



# Backend API

This page gathers all the API calls that can be used by the front end.

Front end -> Backend

## Global configuration

Collection of all functions/API calls available to the front end that handles the global variables.

### saveConfigParams(configParams)

#### saveConfigParams(configParams)

Saves the config parameters to the persistent file

##### React usage example

```
import { saveConfigParams } from './api/backend-api';  
  
saveConfigParams(globalConfig);
```

### loadConfigParams()

#### loadConfigParams()

Loads the config parameters from the persistent file

##### React usage example

```
import { loadConfigParams } from './api/backend-api';  
  
globalConfig = loadConfigParams();
```

# Data base

Collection of all functions/API calls available to the front end that handles the communication with the data base, such as fetching and storing data.

## getMaterialList()

### getMaterialList()

#### TODO

##### React usage example

```
import { getMaterialList } from "../api/backend-api";

globalConfig = getMaterialList();
```

## getMaterialAt(index)

### getMaterialAt(index)

Returns the material at an `index` from the database.

##### React usage example

```
import { getMaterialAt } from "../api/backend-api";

const elem21 = getMaterialAt(21);
```

## getExperimentAt(index)

### getExperimentAt(index)

Returns the experiment at an `index` from the database.

##### React usage example

```
import { getExperimentAt } from "../api/backend-api";

const elem21 = getExperimentAt(21);
```

## getDataPointArrayAt(index)

### **getDataPointArrayAt(index)**

Returns an array of `DataPoint` at an `index` from the database.

#### **React usage example**

```
import { getDataPointArrayAt } from "../api/backend-api";
import { DataPointType } from "types/DataPointTypes";

const dataPointArrya: DataPointType[] = getDataPointArrayAt(21);
```

## postMaterialJS(material)

### **postMaterialJS(material)**

Posts a new material to the Data base

#### **React usage example**

```
import { postMaterialJS } from "../api/backend-api";

postMaterialJS({
  //...
})
```

## patchMaterialByIdJS(patchMaterial)

### **patchMaterialByIdJS(patchMaterial)**

Patches an existing material in the Data base

#### **React usage example**

```
import { patchMaterialByIdJS } from "../api/backend-api";

patchMaterialByIdJS({
  id: 2,
  supplier_name: "Meu novo fornecedor",
  supplier_contact_info: "(12) 9 9123-0192",
  extra_info: "Hehe muito legal",
})
```

## deleteMaterialByIdJS(id)

### deleteMaterialByIdJS(id)

Deletes an existing material in the Data base.

#### React usage example

```
import { deleteMaterialByIdJS } from "./api/backend-api";

deleteMaterialByIdJS(22)
```

## postExperimentJS(experiment)

### postExperimentJS(experiment)

Posts a new experiment to the Data base

#### React usage example

```
import { postExperimentJS } from "./api/backend-api";

postExperimentJS({
  // ...
})
```

## patchExperimentByIdJS(patchExperiment)

### patchExperimentByIdJS(patchExperiment)

Patches an existing experiment in the Data base

#### React usage example

```
import { patchExperimentByIdJS } from "./api/backend-api";

patchExperimentByIdJS({
  id: 2,
  name: "Meu novo nome",
  extra_info: "Hehe muito legal",
})
```

## deleteExperimentByIdJS(id)

### deleteExperimentByIdJS(id)

Deletes an existing experiment in the Data base.

#### React usage example

```
import { deleteExperimentByIdJS } from "./api/backend-api";

deleteExperimentByIdJS(22)
```

## Core

## checkCanStartExperimentJS()

### checkCanStartExperimentJS()

This function calls the `check_can_start_experiment(experiment_id)` on the backend.

The front end will call this function when the user click to start experiment.

The backend **MUST** respond with a 1 if everything is ok or 0 if something is not correct.

In case something is wrong the backend also displays an error to the user telling what went wrong

#### React usage example

```
import { checkCanStartExperimentJS } from "./api/backend-api";

onClick(()=>{
  checkCanStartExperimentJS(2);
};)
```

## startExperimentRoutineJS(experimentId)

### startExperimentRoutineJS(experimentId)

This function calls the `start_experiment_routine(experiment_id)` on the backend.

The front end will call this function after everything is correct and ready to change pages.

Receives an `id` to an experiment as parameter.

The backend **MUST** send a command to change to the experiment page.

Returns 1 if succeeded.

#### React usage example

```
import { startExperimentRoutineJS } from "./api/backend-api";

onClick(()=>{
  startExperimentRoutineJS(2);
});
```

## endExperimentRoutineJS()

### endExperimentRoutineJS()

This function calls the `end_experiment_routine()` on the backend.

Usually it should be used to handle when the user press a "end experiment" button or something similar.

