

# Bolinho

---

**Solution for data gathering, processing and interaction**

*Hefestus*

*Copyright © 2023 Hefestus.*

# Table of contents

---

|                    |    |
|--------------------|----|
| 1. Home            | 3  |
| 1.1 Home           | 3  |
| 1.2 Setup          | 4  |
| 2. Back End        | 6  |
| 2.1 Back End       | 6  |
| 2.2 DataHandler.py | 7  |
| 2.3 DBHandler      | 8  |
| 2.4 DataHandler    | 11 |
| 2.5 Routines       | 12 |
| 3. Front End       | 13 |
| 3.1 Front End      | 13 |
| 3.2 Pages          | 14 |
| 3.3 Components     | 17 |
| 3.4 Contexts       | 20 |
| 3.5 Alerts         | 21 |
| 3.6 Styling        | 22 |
| 4. Embedded        | 23 |
| 4.1 Embedded       | 23 |
| 5. API             | 24 |
| 5.1 API            | 24 |
| 5.2 Front end API  | 25 |
| 5.3 Backend API    | 36 |
| 5.4 Data types     | 42 |
| 6. About           | 49 |
| 6.1 About          | 49 |

# 1. Home

---

## 1.1 Home

---

Documentation of the FullStack solution **bolinho**.

This documentation automatically generates a **PDF file** from it's content. You can download it [here](#).

### 1.1.1 You can see the React [Front-end HERE!](#)

---

This is a static version of the app, without access to the server, therefore most features won't work.

#### Info

Remember that you need to build the app for it to show on the static page, so run `npm run buildWeb` or something similar to build it.

Use the **Tabs** above to navigate through the documentation.

---

### 1.1.2 Running

---

As for running the program we have a few options:

- Run only the frontend `npm run startWeb`
- Run only the backend `npm run startEel`
- Serve the full application `npm run serve`

This command will start the eel as headless and start the web serve, it doesn't need to build the front end before executing. **Less performant**.

To update the backend ability to call front end functions you should first build the front.

- Run the full application `npm run start`

With this command it will first build the react front end, then run the python script.

- Build the react frontend `npm run buildWeb`
- Build binaries. `npm run buildBin`
- You can build the "binaries", more like a python environment wrapper, it uses [PyInstaller](#) to generate the bins.
- The output path is `bolinho/src/dist/`

Did you like this documentation? You can check out the repo [ZRafaF/ReadTheDocksBase](#) for more info 😊.

## 1.2 Setup

---

This page will define the step-by-step to build this project.

This project assumes you have the latest version of [Python](#), **PIP** and **GIT**,

This project was developed using python version `Python 3.10.x`

### 1.2.1 Clone the repo

---

**Bash**

```
git clone https://github.com/HefestusTec/bolinho  
  
cd bolinho
```

### 1.2.2 Creating a virtual environment

---

The following step isn't mandatory but **recommended**.

**Bash**

```
python -m pip install --user virtualenv  
  
python -m venv venv
```

The a directory `venv` should be created in the root folder.

How to activate:

#### Windows activation

```
venv/Scripts/activate
```

OR

#### Linux activation

```
source venv/bin/activate
```

---

### 1.2.3 Installing dependencies

---

**Bash**

```
npm run installDep
```

---

## 1.2.4 Documentation

The following step is only required for those that want to **edit the documentation**.

### Installing dependencies

**Bash**

```
pip install -r docs/requirements.txt
```

### Build

We have two options to create a build:

- **Serve:**

This option is used for debugging, it will open the static page in one of the localhost ports.

```
mkdocs serve
```

- **Build:**

This option creates a build of the documentation and saves it on the directory `/site/`.

```
mkdocs build
```

#### Note

Be aware of the **Environment Variable** `ENABLE_PDF_EXPORT`, it will only generate the PDF if this variable is set to `1`.

You can change the `mkdocs.yml` file and remove this line if you so choose.

For more info about the documentation please checkout [ZRafaF/ReadTheDocksBase](#).

