Bolinho

Solution for data gathering, processing and interaction

Hefestus

Table of contents

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 Known bugs

The full list of known bugs can be seen at https://github.com/HefestusTec/bolinho/issues?q=is%3Aopen+is%3Aissue+label%3Abug

1.1.3 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 de 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. Manual do usuário

2.1 Manual do usuário

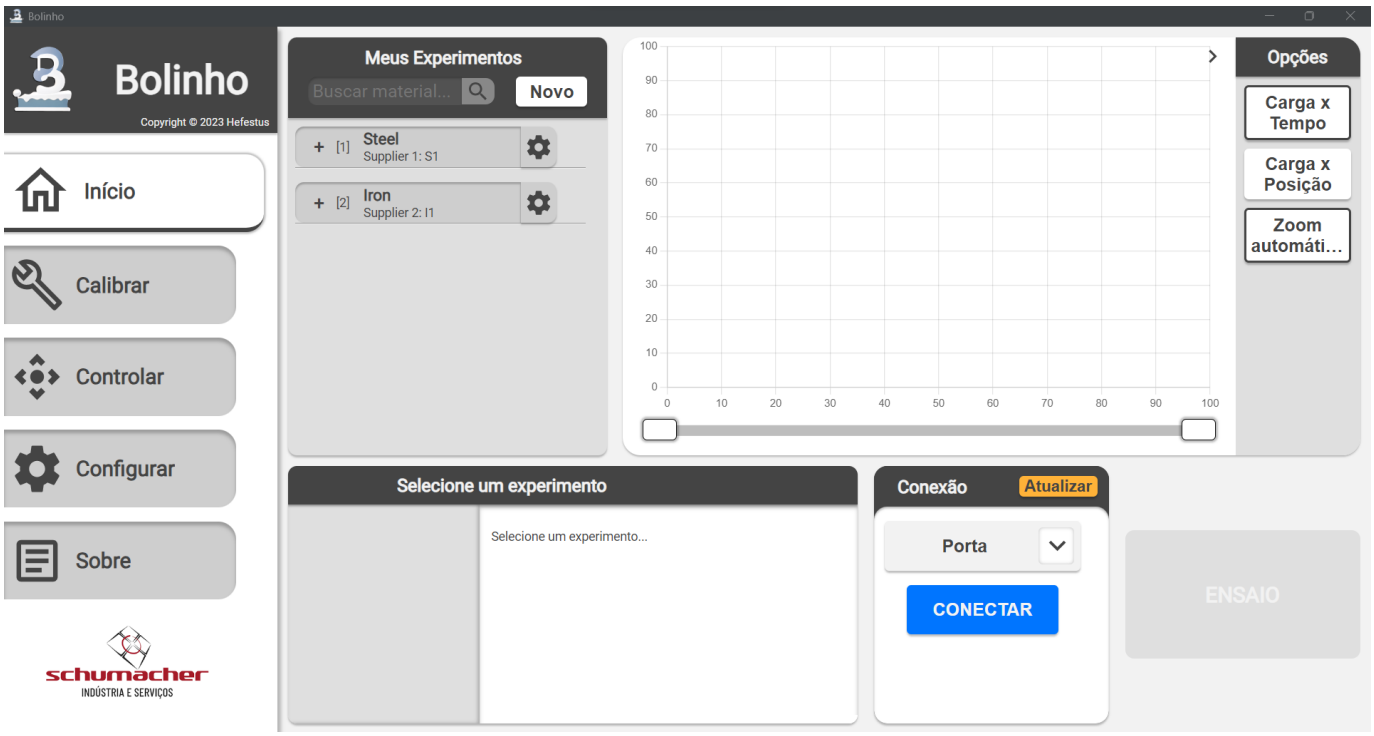
Aqui você encontrará o manual do usuário de todo o sistema do **Bolinho**.

2.1.1 Conhecendo os componentes

O sistema Bolinho é composto por dois componentes diferentes, o **Bolinho** e o **Granulado**

Bolinho

Bolinho é a **interface humana** responsável por orquestrar todo o funcionamento do sistema Bolinho.



