# BrainCEMISID 3.0's API

## What is the BrainCEMISID ?

The BrainCEMISID's API is a powerful tool that allows a client app interacts with the kernel relied in the REST architecture, simplifying the prior rough interface into easy-to-recall commands in json format without sacrificing any of the functionalities that the kernel offers. The API was made with DRF (Django Rest Framework) that is used along with Django in order to support the first two layers of the project.

## How to Use It ?

The BrainCEMISID's API responds to the protocol BBCC along with the signal of **SET_ZERO**, **SET_ADD_OPERATOR** and **SET_EQUAL_SIGN**. In this section we are going to describe how the protocol works and how to use the signals in order to build a query taking care of the semantic and syntaxis.

### The BBCC protocol

The BBCC protocol is the set of rules that we use to communicate with the kernel, the name of the protocol is an acronym of the 4 commands used in order to give an instruction. Those commands are:

**BUM**: It is the first command in the sequence, is used to give the signal to the kernel that we are going to initiate more than one command at a time. even though this command is only to give a signal without any knowledge, is necessary to utilize it, otherwise the kernel will recognize single commands as independents instructions rather than a sequence as a whole.

**BIP**: It is the second command in the sequence, is used to put the first data knowledge of the instruction that we need to send to the kernel. However, if you need to put more than 3 data knowledge you have to repeat this command "**N-2**" times being "**N**" the number of data knowledge to live an episode.

**CHECK**: It is the penultimate command in the sequence, if you are going to use less than two commands with patterns to live an episode you should skip this command and put the last data in the **clack** command instead. Also, this command can be used as the last command to *check* some knowledge already learned in the kernel whether you want to execute the bbcc protocol or just the single command.

**CLACK**: It is the last command in the sequence, is used to put the last data knowledge that you would need to send to the kernel for saving. You might also want to use this command for the purpose of send an instruction of only one pattern without using the **BUM** command.

### Signals

The signals are used to give a flag to the kernel of what of the knowledge already learned are special knowledge to make addition operations.

**SET_ZERO**: This signal is use to set an already learned knowledge the definition of zero.

**SET_ADD_OPERATOR**: This signal is use to set an already learned knowledge the mathematic definition of the operator for addition.

**SET_EQUAL_SIGN**: This signal is use to set an already learned knowledge the mathematic definition of the equality sign.

### Data Knowledge

The data knowledge is the data that is send in every command that you use, every command should have data knowledge. Depending of the query, some arguments may or may not be fill with data, instead, you should let it in default values (even the "**BUM**" command should be fill with default values. More details below in the section of examples).

**Hearing pattern**: this field is a 64-size array of **integers** in which you can put values from 0-15 in each position in order to describe the hearing pattern. If you won't use this field, you should instead set "**0**" to all the values in the array.

**Hearing class**: this field is to set the name of the knowledge. If you already saved the knowledge or you are making a "**BUM**" command, you don't have to use this value, hence, you can leave it empty.

**Sight pattern**: this field is a 64-size array of **integers** in which you can put values from 0-15 in each position in order to describe the sight pattern. If you won't use this field, you should instead set "**0**" to all the values in the array.

**Intentions input**: this field is a 3-size array of **floats** in which you have to put the BCF (Biology, Cultural, Feelings) values in that order respectively that can vary from 0-1. If you already saved the knowledge or you are making a "**BUM**" command, you don't have to use this field, hence, you can set "**0**" to all the values in the array.

**Image_id**: this field is an **integer** to pair an image stored in the file system of the platform with the knowledge. If you don't want to use any image stored in the file system you may want to set this value as **-1**, in order to only save the knowledge without pair it.

**Rename**: this field is a **boolean** that is used as a flag is you may want to change the paired knowledge for any other image in the filesystem. You would use it by setting **true** and changing the **image_id**, otherwise you should use it always in **false**.