## 2. Back End

---

### 2.1 Back End

---

#### 2.1.1 DBHandler.py

---

Defines database models using orm\_sqlite

#### 2.1.2 Models

---

- Material
- Experiment
- Body
- Reading

#### 2.1.3 Classes

---

### DBHandler

Handles database connection and CRUD operations

#### METHODS

- **init**(self, db\_path)
- add\_material(self, name, batch)
- add\_experiment(self, name, material, date, time, load\_loss\_limit, max\_load, max\_travel, max\_time, compress, z\_axis\_speed)
- add\_reading(self, experiment, load, z\_pos, time)
- get\_materials(self)
- get\_material\_by\_id(self, id)
- get\_bodies(self)
- get\_body\_by\_id(self, id)
- get\_bodies\_by\_material(self, material)
- get\_bodies\_by\_type(self, body\_type)
- get\_bodies\_by\_material\_and\_type(self, material, body\_type)
- get\_experiments(self)
- get\_experiment\_by\_id(self, id)
- get\_experiments\_by\_material(self, material)
- get\_experiments\_by\_date(self, date)
- get\_experiments\_by\_date\_and\_material(self, date, material)
- get\_experiment\_readings(self, experiment)
- delete\_experiment\_by\_id(self, id)
- delete\_material\_by\_id(self, id)
- populate(self)

## 2.2

---

Handles Raspberry Pi GPIO and data acquisition

### 2.2.1 [Classes](#)

---

#### **DataHandler**

Abstract class for data acquisition

#### **LoadCell**

Handles load cell data acquisition

#### **StepMotor**

Handles step motor data acquisition

## 2.3 DBHandler

---

### 2.3.1 DBHandler.py

---

Defines database models using orm\_sqlite

### 2.3.2 Models

---

- Material
  - id : Integer Primary Key
  - name: String
  - batch: String
- Experiment
  - id : Integer Primary Key
  - name: String
  - material: Integer Foreign Key
  - date: Date
  - time: Time
  - load\_loss\_limit: Float
  - max\_load: Float
  - max\_travel: Float
  - max\_time: Float
  - compress: Boolean
  - z\_axis\_speed: Float
- Body
  - id : Integer Primary Key
  - body\_type: String
  - material: Integer Foreign Key
  - param\_a: Float
  - param\_b: Float
  - height: Float
- Reading
  - id: Integer Primary Key
  - experiment: Integer Foreign Key
  - load: Float
  - z\_pos: Float
  - time: Float



## 2.3.3 Classes

### DBHandler

Handles database connection and CRUD operations

#### METHODS

- **init**(self, db\_path)
  - Creates database connection, binds models to database, creates and populates the database if it doesn't exist
- db\_path: Path to database file
- add\_material(self, name, batch)
  - name: Name of the material
  - batch: Batch number of the material
- add\_experiment(self, name, material, date, time, load\_loss\_limit, max\_load, max\_travel, max\_time, compress, z\_axis\_speed)
  - name: Name of the experiment
  - material: Name of the material
  - date: Date of the experiment
  - time: Time of the experiment
  - load\_loss\_limit: Load loss limit of the experiment
  - max\_load: Maximum load of the experiment
  - max\_travel: Maximum travel of the experiment
  - max\_time: Maximum time of the experiment
  - compress: Whether the experiment is compressive or not
  - z\_axis\_speed: Speed of the z axis
- add\_reading(self, experiment, load, z\_pos, time)
  - experiment: Name of the experiment
  - load: Load of the reading
  - z\_pos: Z position of the reading
  - time: Time of the reading
- get\_materials(self)
  - Returns all materials in the database
- get\_material\_by\_id(self, id)
  - id: Id of the material
  - Returns the material with the given id
- get\_bodies(self)
  - Returns all bodies in the database
- get\_body\_by\_id(self, id)
  - id: Id of the body
  - Returns the body with the given id

• `get_bodies_by_material(self, material)`

- material: Name of the material
- Returns all bodies with the given material

• `get_bodies_by_type(self, body_type)`

- body\_type: Type of the body
- Returns all bodies with the given type

• `get_bodies_by_material_and_type(self, material, body_type)`

- material: Name of the material
- body\_type: Type of the body
- Returns all bodies with the given material and type

• `get_experiments(self)`

- Returns all experiments in the database ordered by date and time descending

• `get_experiment_by_id(self, id)`

- id: Id of the experiment
- Returns the experiment with the given id

• `get_experiments_by_material(self, material)`

- material: Name of the material
- Returns all experiments with the given material

• `get_experiments_by_date(self, date)`

- date: Date of the experiment
- Returns all experiments with the given date

• `get_experiments_by_date_and_material(self, date, material)`

- date: Date of the experiment
- material: Name of the material
- Returns all experiments with the given date and material