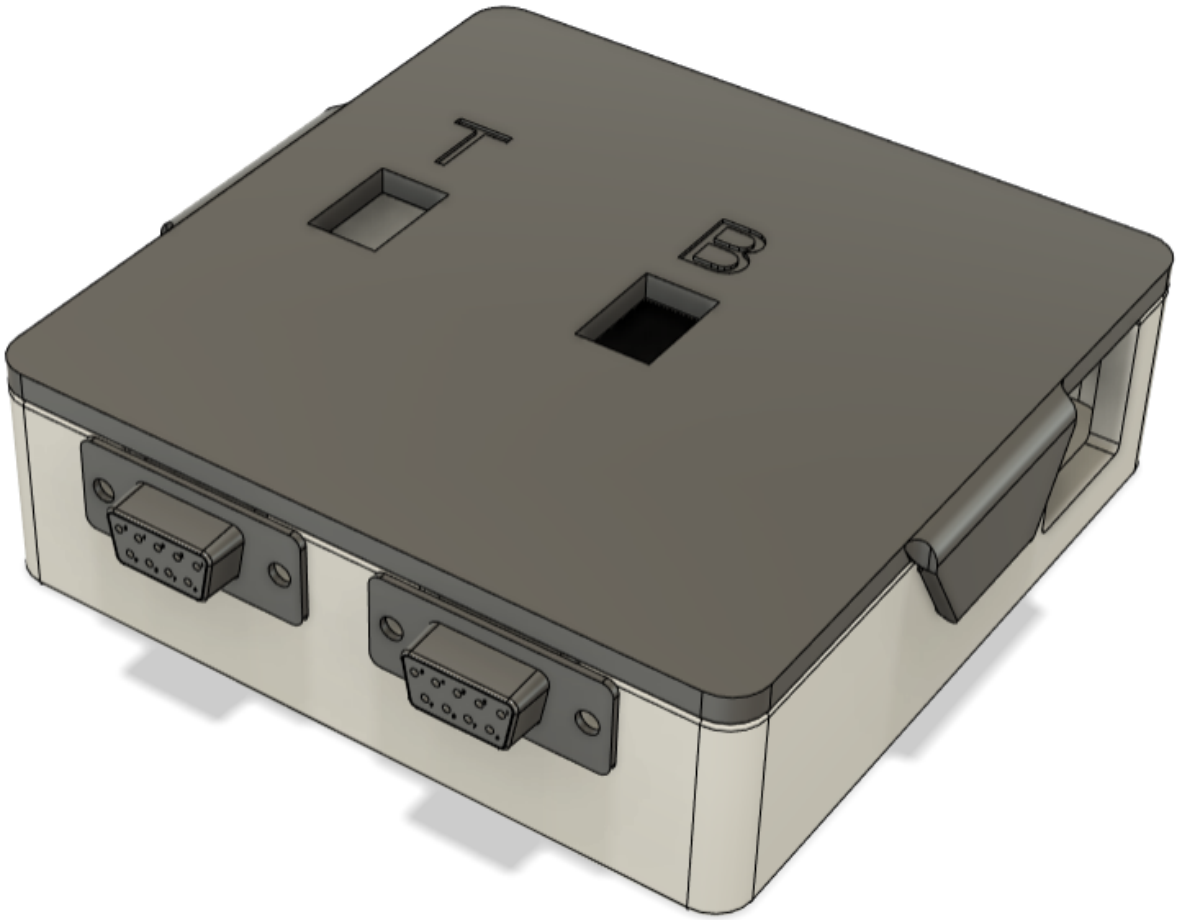
Bolinho é uma aplicação padrão de computador, e atualmente suporta apenas Linux como seu sistema operacional.

> **Info**
>
> Bolinho foi testado na distribuição Debian da OrangePI no hardware OrangePI 5 8Gb RAM

Para informações sobre a sua configuração visite a página Manual do usuário - configuração.

O repositório com o código fonte do bolinho pode ser encontrado em https://github.com/HefestusTec/bolinho junto com sua documentação.

Granulado



Granulado é o *firmware* embarcado do sistema, este é responsável pela interface com o *hardware* e seu *driver*.