### Modes

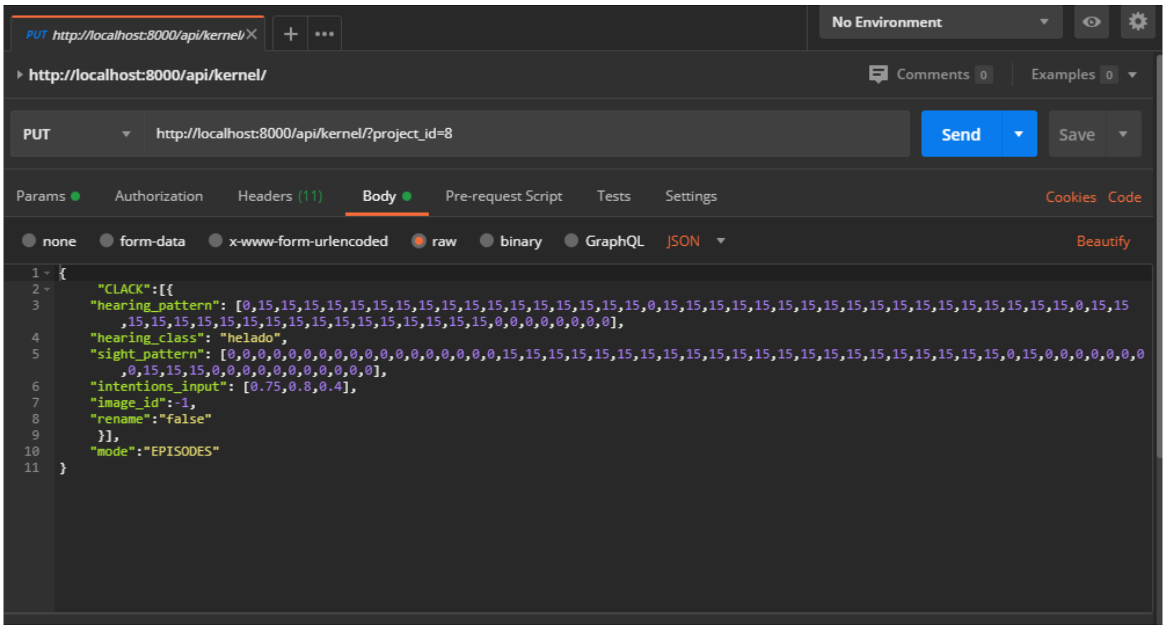
The mode is the type of knowledge the kernel is receiving in order to know what neural networks will use to process it depending in which mode is set. There are several modes, these are, **episodes**, **intentions**, **count**, **reading**, **addition**.

## Example of queries

Every query needs to have at least a command and the **mode** in which the kernel will interpret the knowledge. In every knowledge you should declare the **hearing pattern**, **hearing class**, **sight pattern**, **intentions input**, **image_id** and **rename**. Also, every query has to be in json format.

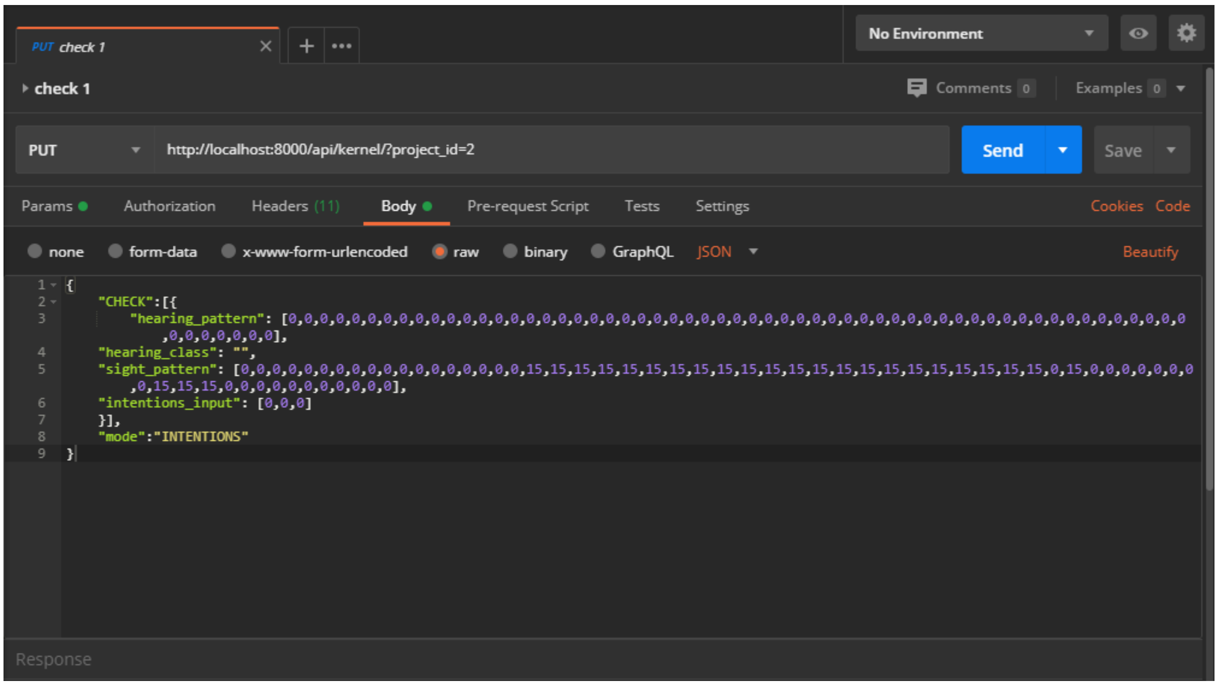
### Simple query

A simple query is composed by a mode and a command. When you are doing a saving or *check* of knowledge you may choose to set in **mode** between ***EPISODES*** or ***INTENTIONS*** they will be interpret as the same by the kernel. You can use the **clack** to save a knowledge and **check** to *check* an already saved knowledge.