#### React usage example

```
import { getMaterialList } from "./api/backend-api";

onClick(()=>{
  endExperimentRoutineJS();
});
```

## setCustomMovementDistanceJS()

### setCustomMovementDistanceJS()

#### Warning

DEPRECATED

This function calls the `set_custom_movement_distance(new_movement_distance)` on the backend.

Sets the movement distance that the z-axis moves when the user is controlling the machine manually.

This distance is set in MILLIMETERS

Returns 1 if succeeded.

#### React usage example

```
import { setCustomMovementDistanceJS } from './api/backend-api';

onClick(()=>{
  // Sets the movement distance to 50 mm
  setCustomMovementDistanceJS(50);
};)
```

## returnZAxisJS()

### returnZAxisJS()

This function calls the `return_z_axis()` on the backend.

Returns the z-axis to the origin.

Returns 1 if succeeded (if the function was acknowledged).

#### React usage example

```
import { returnZAxisJS } from './api/backend-api';

onClick(()=>{
  returnZAxisJS();
};)
```



## stopZAxisJS()

### stopZAxisJS()

This function calls the `stop_z_axis()` on the backend. Stops the z-axis. Returns 1 if succeeded (if the function was acknowledged).

#### React usage example

```
import { stopZAxisJS } from "./api/backend-api";

onClick(()=>{
  stopZAxisJS();
};)
```

## moveZAxisMillimetersJS(distance)

### moveZAxisMillimetersJS(distance)

This function calls the `move_z_axis_millimeters(distance)` on the backend. Moves the z-axis [distance]mm. This distance is set in MILLIMETERS Returns 1 if succeeded (if the function was acknowledged).

#### React usage example

```
import { moveZAxisMillimetersJS } from "./api/backend-api";

onClick(()=>{
  moveZAxisMillimetersJS(10);
};)
```

## moveZAxisRevolutionsJS(revolutions)

### **moveZAxisRevolutionsJS(revolutions)**

This function calls the `move_z_axis_revolutions(revolutions)` on the backend. Moves the z-axis [revolutions].

This distance is set in revolutions

Returns 1 if succeeded (if the function was acknowledged).

#### **React usage example**

```
import { moveZAxisRevolutionsJS } from "./api/backend-api";

onClick(()=>{
  moveZAxisRevolutionsJS(0.5);
});
```

## getAvailablePortsListJS()

### **getAvailablePortsListJS()**

This function calls the `get_available_ports_list()` on the backend. Returns a JSON object containing the available COM ports:

#### **JSON**

```
{
  "port": x,
  "desc": y,
}
```

#### **React usage example**

```
import { getAvailablePortsListJS } from "./api/backend-api";

onClick(()=>{
  getAvailablePortsListJS().then((availablePorts)=>{
    if(availablePorts) console.log(availablePorts);
  });
});
```

## connectToPortJS()

### connectToPortJS()

This function calls the `connect_to_port()` on the backend. Connects to a port. The port argument is a string like `COM4`

Returns 1 connection was successful

#### React usage example

```
import { connectToPortJS } from "./api/backend-api";

onClick(()=>{
  connectToPortJS("COM3");
};)
```

## disconnectGranuladoJS()

!!! quote "### disconnectGranuladoJS() ()" This function calls the `disconnect_granulado()` on the backend.

#### Text Only

Returns 1 connection was successful

```
``` javascript title="React usage example"
import { disconnectGranuladoJS } from "./api/backend-api";

onClick(()=>{
  disconnectGranuladoJS("COM3");
};)
```
```

## tareLoadJS()

### tareLoadJS()

This function calls the `tare_load()` on the backend. Tares the load cell Returns 1 if succeeded (if the function was acknowledged).

#### React usage example

```
import { tareLoadJS } from "./api/backend-api";

onClick()=>{
  tareLoadJS();
};)
```

## calibrateKnownWeightJS()

### calibrateKnownWeightJS()

This function calls the `calibrate_known_weight()` on the backend. Calibrates the load cell to the known weight Returns 1 if succeeded (if the function was acknowledged).

#### React usage example

```
import { calibrateKnownWeightJS } from "./api/backend-api";

onClick()=>{
  calibrateKnownWeightJS();
};)
```

## calibrateZAxisJS()

### **calibrateZAxisJS()**

This function calls the `calibrate_z_axis()` on the backend. Calibrates z axis of the machine Returns 1 if succeeded (if the function was acknowledged).

#### **React usage example**

```
import { calibrateZAxisJS } from "./api/backend-api";

onClick()=>{
  calibrateZAxisJS();
};)
```

## getGranuladoIsConnectedJS()

### **getGranuladoIsConnectedJS()**

This function calls the `get_granulado_is_connected()` on the backend. Checks if granulado is connected

Returns a `boolean`

#### **React usage example**

```
import { getGranuladoIsConnectedJS } from "./api/backend-api";

onClick()=>{
  alert(getGranuladoIsConnectedJS());
};)
```