O *firmware* foi escrito para atuar no ESP32-S3 e utiliza um USB para sua comunicação serial com o Bolinho. Mais informações especificas sobre o embarcado podem ser encontradas em embedded.

Para informações sobre a sua configuração visite a página Manual do usuário - configuração.

O repositório com o código fonte do bolinho pode ser encontrado em https://github.com/HefestusTec/granulado.

2.1.2 Sobre a documentação

A documentação é uma coleção completa de todas as informações pertinentes ao sistema Bolinho. Ela está escrita predominantemente em **Inglês** e possui o tópico **Manual do Usuário** escrito em **Português**.

Toda a documentação do Bolinho está disponível em dois formatos `Página estática` e `PDF`

> **Atenção**
>
> A Hefestus não se compromete em manter os arquivos relacionados a documentação acessiveis, caso deseje mantê-los você pode fazer uma cópia local de toda a documentação

Página estática

O que são? Paginas estáticas são páginas que podem ser abertas diretamente sem a necessidade de um servidor para servi-las, entretanto caso você queira também pode servi-la VOCÊ MESMO! Isso garante que todo o conteúdo será somente acessível a você / empresa.

> **Atenção**
>
> Algumas ferramentas como por exemplo a busca, apenas estão disponíveis se a página estiver sendo servida.

Quais os benefícios? Páginas estáticas permitem diferentes funcionalidades, não ficando restringido a um formato de folha de papel, e por se tratar de uma página minimamente responsiva permite que todo o trabalho de renderizar os conteúdos sejam feitos no momento de compilação do código.

Onde encontro os arquivos? Os arquivos relacionados a página estática podem ser encontrados na *branch* gh-pages do repositório do Bolinho.

Como rodar? Para ver os conteúdos de uma página estática basta abrir o arquivo `index.html` encontrado na *root* do diretório com seu navegador de preferência! Ex.: Google Chrome, Microsoft Edge, Firefox etc.

PDF

Esta documentação também gera automaticamente uma versão em PDF, o qual você poderá usar para compatibilidade retroativa com sua documentação já existente!

O PDF pode ser encontrado aqui

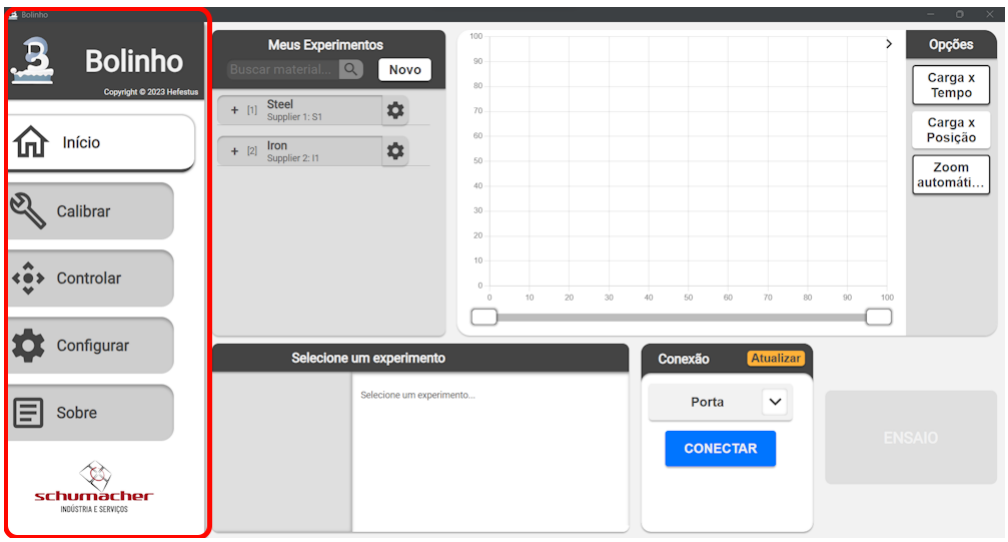
> **Atenção**
>
> Talvez esse link não esteja mais disponível por isso mantenha uma cópia do PDF localmente.

2.2 Inspecionando

Bem vindo a seção `Inspecionando` da documentação do Bolinho. Aqui você encontrará uma introdução à interface do Bolinho assim como as informações necessárias para inspecionar um experimento já realizado.