PUT http://localhost:8000/api/kernel/?project_id=8

```
{
     "CLACK":[{
    "hearing_pattern": [0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,15
        ,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0],
    "hearing_class": "helado",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,0,0,0,0,0,0,0
        ,0,15,15,15,0,0,0,0,0,0,0,0,0,0,0],
    "intentions_input": [0.75,0.8,0.4],
    "image_id":-1,
    "rename":"false"
     }],
    "mode":"EPISODES"
}
```

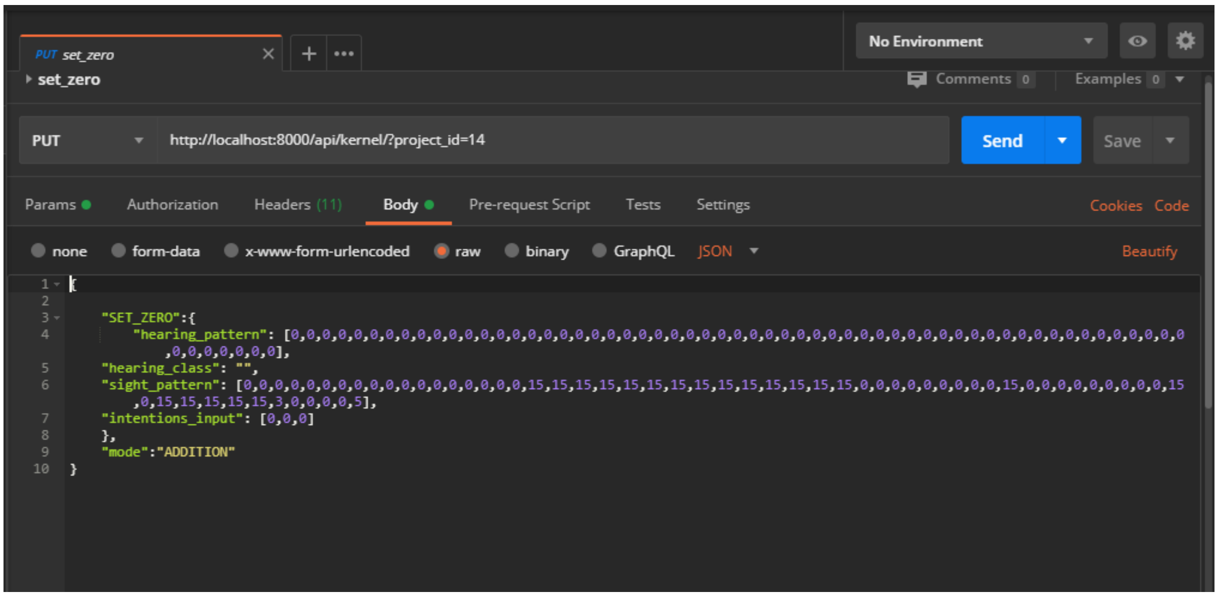


http://localhost:8000/api/kernel/?project_id=2

```
{
    "CHECK":[{
        "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            ,0,0,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,0,0,0,0,0,0,0
        ,0,15,15,15,0,0,0,0,0,0,0,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "mode":"INTENTIONS"
}
```

## Set a signal

To set a signal you have to only pass the sight pattern that matches with the sight pattern of one of the already stored knowledge, and set the mode to ***ADDITION***. Make sure you have already knowledge that you want to use for the signal that you choose by executing a simple query to save knowledge previously.