• `get_experiment_readings(self, experiment)`

- experiment: Name of the experiment
- Returns all readings of the given experiment

• `delete_experiment_by_id(self, id)`

- id: Id of the experiment
- Deletes the experiment with the given id and all its readings

• `delete_material_by_id(self, id)`

- id: Id of the material
- Deletes the material with the given id and all its experiments and readings

• `populate(self)`

- Populates the database with some dummy data

## 2.4 DataHandler

---

### 2.4.1 [DataHandler.py](#)

---

Handles Raspberry Pi GPIO and data acquisition

### 2.4.2 Classes

---

#### **DataHandler**

Abstract class for data acquisition

#### **LoadCell**

[NOT IMPLEMENTED]

Handles load cell data acquisition

#### **StepMotor**

[NOT IMPLEMENTED]

Handles step motor data acquisition

## 2.5 Routines

---

This page is a summary of the routines of the machine.

## 3. Front End

---

### 3.1 Front End

---

As front end framework/library we are using [ReactJs](#).

You can check out the [about](#) page for a list of the **third party projects** being used.

### Global variables

The global variables can be found at `variables.css`. You can import it into your stylesheet using

CSS

```
@import url(./variables.css);
```

#### 3.1.1 Component architecture

---


flowchart TD

```
App --> MainPage
App --> ExperimentPage
MainPage --> SideBar
SideBar --> Header
MainPage --> Content
Content --> Inicio
Content --> Calibrar
Content --> Controlar
Content --> Config.
Content --> Sobre
```

### 3.2 Pages

The following pages are just place holders / mockups of the final application.

#### 3.2.1 Initial page

**Bolinho**  
Copyright © 2023 Hefestus


Início

Calibrar

Controlar

Config.

Sobre



Selecionar Material

Buscar

[592] Aço carbono 12

+ [593] Alumínio

+ [594] Fibra de vidro

+ [595] Alumínio 2020

+ [596] Mat 1

+ [597] Mat 2

+ [598] Mat 3

[592] Aço carbono 12

[592] Aço carbono 12

[595] Alumínio 2020

Material: Aço carbono

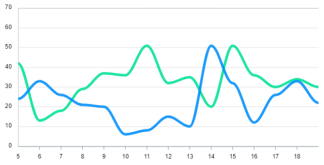
Lote: 1202

Fornecedor: MinasLTDA

Ensaio: 02

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam malesuada placerat fringilla. Ut viverra, nulla vitae egestas.

Dynamic Updating Chart



Opções

Força X Posição

Tensão X Deformação

Redefinir Zoom

Extra

Gerar relatório

Ensaio

### Main

Initial page of the application, you are able to:

- Visualize previous experiments;
- Exclude experiments;
- Exclude materials;
- Access different sub-pages;
- Generate report; and more!.