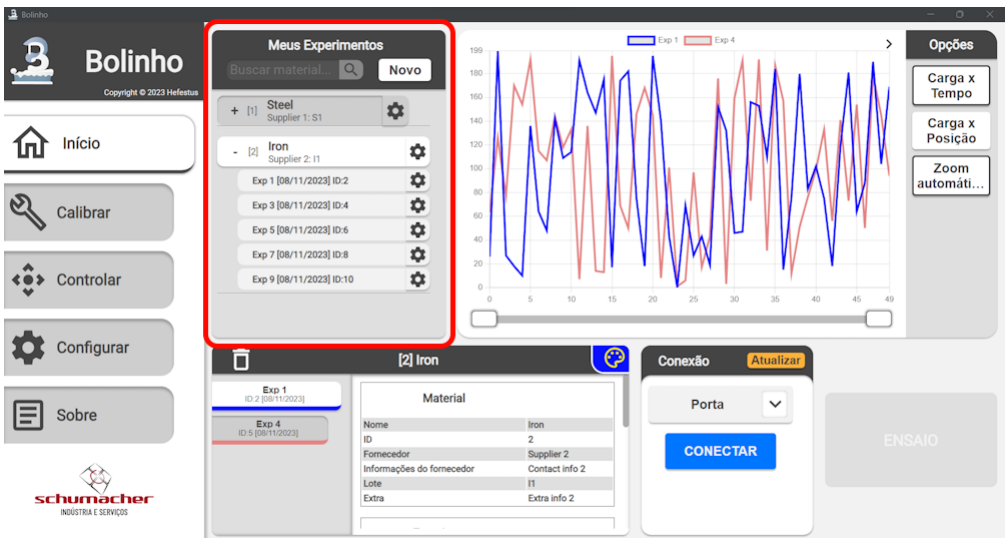
2.2.1 Conhecendo os componentes

Menu



O menu permite que você navegue pelas diferentes páginas do aplicativo.

Seletor de experimentos

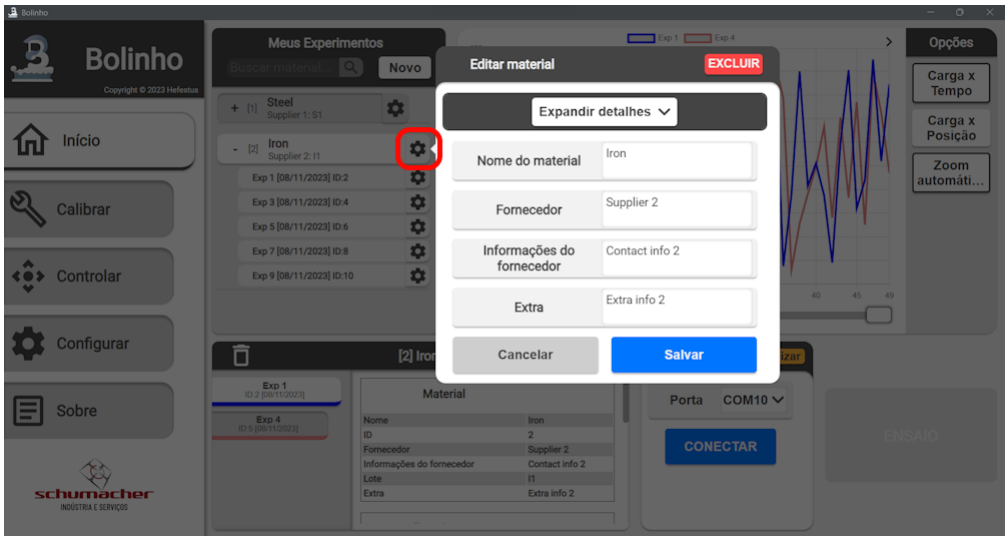


No seletor de experimentos os experimentos são organizados por seus materiais.

Você pode utilizar a **Barra de pesquisa** para filtrar os diferentes **materiais**.

Ao expandir um **material** todos os experimentos relacionados a ele serão apresentados.

Você tambem pode abrir o **dialogo de configuração** de seus **materiais** e **experimentos** para editar e revisar suas informações.