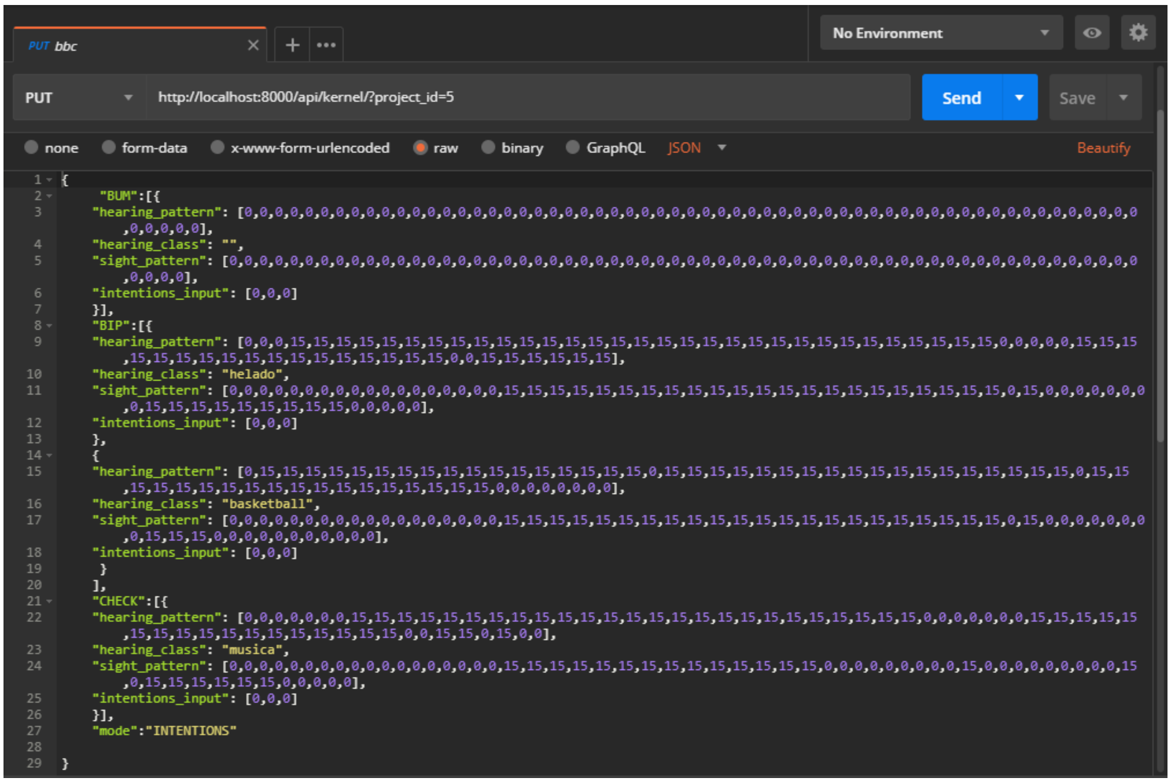
http://localhost:8000/api/kernel/?project_id=14

```
{

    "SET_ZERO":{
        "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            ,0,0,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15
        ,0,15,15,15,15,15,3,0,0,0,0,5],
    "intentions_input": [0,0,0]
    },
    "mode":"ADDITION"
}
```

## Letting the brain live an episode already saved

To let the brain live an episode just simply pass the patterns by executing the bbcc protocol and set the mode to ***INTENTIONS***, also you can choose whether to pass the hearing class or not, this parameter can be left empty but for this example we put the hearing class *only for the developer knows from what knowledge belongs the patterns.*