Calibration sub-page



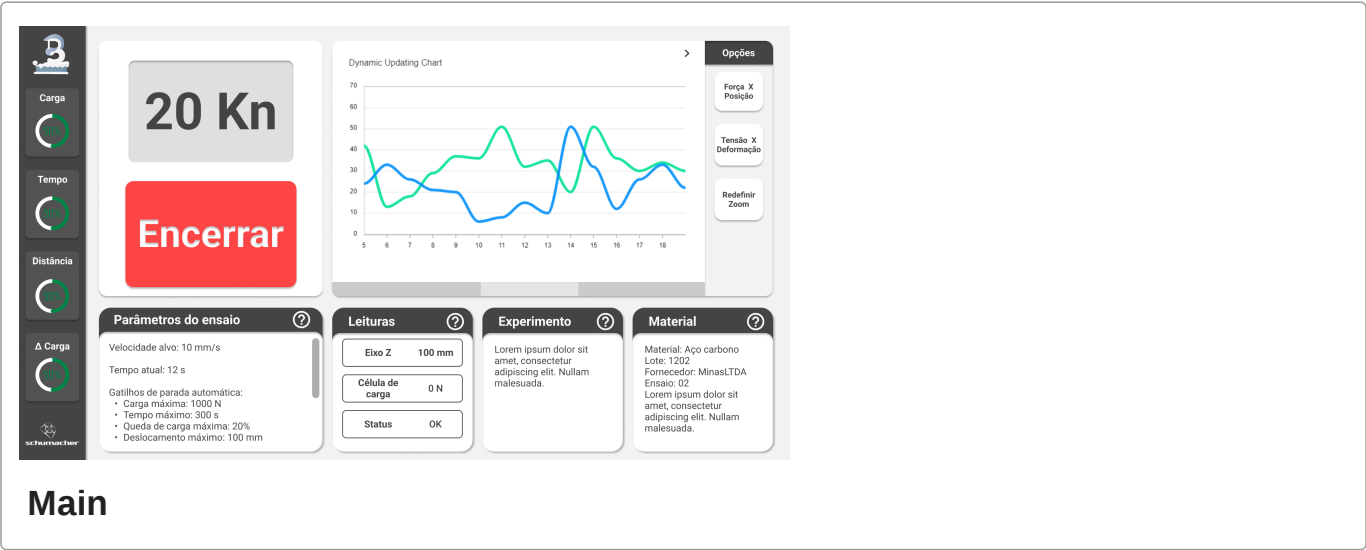
Control sub-page



Configuration sub-page



3.2.2 Experiment page





## 3.3 Components

Most custom React components are here

### 3.3.1 ZoomComponent

Component that enables zooming on it's contents, press and hold to zoom in.

It uses the [use-long-press](#) hook to handle this action.

You can enable/disable and change the activation time on the [GlobalConfigContext](#) context.

#### Props

##### React

```
{
  scaleOrigin = "top",
  className = "",
  children,
}
```

- **scaleOrigin** : What is the origin point (which direction does it grows).
- **className** : ClassName of the component.
- **children** : Children props, Don't worry about it.

#### Usage example

##### React

```
import ZoomComponent from "../zoomComponent/zoomComponent";
import ExperimentsInspector from "../experimentsInspector/experimentsInspector";

export default function exampleComponent(){
  return(
    <ZoomComponent
      className={styleModule.experiments_inspector}
      scaleOrigin="bottom left"
    >
      <ExperimentsInspector />
    </ZoomComponent>
  );
}
```

## 3.3.2 ExperimentInspector

This component makes use of the `SelectedObjectContext`.

The experiment inspector component holds the following children:

- `ColorPicker`: Color picker component for choosing the color of the active experiment.
- `ExperimentButton`: Buttons that make the list of selected experiments.
- `ExperimentDescription`: Component that parses the description of the active material.

### Props

none

### Usage example

#### React

```
import { eel } from "../App";
import ExperimentsInspector from "../experimentsInspector/experimentsInspector";

export default function exampleComponent(){

  const getMaterialList = async () => {
    try {
      const materialList = JSON.parse(await eel.get_material_list());
      return materialList;
    } catch (error) {
      return [];
    }
  };

  return(
    <SelectedObjectContext.Provider>
      <ExperimentsInspector />
    </SelectedObjectContext.Provider>
  );
}
```

### 3.3.3 MaterialSelector

The `MaterialSelector` component holds the following children:

- `MaterialSelectorButton` : Buttons that make the list of available experiments.
- `DropDownButton` : Buttons that make a dropdown, so the user can choose which experiment they want to inspect.

It can be wrapped with the `ZoomComponent` to allow zooming.

## Props

### React

```
{
  materialList // List of available materials, fetched from the backend
}
```

## Usage example

### React

```
import { eel } from "../App";
import MaterialSelector from "../materialSelector/materialSelector";

export default function exampleComponent(){

  const getMaterialList = async () => {
    try {
      const materialList = JSON.parse(await eel.get_material_list());
      return materialList;
    } catch (error) {
      return [];
    }
  };

  return(
    <MaterialSelector materialList={getMaterialList} />
  );
}
```

## 3.4 Contexts

---

This page contains all the [contexts](#) used on the application.

### 3.4.1 GlobalConfigContext

---

**React**

```
{
  theme: "light", // light | dark
  animationSpeed: "slow", // fast | slow | off
  animateGraph: "on", // on | off
  enableZoom: true, // Should zooming be enable?
  zoomDelay: 300, // How long [ms] should I press to zoom
  blurOnZoom: true, // Should it blur when zooming?
  absoluteMaximumForce: 10000,
}
```

---

### 3.4.2 SelectedObjectsContext

---

This context is accessible to the children of the `MainPage` component. It holds a list of the selected objects "experiments data".

**React**

```
{
  material, // material_fragment
  experiment, // experiment_fragment
  data_array, // data_array_fragment,
  color, // color associated to an experiment
}
```

## 3.5 Alerts

---

This project is using [react-toastify](#) to handle alerts. If you want more in-depth info about them please refer to the [documentation](#).