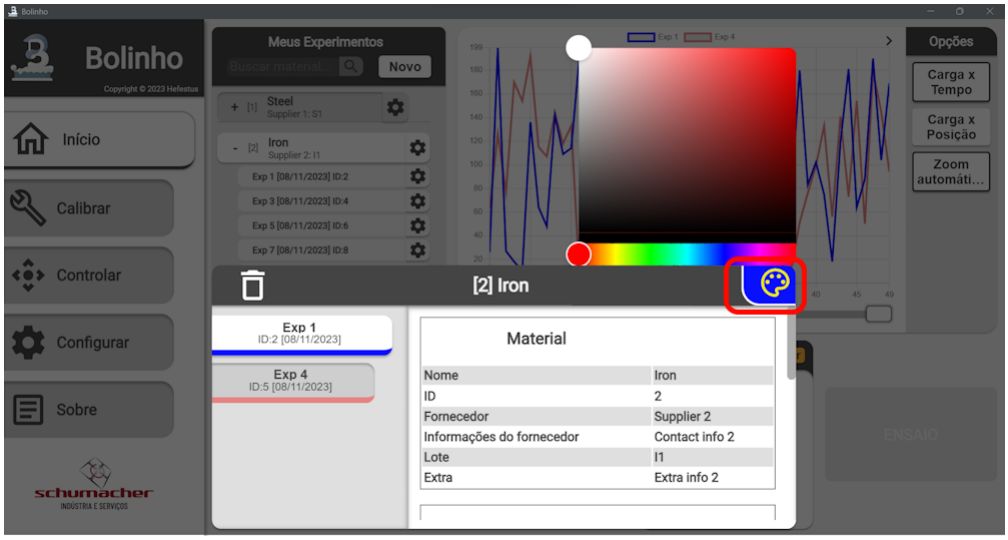
Info

Alguns parâmetros de seus **experimentos** e **materiais** não são editáveis em função de manter sua **integridade de dados**.

Inspetor de experimento

O **Inspetor de experimento** apresenta todos os **experimentos selecionados** e suas informações, aqui você pode encontrar dados sobre o tipo de corpo do experiment, material e mais.

Você pode **alterar a cor de plot** de um experimento aqui:



Plot de experimentos

O **Plot de experimentos** apresenta para você os dados coletados em seu experimento.

A barra horizontal encontrada na parte inferior do componente permite que você navegue o experimento e aumente os detalhes em determinado ponto de interesse.

À direita está localizado a barra de opções com os seguintes botões:

- `Carga X Tempo` : Apresenta o gráfico da carga em função do Tempo.
- `Carga X Posição` : Apresenta o gráfico da carga em função da Posição.
- `Zoom automático` : Restitui o **zoom** para a posição inicial, durante um experimento essa função também acompanha a criação de novos pontos de dado.

A barra de opções também pode ser **minimizada** ao apertar a seta indicadora no canto superior direito.

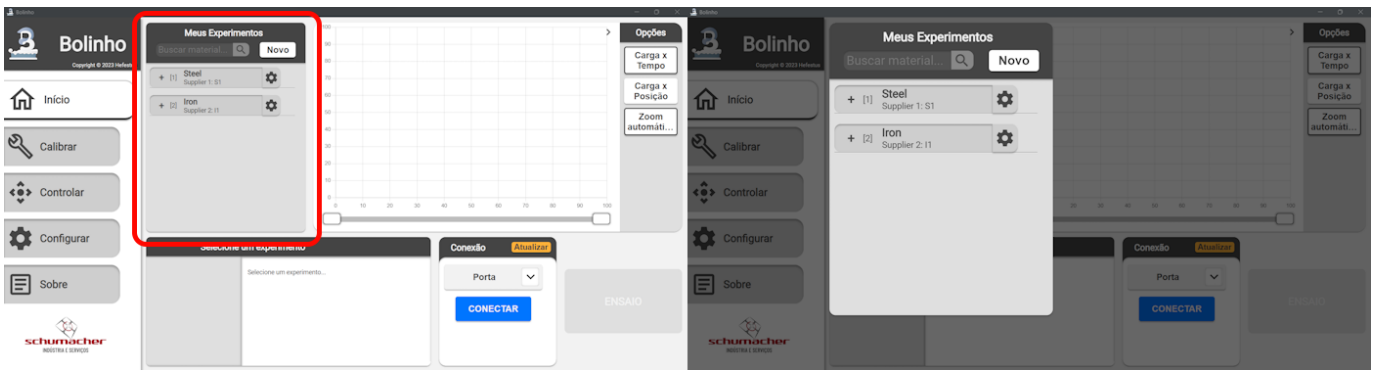
Componente de conexão

O **Componente de conexão** permite que você conecte o Granulado ao Bolinho.

