iMODE REST INTERFACE

# DRAFT

|  |  |  |
| --- | --- | --- |
| URI | HTTP-Methods | Priority |
| users/{u-id}/projects | GET, POST | 1 |
| users/{u-id}/projects/{id} | GET, PUT,DELETE | 1 |
| users/{u-id}/projects/{id}/results | GET | 1 |
|  |  |  |
| users/{u-id}/processes/simulations | POST | 1 |
|  |  |  |
| users/{u-id}/library/components | GET, POST | 1 |
| users/{u-id}/library/components/{id} | GET, PUT,DELETE | 1 |
| users/{u-id}/library/resources | GET, POST | 2 |
| users/{u-id}/library/resources/{id} | GET, PUT,DELETE | 2 |
| users/{u-id}/library/loads | GET, POST | 2 |
| users/{u-id}/library/loads/{id} | GET, PUT,DELETE | 2 |
|  |  |  |
| simulations/{id}/results/data | GET | 2 |
|  |  |  |

users/{u-id}/projects

bin/rails generate model Projects title:string author:string description:string coordinates:string

# GET- Priority 1

This method should return the list of all user’s projects to show on the left sidebar.

NOTE: When the user clicks “Open Projects”, we use these data to visualize all existing projects.

Example of server Response:

INSERT INTO projects

->(title, author, created\_at, description, coordinates)

->VALUES

->("My Project", "Daniel Cantoni", NOW(), “A multiple apartment building with 70 kW.“, “[52.03305432186064, 11.93868568749997]”);

[

{

Id: “L12345”

title: "My Project",

author: "Daniel Cantoni",

lastModified: "21/09/2016",

description: "A multiple apartment building with 70 kW...",

coordinates: [52.03305432186064, 11.93868568749997]

},

{

Id: “L12346”

title: "My Project 2",

author: "Daniel Cantoni",

lastModified: "22/09/2016",

description: "A multiple apartment building with 70 kW...",

coordinates: [52.03305432186064, 11.93868568749997]

process: “” }

]

# POST– Priority 2

Used when the user creates a new project.

The server should return the ID of the created project or an error if the project cannot be saved.

Example of JSON contained in the client request:

{

title: "My Project",

author: "Daniel Cantoni",

description: "A multiple apartment building with 70 kW...",

coordinates: [52.03305432186064, 11.93868568749997],

resources: \*\* to be defined \*\*,

loads: \*\* to be defined \*\*,

components: {

electric: {

eg: ["URI1", "URI2", …],

hp: ["URI3", "URI4", …],

}

heat: { … }

cooling: { … }

},

system: \*\* to be defined \*\*,

settings: \*\* to be defined \*\*

}

NOTE: projects created by using a default template have a default ID of L0000. This ID is going to be checked any time the user wants to save the project (or immediately before issuing a CALCULATE command), because the user must save the project as a new one, in order to change its ID.

users/{u-id}/projects/{id}

# GET– Priority 1

Get all information about a project, whose unique identifier is the parameter *{id}*.The JSON returned from the serverrefers to a single project.

NOTE: We call this method when the user clicks the button*activate* to open and modify a project.

Example of server Response:

{

title: "My Project",

author: "Daniel Cantoni",

lastModified: "21/09/2016",

description: "A multiple apartment building with 70 kW...",

coordinates: [52.03305432186064, 11.93868568749997],

resources: \*\* to be defined \*\*,

loads: \*\* to be defined \*\*,

components: {

electric: {

eg: ["URI1", "URI2", …],**\*\*\*\* See NOTE #1 \*\*\***

hp: ["URI3", "URI4", …],

}

heat: { … }

cooling: { … }

},

system: \*\* to be defined \*\*,

settings: \*\* to be defined \*\*

}

NOTE#1:

The URI represents the address of the REST resource(s) associated to a component sent by the server to the client. This URI can dynamicallychange at runtime according to server development choices (HATEOAS REST). Client completely ignores how the URI is built.

# PUT– Priority 1

Method used to update information about a project when the user saves the project.

NOTE: *lastModified*attributeshould be added at the server side.

Example of JSON contained in the client request:

{

title: "My Project",

author: "Daniel Cantoni",

description: "A multiple apartment building with 70 kW...",

coordinates: [52.03305432186064, 11.93868568749997],

resources: \*\* to be defined \*\*,

loads: \*\* to be defined \*\*,

components: {

electric: {

eg: ["URI1", "URI2", …],

hp: ["URI3", "URI4", …],

}

heat: { … }

cooling: { … }

},

system: \*\* to be defined \*\*,

settings: \*\* to be defined \*\*

}

The server response should contain a status code as JSON string:

* OK: If the project has been saved
* FAIL: if the JSON contains error or if the server can’tsave the project

# DELETE– Priority 2

Method used to delete a project.

users/{u-id}/projects/{id}/results

# GET– Priority 1

Method used to view the simulation results.