### 3.5.1 Usage

---

Here is an exemple of how you can create an alert.

#### React

```
import React from "react";
import { toast } from "react-toastify";

export default function myComponent() {
  const iWasPressed = () => {
    // This will show an error alert.
    toast.error("Não foi possível acessar o backend");
  };

  return(
    <button onClick={iWasPressed}>
      Press me :D
    </button>
  );
}
```

## 3.6 Styling

---

Here we will show some tips and styling techniques used on this project.

### 3.6.1 Linking stylesheets to a component

---

Stylesheets are made with `style.module.css` and imported into the `.jsx` or `.tsx` allowing us to use it as an object to name `classNames`.

As for naming convention snake case is being used for styling names

**React**

```
import styleModule from "../mainPage.module.css";

<div className={styleModule.my_custom_div}>
  Im a styled div
</div>
```

### 3.6.2 Adding ellipsis

---

You can add ellipsis to most texts using the following code

**CSS**

```
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

#### Tip

This **must not be wrapped in a flex div**, therefore we recommend you using a div only for the text, for example:

**React**

```
<div className="my-cool-div">
  <div className="my-text">
    This text will add ellipsis when it overflows.
  </div>
</div>
```

**CSS**

```
.my-text{
  white-space: nowrap;
  overflow: hidden;
  text-overflow: ellipsis;
}
```

## 4. Embedded

---

### 4.1 Embedded

---

Bolinho uses a microcontroller [esp32-s3](#) for controlling the hardware.

The microcontroller communicates via serial to the host, and is responsible for reading the load cell and controlling the stepper motor.

## 5. API

---

### 5.1 API

---

In this section you will be able to find every **API call** available.

These **calls** are exposed to the **front-end** via the **eel** object, giving it access to the **data base**, **systems** and **hardware**. This solution makes use of the **eel** library to realize the communication between the front-end and back-end;

This API reference will show the methods being called by the front-end in JavaScript, and every call should be made **asynchronously**.

#### 5.1.1 How to create and expose functions to the backend

---

##### React

```
function myJsFunction(message){
  console.log(`Got this from the back end ${message}`)
}

// This line exposes the function to the back end, note the second argument, it is the name
// that the back end needs to call
window.eel.expose(myJsFunction, "myJsFunction");
```

##### Python

```
try:
    eel.myJsFunction("IT'S WORKING")
except:
    pass
```



## 5.2 Front end API

---

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

Backend -> Front end

### Warning

The functions can only be called if they are available on the `web/build` directory, therefore if you make a change using `npm run serve` won't show it, you will need to rebuild the front end with `npm run buildWeb` or by using `npm run start`.

### Note

These functions can only be called after eel is initiated with `eel.init()`.

### 5.2.1 Core API

---

Collection of all functions/API calls available to the backend. You can find them in the `bolinho_api/core.py` file.

The JavaScript file can be found in the `api` folder.

---

## ping()

### ping()

Tries to ping the bolinho front-end, returns 1 if it worked

#### Python usage example

```
from bolinho_api.core import core_api

while True:
    try:
        if core_api.ping():
            print("got a ping!")
            break
        pass
    except:
        eel.sleep(1)
```

---

## get\_config\_params()

### get\_config\_params()

Tries to ping the bolinho front-end, returns 1 if it worked

#### Python usage example

```
from bolinho_api.core import core_api

config = core_api.get_config_params()
current_save_version = config["configVersion"]
print(current_save_version)
```

This function is located at `src/web/src/App.js`

## go\_to\_experiment\_page()

### go\_to\_experiment\_page()

Asks the front end to go to the experiment page.

Returns 1 if succeeded.

#### Python usage example

```
from bolinho_api.core import core_api

change_pages = True
if change_pages:
    core_api.go_to_experiment_page()
```

## go\_to\_home\_page()

### go\_to\_home\_page()

Asks the front end to go to the home page.

Returns 1 if succeeded.

#### Python usage example

```
from bolinho_api.core import core_api

change_pages = True
if change_pages:
    core_api.go_to_home_page()
```

## show\_connect\_prompt()

### show\_connect\_prompt()

Asks the front end to show the connection prompt.

The connection prompt is used to select the serial port.

Returns 1 if succeeded.

#### Python usage example