O seletor `Porta` apresentará todas as portas disponíveis ao Bolinho naquele momento. Caso o dispositivo de interesse não esteja aparecendo você poder pressionar `Atualizar` para que o Bolinho recupere os dispositivos conectados mais recentes.

2.2.2 Funcionalidades básicas

A maioria dos softwares do bolinho possuem a capacidade de expandir. Para expandir um componente basta pressiona-lo por `zoomDelay`, por padrão esse valor é setado em `500ms`.



2.3 Configuração

Como configurar.

2.4 Calibração

Como calibrar.

2.5 Controle manual

Como controlar o maquinário manualmente.

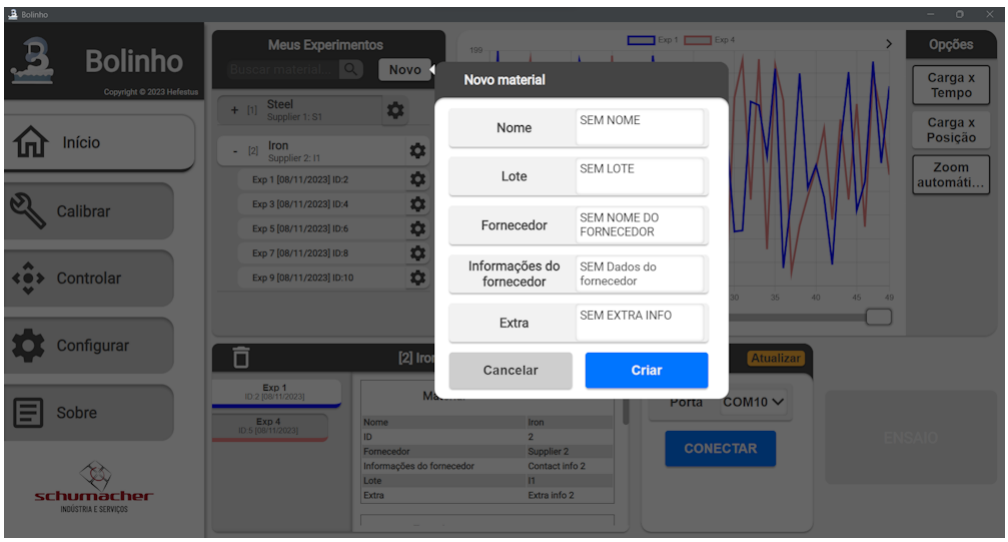
2.6 Novo experimento

Como criar um novo experimento.

2.6.1 Criando um material novo

Pressione o botão `Novo` no componente **Seletor de experimentos**

Ao pressionar o você será apresentado o seguinte prompt de **Criação de Material**



Preencha com os dados de seu material e pressione `Criar`.

> **Note**
>
> Durante um experimento o plote de dados deve ser lido apenas como uma **sugestão** do resultado, já que enquanto um experimento está sendo executado a **quantidade de pontos apresentados no gráfico é reduzido** para poder alocar mais recursos ao experimento em si.

3. Embedded

3.1 Embedded

Bolinho uses a microcontroller esp32-s3 for controlling the hardware.

For more info check the Granulado repository.

3.1.1 Serial communication

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

This communication is done via interrogation, so that the host **prompts** the peripheral for data and it complies.

Protocol:

These are the available commands for the communication between the host and the peripheral.

A instruction is divided in three parts:

`command` `data` `\n`

- `command` is a **1 byte** `character`.
- `data` is the payload as a `string` it can be also empty.
- `\n` is the **line terminator** to identify the end of an instruction.