PUT http://localhost:8000/api/kernel/?project_id=5

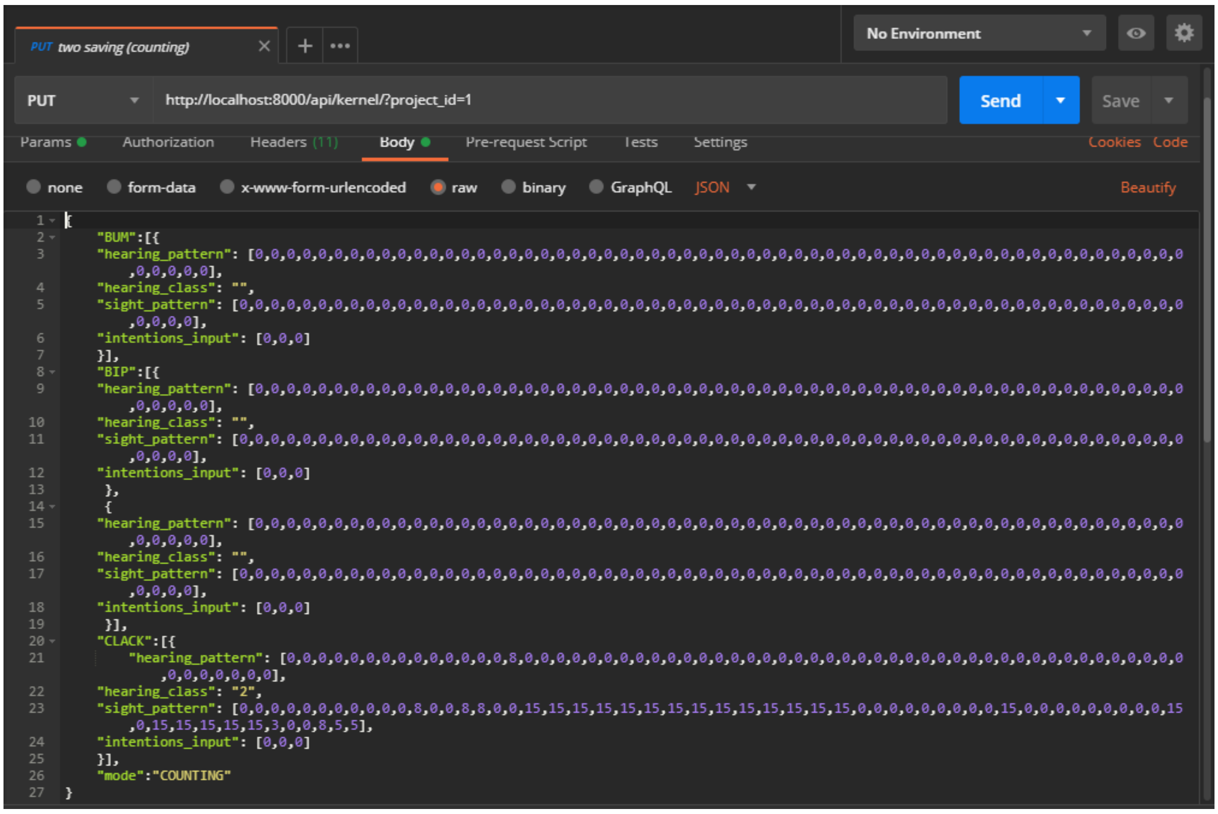
none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
     "BUM":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "BIP":[{
    "hearing_pattern": [0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,15,15,15
        ,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,15,15,15,15,15,15],
    "hearing_class": "helado",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,0,0,0,0,0,0,0
        ,0,15,15,15,15,15,15,15,15,15,0,0,0,0,0],
    "intentions_input": [0,0,0]
    },
    {
    "hearing_pattern": [0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,15
        ,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0],
    "hearing_class": "basketball",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,15,0,0,0,0,0,0,0
        ,0,15,15,15,0,0,0,0,0,0,0,0,0,0,0],
    "intentions_input": [0,0,0]
     }
    ],
    "CHECK":[{
    "hearing_pattern": [0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,15,15,15,15,15
        ,15,15,15,15,15,15,15,15,15,15,15,15,0,0,15,15,0,15,0,0],
    "hearing_class": "musica",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15
        ,0,15,15,15,15,15,15,0,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "mode":"INTENTIONS"

}
```

## Saving a number

To save a number execute the bbcc protocol and pass as many empty **bips** times as the number you would need to save, for example, if you want to save the number "2" you have to pass 2 times the **bip** command, and after that pass the **clack** command with the sight pattern, the hearing class, and setting the mode to ***COUNTING***.

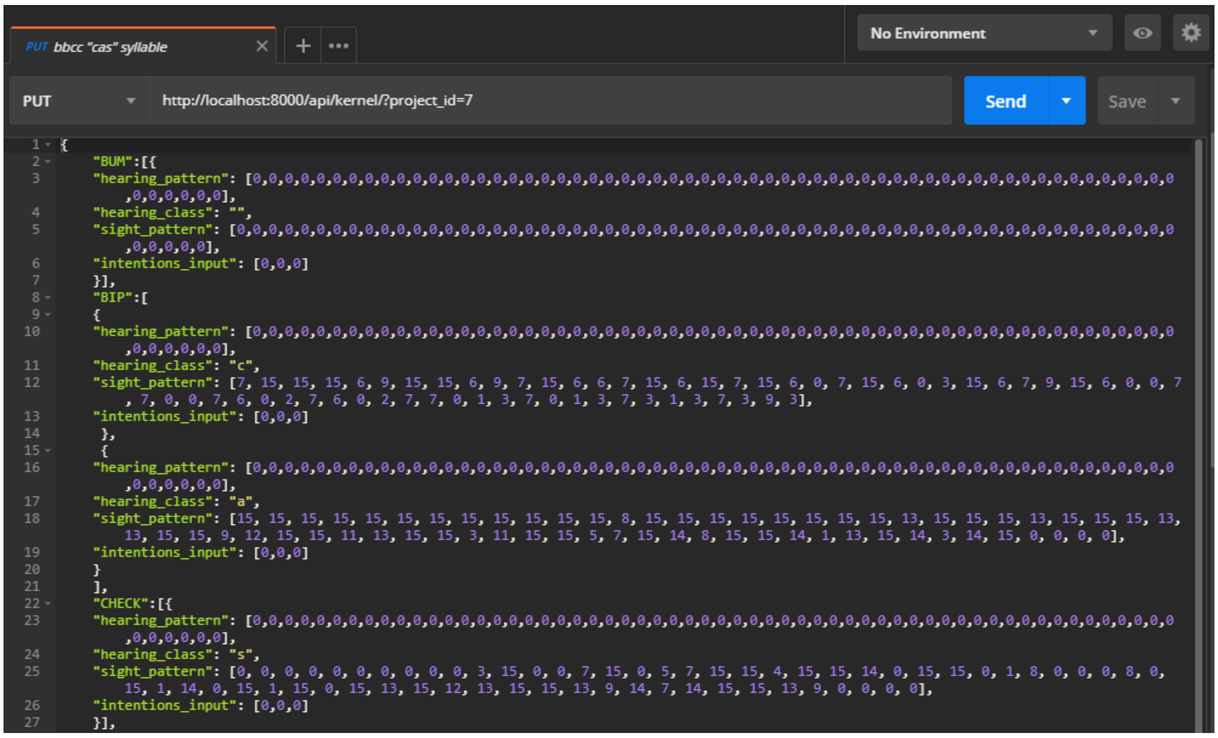