```
from bolinho_api.core import core_api

config = core_api.get_config_params()
device_port = config["port"]

while not device_port:
    core_api.show_connect_prompt()
    device_port = config["port"]
```

---

## 5.2.2 UI API

Collection of all functions/API calls available to the backend for UI in general. You can find them in the `bolinho_api/ui.py` file.

The JavaScript file can be found in the `api` folder.

---

## success\_alert(text)

### success\_alert(text)

Uses [React-Toastify](#) to create an success alert.

#### Python usage example

```
from bolinho_api.ui import ui_api

ui_api.success_alert("Success!")
```

---

## error\_alert(text)

### error\_alert(text)

Uses [React-Toastify](#) to create an error alert.

#### Python usage example

```
from bolinho_api.ui import ui_api

ui_api.error_alert("Error!")
```

## prompt\_user(description, options, callback\_func)

### prompt\_user(description, options, callback\_func)

Prompts the user with a 'description', and shows the 'options' to the user.

The result is passed to the callback\_function

#### Python usage example

```
from bolinho_api.ui import ui_api

def get_result(result):
    if result == "yes":
        print("The user chose yes")
    print("The user chose no")

ui_api.prompt_user(
    description="Do you want to pay 1000?",
    options=["yes", "no"],
    callback_func= get_result,
)
```

## set\_focus(focus\_element: str)

### error\_alert(focus\_element: str)

Focus in an specific element on the frontend.

WARNING Pay attention to the name of the element you are trying to focus

You can find them at <https://github.com/HefestusTec/bolinho/blob/main/src/web/src/api/apiTypes.ts>

#### Python usage example

```
from bolinho_api.ui import ui_api

ui_api.set_focus("connection-component")
```

## 5.2.3 Experiment page API

Collection of all functions/API calls available to the backend for the **experiment** routine. You can find them in the `bolinho_api/experiment.py` file.

The JavaScript file can be found at `web/src/api/contexts/ExperimentPageContext.tsx`.

## get\_load\_percentage()

### get\_load\_percentage()

Asks the front for the current load percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

Returns the load percentage value

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_load_percentage())
```

## set\_load\_percentage(newValue)

### set\_load\_percentage(newValue)

Sets the current load percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

for number in range(100):
    experiment_api.set_load_percentage(number)
    eel.sleep(0.1)
```

## get\_time\_percentage()

### get\_time\_percentage()

Asks the front for the current time percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

Returns the load percentage value

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_time_percentage())
```

## set\_time\_percentage(newValue)

### set\_time\_percentage(newValue)

Sets the current time percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_time_percentage(22)
```

## get\_distance\_percentage()

### get\_distance\_percentage()

Asks the front for the current distance percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

Returns the load percentage value

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_distance_percentage())
```

## set\_distance\_percentage(newValue)

### set\_distance\_percentage(newValue)

Sets the current distance percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_distance_percentage(22)
```

## get\_delta\_load\_percentage()

### get\_delta\_load\_percentage()

Asks the front for the current delta load percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

Returns the load percentage value

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_delta_load_percentage())
```

## set\_delta\_load\_percentage(newValue)

### set\_delta\_load\_percentage(newValue)

Sets the current delta load percentage.

This variable is shown to the user in a progress bar. And is usually between 0-100.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_delta_load_percentage(22)
```

## get\_experiment\_parameters()

### get\_experiment\_parameters()

Asks the front for the current experiment parameters.

Returns a formatted string

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_experiment_parameters())
```

## set\_experiment\_parameters(newValue)

### set\_experiment\_parameters(newValue)

Sets the current experiment parameters.

Receives a formatted string.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_experiment_parameters("Experiment 202 <br/> Load cell: 1xi92")
```



## get\_readings()

### get\_readings()

Asks the front for the current Readings.

Returns an object of type Readings, this object gathers all the current readings of the machine. Such as Current z axis position, current load, and status

#### Python usage example

```
from bolinho_api.experiment import experiment_api

reading_obj = experiment_api.get_readings()

print(reading_obj.status)
```

## set\_readings(newValue)

### set\_readings(newValue)

Sets the current Readings.

Receives an object of type Readings, this object gathers all the current readings of the machine. Such as Current z axis position, current load, and status.

This function dumps the object to a JSON and sends it to the front end

#### Python usage example

```
from bolinho_api.experiment import experiment_api
from bolinho_api.classes import Readings

new_machine_readings = Readings(299, 87, 300, "not good")

experiment_api.set_readings(new_machine_readings)
```

## get\_description()

### get\_description()

Asks the front for the current description.

Returns a formatted string

#### Python usage example

```
from bolinho_api.experiment import experiment_api

print(experiment_api.get_description())
```

## set\_description(newValue)

### set\_description(newValue)

Sets the current description.

Receives a formatted string.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_description("New Experiment description")
```

## get\_material()

### get\_material()

Asks the front for the current Material.