BOLINHO -> GRANULADO

- `p` -> Ping
- `m[str]` -> Moves stepper motor x millimeters.

  str is an `int` in `string` format.
- `s` -> Stop
- `t` -> Move to top
- `g` -> Get motor position millimeters.
- `r` -> Get instantaneous reading.
- `@` -> Tare load cell
- `w` -> Calibrate known weight
- `x[str]` -> Set known weight

  str is an `int` with the weight in `grams` in `string` format.
- `y[str]` -> Set z-axis length

  str is an `int` with the length of the z-axis in `millimeters` in `string` format.

- `j` -> Get z-axis length
- `z` -> Calibrate z-axis
- `d` -> Get delta load
- `l[str]` -> Set max load

  str is an `int` with the maximum experiment load in `grams` in `string` format.
- `v[str]` -> Set max travel

  str is an `int` with the maximum experiment travel in `mm` in `string` format.
- `a[str]` -> Set max delta load

  str is an `int` with the maximum experiment delta load in `grams / second` in `string` format.
- `-` -> Nothing

GRANULADO -> BOLINHO

- `p` -> Ping Response
- `e[str]` -> Erro.

  str is an `string` with the description of the error.
- `r[str]` -> Returns current reading

  str is an `int` in `grams` in `string` format.
- `g[str]` -> Returns current position in millimeters

  str is an `int` in `string` format.
- `j[str]` -> Returns z-axis length

  str is an `int` in `string` format.
- `b` -> Bottom interrupt was triggered
- `t` -> Top interrupt was triggered
- `d[str]` -> Returns delta load

  str is an `int` in `string` format.
- `s` -> Response to the stop command
- `i[str]` -> Debug info

  str is any `string` to be shown on the terminal.

4. API

4.1 API

In this section you will be able to finde 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 solutions 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**.

4.1.1 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
```

4.2 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()`.

4.2.1 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**

```python
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()
```

set_is_connected()

set_is_connected()

Sets the variable "isConnected" on the front-end.

Python usage example

```
from bolinho_api.core import core_api

core_api.set_is_connected(True)
```

refresh_data()

refresh_data()

Sets the variable "isConnected" on the front-end.

Python usage example

```
from bolinho_api.core import core_api

add_material_to_db() #Arbitrary function that adds a material to the DB

core_api.refresh_data()
```

refresh_realtime_experiment_data()

refresh_realtime_experiment_data()

Triggers a call to refresh the data.

It will refetch the data points of the current experiment.

A use case is to trigger a refresh to show an update on the readings

Python usage example

```
from bolinho_api.core import core_api

core_api.refresh_realtime_experiment_data()
```

4.2.2 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!")
```

loading_alert(text, callback_func)

loading_alert(text, callback_func)

Uses React-Toastify to create an loading alert.

Returns the Toast ID **Asynchronously** to a `callback_func`.

Must be used together with `update_alert`.

Python usage example

```
def save_and_end(toast_id):
    bolinho_app.end_experiment()
    run(bolinho_app.end_experiment())
    core_api.go_to_home_page()
    ui_api.update_alert("Salvo com sucesso!", True, toast_id)

ui_api.loading_alert("AGUARDE! Salvando no banco...", save_and_end)
```

update_alert(text, success, id)

update_alert(text, success, id)

Uses React-Toastify to update an existing alert.

If success is set to true it displays a success other wise shows an error

Python usage example

```
def save_and_end(toast_id):
    bolinho_app.end_experiment()
    run(bolinho_app.end_experiment())
    core_api.go_to_home_page()
    ui_api.update_alert("Salvo com sucesso!", True, toast_id)

ui_api.loading_alert("AGUARDE! Salvando no banco...", save_and_end)
```

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

```python
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

```python
from bolinho_api.ui import ui_api

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

set_save_experiment_progress(total: int, amount: int)

set_save_experiment_progress(total: int, amount: int)

Set the progress bar experiment save.

Python usage example

```
from bolinho_api.ui import ui_api

# Sets the progress to 2%
ui_api.set_save_experiment_progress(100, 2)
```

4.2.3 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`.

set_time(newValue)

set_time(newValue)

Sets the current time of the experiment.