The format of the server response depends on the **media type** requested from the client, it could be:

* JSON (**Priority 1**)
* COMPRESSED DATA (**Priority 2**): see <https://github.com/dberardo/toShare/blob/master/Time_series_transmission.md>

NOTE: this call will usually be subject to a REDIRECT on the server, de facto addressing the URI: simulations/{id}/results.  
The existence of an absolute URI for the simulations is important for the developer, because from there a list of all the simulations can be retrieved, independently from the User-ID.

users/{u-id}/processes/simulations

# POST– Priority 1

With this method the client sends a request to execute a simulation.

NOTE: The user cannot run a simulation for a new project that has been not saved. See note on PUT users/{u-id}/projects/{id}

Example of JSON contained in the client request:

{

priority: 1,

projectID: XY,

.

.

.

}

NOTE: to avoid that the user starts a simulation without saving a project, the client automatically saves the project before starts the simulation. To do that, when the user clicks on *save and calculate* button (note: the *calculate* button will be renamed in *save and calculate*), the client sends a PUT to **users/{u-id}/projects/{id}**and if the response of this call is OK, a POST to **users/{u-id}/processes/simulations**will be done.

users/{u-id}/library/components

# GET– Priority 1

This method should return the list of all available components.

Category

Category ID -> one to many Category ID

bin/rails generate model ComponentDetails category\_id: integer name:string <url:string> iconURL:string

NOTE: When the user opens the “Components” tab, we use these data to visualize all available components on the left sidebar split by category (Electric, Heat, Cooling).

Example of server response:

[

{

category: “Electric”,

items: [

{

name: “Electric generator”,

url: URI1, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

},

{

name: “Wind turbine”,

url: URI2, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

}

]

},

{

category: “Heat”,

items: [

{

name: “Heat storage”,

url: URI4, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

},

{

name: “Solar thermal”,

url: URI5, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

}

]

},

{

category: “Cooling”,

items: [

{

name: “Compressor chiller”,

url: URI7, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

},

{

name: “Storage”,

url: URI8, **\*\*\* NOTE #2 \*\*\***

iconURL: “icon path”

}

]

}

]

NOTE #2:  
This URL is used to retrieveall manufacturers and models about a category type (eg: electric generator, heat pump, …). We call this URL when the user clicks on a specific category type (eg: electric generator) on the left sidebar.

An example of the response to the URI1, URI2… call (**provided by the server**, thus we completely ignore the link structure: we just call it) is the following JSON:

[

{

manufacturer: “Brand 1”,

models: [

{

model: “Model 1”,

url: URI1

},

{

model: “Model 2”,

url: URI2

}

]

},

{

manufacturer: “Brand 2”,

models: [

{

model: “Model A”,

url: URI3

},

{

model: “Model B”,

url: URI4

}

]

}

]

This JSON string specifies which models are available for each manufacturer, including a URI to fetch the datasheet for a specific component.

# POST– Priority 1

Used to create a new component. All the information must be sent to the server for the new component creation. Returns an OK status and a link to the resource (or equivalently an ID)

NOTE: this method is called when the “Save new model” button is pressed from the *Components* tab.

Example of data sent to the server:

{

cateogry: (*electric* | *cooling* | *heating*),

type: (*hp* | *eg* | ….),

manufacturer: “BrandWonderful”,

model: “ModelXYZ”,

shortDescription: “Lorem ipsum dolor…”,

notes: “This model is the one used also in project AAA…”,

.

.

.

.

}

users/{u-id}/library/components/{id}

# GET– Priority 1

Get all information about a component whose unique identifier is the parameter *{id}*.

NOTE: we call this method when the user loads in the *Components* tab a new type by selecting manufacturer and model in the drop-down menu in the component view (top left) and also when the user clicks on a specific component listed under the two drop-down menus for manufacturer and model selection.

Example of server response:

{

lastModified: date

cateogry: (*electric* |*cooling* |*heating*), **\*\*\* NOTE #3 \*\*\***

type: (*hp* | *eg* | ….) **\*\*\* NOTE #3 \*\*\***

manufacturer: “BrandWonderful”,

model: “ModelXYZ”,

shortDescription: “Lorem ipsum dolor…”,

notes: “This model is the one used also in project AAA…”,

.

datasheet: “URI1”

.

.

}

NOTE #3:

Fields *category* and *type* are necessary to correctly update the JSON that represents a project because the attribute *components*of the JSON contains all components grouped by category and type.

# PUT– Priority 2

Changes the component data set storing all the information to the server.

NOTE: the update behavior/user experience must be defined yet.

Example of data sent to the server:

{

cateogry: (*electric* | *cooling* | *heating*),

type: (*hp* | *eg* | ….),

manufacturer: “BrandWonderful”,

model: “ModelXYZ”,

shortDescription: “Lorem ipsum dolor…”,

notes: “This model is the one used also in project AAA…”,

.

.

.

.

}

# DELETE– Priority 2

Used to delete a specific component.

NOTE: the update behavior/user experience must be defined yet.

NOTE: system defaults cannot be deleted (ERROR STATUS CODE)