Returns an object of type Material.

#### Python usage example

```
from bolinho_api.experiment import experiment_api

material_obj = experiment_api.get_material()

print(material_obj.name)
```

## set\_material(newValue)

### set\_material(newValue)

Sets the current Material.

Receives an object of type Material

This function dumps the object to a JSON and sends it to the front end

#### Python usage example

```
from bolinho_api.experiment import experiment_api
from bolinho_api.classes import Material

current_material = Material(
    id=23,
    name="aço 22",
    batch="1",
    experimentArray=[1, 3, 2],
    supplier="Metalúrgica JOSÉ",
    extraInfo="Cilindro",
)

experiment_api.set_material(current_material)
```

## 5.3 Backend API

---

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

Front end -> Backend

---

### 5.3.1 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();
```

### 5.3.2 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);
```

## 5.3.3 Core

## startExperimentRoutineJS()

### startExperimentRoutineJS()

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

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

#### React usage example

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

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

## 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()

### moveZAxisMillimetersJS()

This function calls the `move_z_axis_millimeters()` 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);
});
```



## 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");
});
```

## 5.4 Data types

All different data types will be shown in this page

### ATTENTION

To see a more up to date version of the different data types please see [src/bolinho\\_api/classes.py](#) !

### 5.4.1 DataPoint

#### Python

```
class DataPoint:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
```

- `x` : Position at the measure moment
- type: `float`
- Unity: `mm`
- `y` : Force at the measure moment
- Type: `float`
- Unity: `N`

### 5.4.2 DataPointArray

#### Python

```
class DataPointArray:
    def __init__(self, id=0, data_array=[]):
        self.id = id
        self.data_array = data_array
```

- `id` : Identification (or key) of this element on the data base.
- type: `int`
- Unity: `mm`
- `data_array` : Array of data points, this is the "reading" of an experiment.
- Type: `[DataPoint...]`
- Unity: `N/A`

## 5.4.3 AutoStopParams

### Python

```
class AutoStopParams:
    def __init__(self, force_loss=20, max_force=1000, max_travel=100, max_time=600):
        self.force_loss = force_loss
        self.max_force = max_force
        self.max_travel = max_travel
        self.max_time = max_time
```

- **force\_loss** : Max force loss to trigger auto-stop.
  - Type: **float**
  - Unity: %
- **max\_force** : Max force limit to trigger auto-stop.
  - Type: **float**
  - Unity: N
- **max\_travel** : Max distance the experiment head can travel during the experiment.
  - Type: **float**
  - Unity: mm
- **max\_time** : Experiment time limit.
  - Type: **float**
  - Unity: s

## 5.4.4 BodyParams

### Python

```
class BodyParams:
    def __init__(self, type=0, param_a=0, param_b=0, height=0):
        # Body format | 1 = Rectangle | 2 = Cylinder | 3 = Tube
        self.type = type

        # Rectangle = length | Cylinder = External diameter | Tube = External diameter
        self.param_a = param_a

        # Rectangle = depth | Cylinder = NULL | Tube = Internal diameter
        self.param_b = param_b

        # Height of the test body
        self.height = height
```

- **type** : Body format
- 1 = Rectangle
- 2 = Cylinder
- 3 = Tube
- Type: **int**
- Unity: N/A
- **param\_a** : Param 'a' of the body
- Rectangle = length
- Cylinder = External diameter
- Tube = External diameter
- Type: **float**
- Unity: **mm**
- **param\_b** : Param 'b' of the body
- Rectangle = depth
- Cylinder = NULL
- Tube = Internal diameter
- Type: **float**
- Unity: **mm**
- **height** : Height of the test body
- Type: **float**
- Unity: **mm**

## 5.4.5 ExperimentParams

**Python**

```
class ExperimentParams:
    def __init__(
        self,
        stop_params=AutoStopParams(),
        body_params=BodyParams(),
        compress=True,
        z_speed=5,
    ):
        self.stop_params = stop_params
        self.body_params = body_params
        self.compress = compress
        self.z_speed = z_speed
```