This variable is shown to the user as value and progress bar.

Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_time(22)
```

set_delta_load(newValue)

set_delta_load(newValue)

Sets the current delta load.

This variable is shown to the user as value and progress bar.

Python usage example

```
from bolinho_api.experiment import experiment_api

experiment_api.set_delta_load(22)
```

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)
```

4.3 Backend API

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

Front end -> Backend

4.3.1 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();
```

4.3.2 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);
```

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)
```

4.3.3 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);
};)
```

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());
};)
```

4.4 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 pleas see `src/bolinho_api/classes.py` !

4.4.1 DataPoint

Python

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

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

4.4.2 Material

Python

```python
class Material:
    def __init__(
        self,
        id=0,
        name="NONE",
        batch="",
        supplier_name="",
        supplier_contact_info="",
        extra_info="",
    ):
        self.id = id
        self.name = name
        self.batch = batch
        self.supplier_name = supplier_name
        self.supplier_contact_info = supplier_contact_info
        self.extra_info = extra_info
```

- `id`:
- type: `int`
- Unity: N/A
- `name`:
- type: `string`
- Unity: N/A
- `batch`:
- type: `string`
- Unity: N/A
- `supplier_name`:
- type: `string`
- Unity: N/A
- `supplier_contact_info`:
- type: `string`
- Unity: N/A
- `extra_info`:
- type: `string`
- Unity: N/A

4.4.3 Body

Python

```python
class Body:
    def __init__(
        self,
        id=0,
        type=1,
        material=Material(
            id=0,
            name="Base Material",
            batch="",
            supplier_name="",
            supplier_contact_info="",
            extra_info="",
        ),
        param_a=0,
        param_b=0,
        height=0,
        extra_info="",
    ):
        self.id = id
        self.type = type
        self.material = material
        self.param_a = param_a
        self.param_b = param_b
```

```
        self.height = height
        self.extra_info = extra_info
```

- `id` :
- Type: `int`
- Unity: N/A
- `type` : Body format * 1 = Rectangle * 2 = Cylinder * 3 = Tube * 4 = Other * Type: `int` * Unity: N/A
- `material` :
- Type: `Material`
- 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`
- `extra_info` :
- type: `string`
- Unity: N/A

4.4.4 Experiment

Python

```python
class Experiment:
    def __init__(
        self,
        id=0,
        name="None",
        body: Body = Body(
            id=0,
            type=1,
            material=Material(
                name="Material",
                batch="Batch",
                supplier_name="",
                supplier_contact_info="",
                extra_info="",
            ),
            param_a=0,
            param_b=0,
            height=0,
            extra_info="",
        ),
        date_time=0,
        load_loss_limit=0,
        max_load=0,
        max_travel=0,
        max_time=0,
        z_axis_speed=0,
        compress=False,
        extra_info="",
        plot_color="#ffffff",
    ):
        self.id = id
        self.name = name
        self.body = body
        self.date_time = date_time
        self.load_loss_limit = load_loss_limit
        self.max_load = max_load
        self.max_travel = max_travel
        self.max_time = max_time
        self.z_axis_speed = z_axis_speed
        self.compress = compress
```

```
        self.extra_info = extra_info
        self.plot_color = plot_color
```

- `id` :
- Type: `int`
- Unity: N/A
- `name` :
- type: `string`
- Unity: N/A
- `body` :
- Type: `Body`
- Unity: N/A
- `date_time` : Date and time formatted as `dd/mm/yyyy`
- Type: `string`
- Unity: N/A
- `load_loss_limit` : Max load loss to trigger auto-stop.
- Type: `float`
- Unity: `N/s`
- `max_load` : Max load 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`
- `z_axis_speed` :
- Type: `float`
- Unity: `mm/s`
- `compress` : Is the experiment type of compression? `false` implies expansion.
- Type: `bool`
- Unity: N/A
- `extra_info` :
- type: `string`
- Unity: N/A
- `plot_color` : System parameter
- type: `string`
- Unity: N/A

5. About

5.1 About

This page will present extra info about the project.

5.1.1 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 Circular Progressbar - see license
- React Transition Group - see license
- Reactjs popup - see license

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