```
http://localhost:8000/api/kernel/?project_id=1
```

```
{
    "BUM":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "BIP":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0],
    "intentions_input": [0,0,0]
     },
     {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0],
    "intentions_input": [0,0,0]
     }],
    "CLACK":[{
        "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
            ,0,0,0,0,0,0,0],
    "hearing_class": "2",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,8,0,0,8,8,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15
        ,0,15,15,15,15,15,3,0,0,8,5,5],
    "intentions_input": [0,0,0]
    }],
    "mode":"COUNTING"
}
```

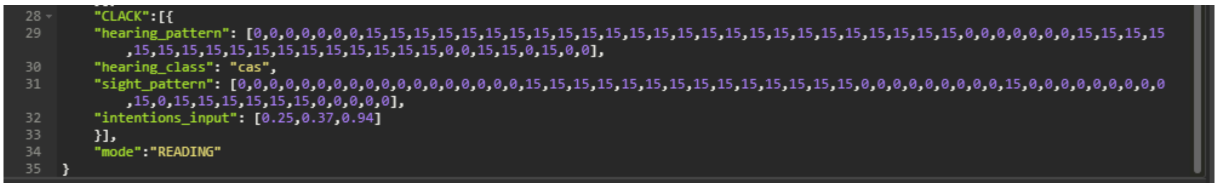
## Saving a syllable

To save a syllable pass the hearing class of the knowledge already saved and the sight pattern by executing the bbcc protocol and set the mode to ***READING***.



PUT http://localhost:8000/api/kernel/?project_id=7

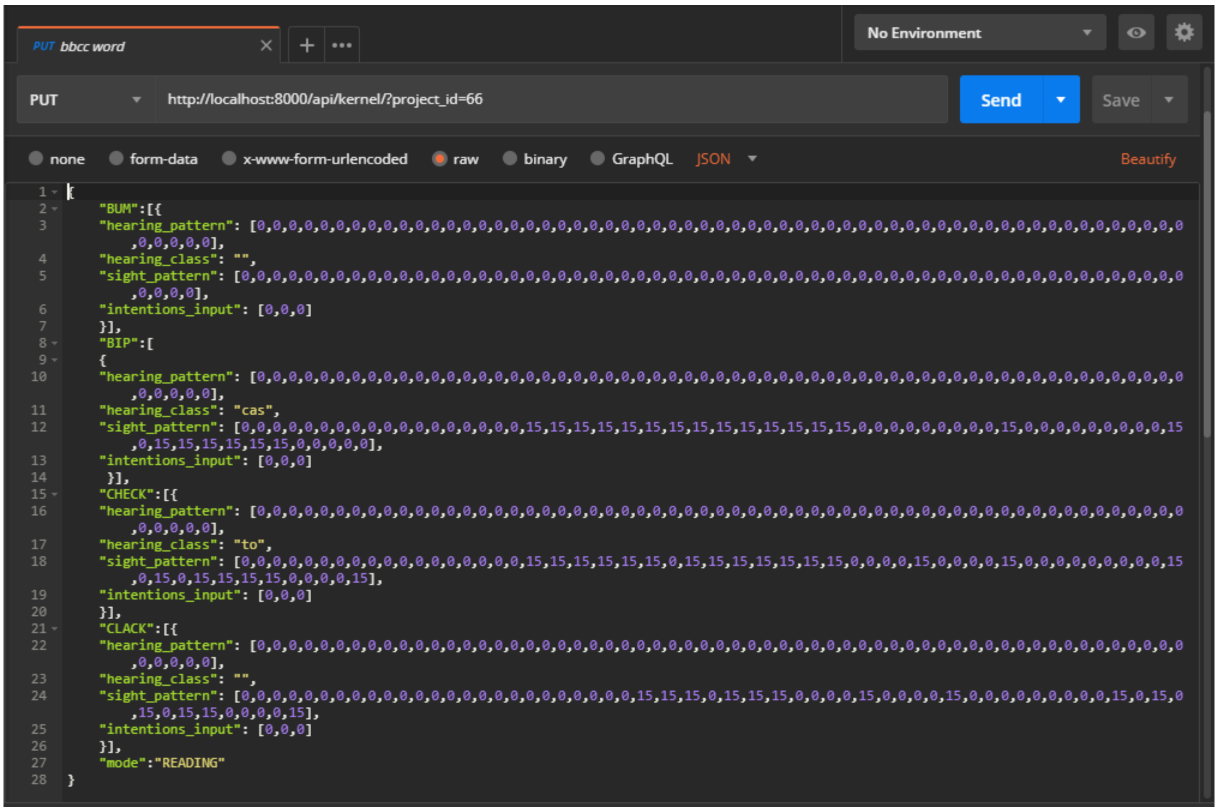
```
{
    "BUM":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "BIP":[
    {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "c",
    "sight_pattern": [7, 15, 15, 15, 6, 9, 15, 15, 6, 9, 7, 15, 6, 6, 7, 15, 6, 15, 7, 15, 6, 0, 7, 15, 6, 0, 3, 15, 6, 7, 9, 15, 6, 0, 0, 7
        , 7, 0, 0, 7, 6, 0, 2, 7, 6, 0, 2, 7, 7, 0, 1, 3, 7, 0, 1, 3, 7, 3, 1, 3, 7, 3, 9, 3],
    "intentions_input": [0,0,0]
     },
     {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "a",
    "sight_pattern": [15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 8, 15, 15, 15, 15, 15, 15, 15, 15, 13, 15, 15, 15, 13, 15, 15, 15, 13,
        13, 15, 15, 9, 12, 15, 15, 11, 13, 15, 15, 3, 11, 15, 15, 5, 7, 15, 14, 8, 15, 15, 14, 1, 13, 15, 14, 3, 14, 15, 0, 0, 0, 0],
    "intentions_input": [0,0,0]
    }
    ],
    "CHECK":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "s",
    "sight_pattern": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 15, 0, 0, 7, 15, 0, 5, 7, 15, 15, 4, 15, 15, 14, 0, 15, 15, 0, 1, 8, 0, 0, 0, 8, 0,
        15, 1, 14, 0, 15, 1, 15, 0, 15, 13, 15, 12, 13, 15, 15, 13, 9, 14, 7, 14, 15, 15, 13, 9, 0, 0, 0, 0],
    "intentions_input": [0,0,0]
    }],
```