- **stop\_params** : Auto stop parameters of the experiment.
- Type: `AutoStopParams`
- Unity: N/A
- **body\_params** : Body parameters of the experiment.
- Type: `BodyParams`
- Unity: N/A
- **compress** : Dictates if the experiment head move up or down. true = compress | false = expand.
- Type: `bool`
- Unity: N/A
- **z\_speed** : Z axis speed during the experiment.
- Type: `float`
- Unity: `mm/s`

## 5.4.6 Date

**Python**

```
class Date:
    def __init__(
        self,
        day=1,
        month=1,
        year=2023,
    ):
        self.day = day
        self.month = month
        self.year = year
```

- **day** : Day.
- Type: `int`
- Unity: N/A
- **month** : Month.
- Type: `int`
- Unity: N/A
- **year** : Year.
- Type: `int`
- Unity: N/A

## 5.4.7 Experiment

### Python

```
class Experiment:
    def __init__(
        self,
        id=0,
        date=Date(),
        experiment_params=ExperimentParams(),
        data_array_id=0,
        extra_info="",
    ):
        self.experiment_params = experiment_params
        self.id = id
        self.data_array_id = data_array_id
        self.extra_info = extra_info
```

- **id** : id (or key) of the experiment on the data base.
- Type: **int**
- Unity: N/A
- **date** : Date of the experiment.
- Type: **Date**
- Unity: N/A
- **experiment\_params** : Parameters of the experiment.
- Type: **ExperimentParams**
- Unity: N/A
- **data\_array\_id** : Identification (or key) of this experiment **DataPointArray** or "reading", on the data base
- Type: **int**
- Unity: N/A
- **extra\_info** : Extra information about the experiment.
- Type: **String**
- Unity: N/A

## 5.4.8 Supplier

### Python

```
class Supplier:
    def __init__(self, name="NONE", email=""):
        self.name = name
        self.email = email
```

- **name** : Name of the material supplier.
- Type: **String**
- Unity: N/A
- **email** : E-mail of the supplier.
- Type: **String**
- Unity: N/A

---

## 5.4.9 Material

---

### Python

```
class Material:
    def __init__(
        self, id=0, name="NONE", batch=0, experiment_array=[], supplier=Supplier(), extra_info=""
    ):
        self.id = id
        self.name = name
        self.batch = batch
        # array of the ids of experiments with this material
        self.experiment_array = experiment_array
        self.supplier = supplier
        self.extra_info = extra_info
```

- **id** : Id (or key) of the material on the data base.
- Type: `int`
- Unity: N/A
- **name** : Name of the material.
- Type: `String`
- Unity: N/A
- **batch** : Batch of the material.
- Type: `int`
- Unity: N/A
- **experiment\_array** : Array with the `ids` or `keys` of the `experiments` made with this material.
- Type: `[int...]`
- Unity: N/A
- **supplier** : Supplier of the material.
- Type: `Supplier`
- Unity: N/A
- **extra\_info** : Extra information about the material.
- Type: `String`
- Unity: N/A



## 6. About

---

### 6.1 About

---

This page will present extra info about the project.

#### 6.1.1 Licenses

---

This software is licensed and distributed under the **GNU General Public License v3.0**

```
Copyright (C) 2023 Hefestus
```

```
Bolinho is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.
```

```
Bolinho is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License  
along with Bolinho. If not, see <http://www.gnu.org/licenses/>.
```

### Included third-party projects

- Python Eel - [see license](#)
- MkDocs - [see license](#)
- Material for MkDocs - [see license](#)
- MkDocs With PDF - [see license](#)
- MkDocs PDF Export Plugin - [see license](#)
- JSX Lexer - [see license](#)
- Roboto family of fonts - [see license](#)
- React - [see license](#)
- rc-slider - [see license](#)
- Chart JS - [see license](#)
- React Chart JS 2 - [see license](#)
- React Long Press Hook - [see license](#)
- React Colorful - [see license](#)
- React Toastify - [see license](#)
- Use Debounce - [see license](#)

- Use Long Press - [see license](#)
  - React Simple Keyboard - [see license](#)
  - React Circular Progressbar - [see license](#)
- 

Agradecemos do fundo do coração todos os autores dos diferentes projetos utilizados, **software livre** é liberdade, muito obrigado a todos.