors**
type: Array

Validation errors.

ApplicationsResponse class

class ApplicationsResponse(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

ApplicationsResponse.**pages**
type: number

Total amount of pages available.

ApplicationsResponse.**count**
type: number

Total amount of sensors available.

ApplicationsResponse.**applications**
type: Array.<Application>

List of sensors.

SensorsResponse class

class SensorsResponse(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

SensorsResponse.**pages**
type: number

Total amount of pages available.

SensorsResponse.**count**
type: number

Total amount of sensors available.

SensorsResponse.**measurements**
type: Array.<Sensor>

List of sensors.

MeasurementsResponse class

class MeasurementsResponse(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

MeasurementsResponse.**next**
type: string

Next measurement id for pagination.

MeasurementsResponse.**prev**
type: string

Previous measurement id for pagination.

MeasurementsResponse.**measurements**
type: Array.<Measurement>

List of measurements.

AlertsResponse class

class AlertsResponse(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

AlertsResponse.**alerts**
type: Array.<Alert>

List of alerts.

SensorAlertsResponse class

class SensorAlertsResponse(*api*, *data*)

Arguments

- **api** (SensorlabApi) – parent api
- **data** (*Object*) – data from response

SensorAlertsResponse.**sensor_alerts**
type: Array.<SensorAlert>

List of sensors.

Error exceptions

Basic error exception

class ApiErrorBasicException(*status*, *message*, *errors*)

Arguments

- **status** (*number*) – HTTP status of response.
- **message** (*string*) – error message
- **errors** (*array*) – validation errors

ApiErrorBasicException.**status**
type: number

HTTP status of response

ApiErrorBasicException.**message**
type: string

Response/error message

ApiErrorBasicException.**errors**
type: Array

Validation errors.

Unauthorized exception

class ApiErrorUnauthorizedException()
SDK will throw this exception on 401 status errors.

Forbidden exception

class ApiErrorForbiddenException()
SDK will throw this exception on 403 status errors.

Not found exception

class ApiErrorNotFoundException()
SDK will throw this exception on 404 status errors.

Validation error exception

class ApiErrorValidationException()
SDK will throw this exception on 422 status (fields validation).

Internal error exception

class ApiErrorInteralException()
SDK will throw this exception on 500 status errors.

Connection refused exception

class ApiErrorConnectionRefusedException()
SDK will throw this exception if there's no connection to API.

2.2 Websockets classes

2.2.1 SensorlabWebsockets class

class SensorlabWebsockets()

SensorlabWebsockets.**measurements**
type: SensorlabMeasurementsWebsocket

Shortcut to SensorlabMeasurementsWebsocket class

SensorlabWebsockets.**alerts**
type: SensorlabAlertsWebsocket

Shortcut to SensorlabAlertsWebsocket class

SensorlabWebsockets.**public**
type: SensorlabPublicWebsocket

Shortcut to SensorlabPublicWebsocket class

2.2.2 BasicWebsocket class

class BasicWebsocket(*websocket_url*)

Arguments

- **websocket_url** (*string*) – websockets url to connect to

BasicWebsocket.**connect**
Connect to websocket namespace with authentication token. Promise will resolve if connection is established and reject is there are problems with connection or authentication

BasicWebsocket.**disconnect**
Disconnect from websocket

2.2.3 SensorlabMeasurementsWebsocket class

class SensorlabMeasurementsWebsocket(*websocket_url*)

Arguments

- **websocket_url** (*string*) – socket.io url to connect to

SensorlabMeasurementsWebsocket.**joinSensor**
Join room with measurement for specific sensor. You can also specify measurement type to listen too. If user has access to the sensor - socket.io will start emitting data for this sensor. If not - it will emit message that can be catched using onAccessDenied method.

SensorlabMeasurementsWebsocket.**leaveSensor**
Leave room with measurements for specific sensor.

SensorlabMeasurementsWebsocket.**leaveAll**
Leave all rooms.

SensorlabMeasurementsWebsocket.**onMeasurements**
Listen to specific emitted measurements.

SensorlabMeasurementsWebsocket.**offMeasurements**
Disconnect listener.

SensorlabMeasurementsWebsocket.**onAccessDenied**
This message will be emitted if your token doesn't have access to the sensor.

2.2.4 SensorlabAlertsWebsocket class

class SensorlabAlertsWebsocket(*websocket_url*)

Arguments

- **websocket_url** (*string*) – socket.io url to connect to

SensorlabAlertsWebsocket.**joinSensor**
Join room with alerts for specific sensor. If user has access to the sensor - socket.io will start emitting data for this sensor. If not - it will emit message that can be catched using onAccessDenied method.

SensorlabAlertsWebsocket.**leaveSensor**
Leave room with alerts for specific sensor.

SensorlabAlertsWebsocket.**leaveAll**
Leave all rooms.

SensorlabAlertsWebsocket.**onAlerts**
Listen to specific emitted alerts.

SensorlabAlertsWebsocket.**offAlerts**
Disconnect listener.

SensorlabAlertsWebsocket.**onAccessDenied**
This message will be emitted if your token doesn't have access to the sensor.

2.2.5 SensorlabPublicWebsocket class

class SensorlabPublicWebsocket(*websocket_url*)

Arguments

- **websocket_url** (*string*) – socket.io url to connect to

SensorlabPublicWebsocket.**connect**
Connect to websocket namespace with public api key token. Promise will resolve if connection is established and reject is there are problems with connection or authentication

SensorlabPublicWebsocket.**joinSensor**
Join room with measurement for specific sensor. You can also specify measurement type to listen too. If user has access to the sensor - socket.io will start emitting data for this sensor. If not - it will emit message that can be catched using onAccessDenied method.

SensorlabPublicWebsocket.**leaveSensor**
Leave room with measurements for specific sensor.

SensorlabPublicWebsocket.**leaveAll**
Leave all rooms.

SensorlabPublicWebsocket.**onMeasurements**
Listen to specific emitted measurements.

Index

P

S

U