```
    "CLACK":[{
    "hearing_pattern": [0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,15,15,15,15
        ,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,15,15,0,15,0,0],
    "hearing_class": "cas",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0
        ,15,0,15,15,15,15,15,15,0,0,0,0,0],
    "intentions_input": [0.25,0.37,0.94]
    }],
    "mode":"READING"
}
```

## Saving a word

To save a word first you had to already saved the syllables that you want to use, then, pass sight pattern and the hearing class of the syllables by executing the bbcc protocol. Make sure also to pass the mode in ***READING***.

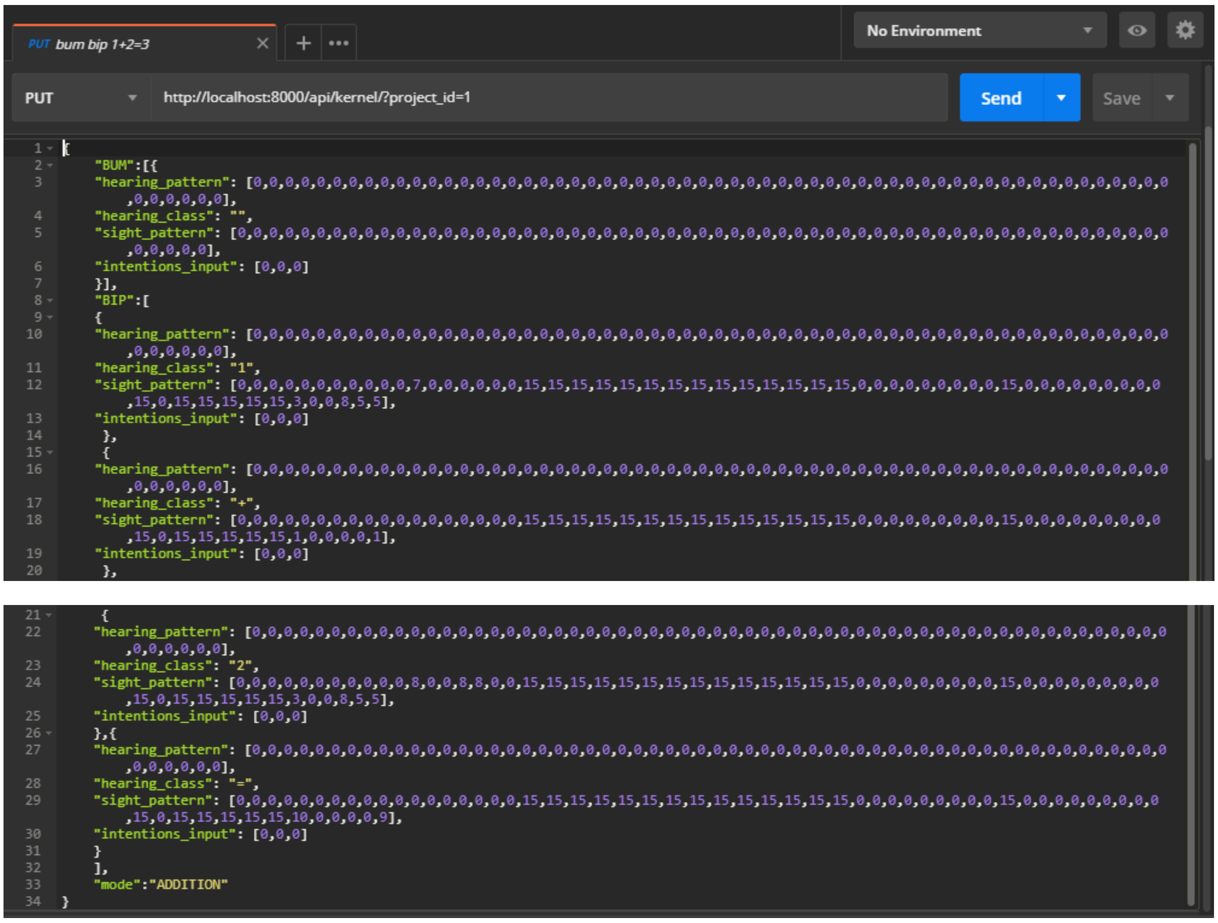


PUT http://localhost:8000/api/kernel/?project_id=66

```
{
    "BUM":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "BIP":[
    {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "cas",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15
        ,0,15,15,15,15,15,15,0,0,0,0,0],
    "intentions_input": [0,0,0]
     }],
    "CHECK":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "to",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,0,15,15,15,15,15,15,15,0,0,0,0,15,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15
        ,0,15,0,15,15,15,15,0,0,0,0,15],
    "intentions_input": [0,0,0]
    }],
    "CLACK":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,0,15,15,15,0,0,0,0,15,0,0,0,0,15,0,0,0,0,0,0,0,0,0,15,0,15,0
        ,15,0,15,15,0,0,0,0,15],
    "intentions_input": [0,0,0]
    }],
    "mode":"READING"
}
```

## Executing an addition

To execute an addition, you can query the kernel by using only **bips** in which you pass the 2 numbers that you would use for the addition. In every **bip** you have to pass the sight pattern and optionally, the hearing class. The order of the commands is **bum**, the first **bip** that would be the first number to add, the signal of the ***ADD_OPERATOR***, the second number to add and lastly, the signal of the ***EQUAL_SIGN***. Make sure don't obliviate set the mode to ***ADDITION***.



```
http://localhost:8000/api/kernel/?project_id=1
```

```
{
    "BUM":[{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0],
    "intentions_input": [0,0,0]
    }],
    "BIP":[
    {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "1",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0
        ,15,0,15,15,15,15,15,3,0,0,8,5,5],
    "intentions_input": [0,0,0]
     },
     {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "+",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0
        ,15,0,15,15,15,15,15,1,0,0,0,0,1],
    "intentions_input": [0,0,0]
     },
     {
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "2",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,8,0,0,8,8,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0
        ,15,0,15,15,15,15,15,3,0,0,8,5,5],
    "intentions_input": [0,0,0]
    },{
    "hearing_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        ,0,0,0,0,0,0],
    "hearing_class": "=",
    "sight_pattern": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,15,0,0,0,0,0,0,0,0,0
        ,15,0,15,15,15,15,15,10,0,0,0,0,9],
    "intentions_input": [0,0,0]
    }
    ],
    "mode":"ADDITION"
}
```

## Last words

You can build the queries with any **REST API** tester but we encourage you to use postman in order to run the queries smoothly. There are more examples of queries in the folder of postman queries in the git of the project on github. Hopefully this manual and the queries would be enough to you to start to develop with the **BrainCEMISID** project !