Base url

By now, `https://biocom.uib.es/halodb/`

Backend Entry Points

The following entry points doesn't require access token.

- `/users/` (GET) Returns a list of users currently registered in the system.
- `/groups/` (GET): Returns a list of groups.
- `/sequences/` (GET): Returns the list of the genomic sequences considered and their possible components. The description of the relation between the different parts of the experiments, and their relative order. The returned value is this exact value:


```
{
    "METAGENOME": [
        "RAW READS",
        "TRIMMED READS",
        "CONTIGS",
        "PREDICTED GENES",
        "MAGS"
    ],
    "METATRANSCRIPTOME": [
        "RAW READS",
        "TRIMMED READS"
    ],
    "METAVIROME": [
        "RAW READS",
        "TRIMMED READS",
        "CONTIGS",
        "PREDICTED GENES",
        "CONTIGS VIRUS"
    ],
    "GENOME PROCARIOTA": [
        "RAW READS",
        "GENOME",
        "PREDICTED GENES"
    ],
    "GENOME VIRUS": [
        "RAW READS",
        "GENOME",
        "PREDICTED GENES"
    ],
    "PROTEOMICS": [
        "PEPTIDES"
    ],
    "SINGLE CELL GENOMICS": [
        "RAW READS",
        "SINGLE CELL GENOME",
        "PREDICTED GENES"
    ],
    "PLASMID": [
        "RAW READS",
        "PLASMID",
```

```
        "PREDICTED GENES"
    ]
}
```

- `/query/sequence/<string:name>/` (GET): Returns the genomic sequence steps related to the specified sequence name.
- `/query/<string:table>/` (GET): Returns the set of categories and the corresponding range for the specified classification table (`temperature`, `ph`, `salinity`).
- `/query/<string:table>/<float:value>/` (GET): Returns the corresponding category for the specified table and value.

User related entry points

- `/login` (POST) User identification. It requires a json with an email and a password

```
{
"email": "user registered email",
"password": "xxxxx"
}
```

If the user and password are valid, returns an access token

```
{
"token": "access token to use"
}
```

The access token has to be used as a Bearer Token.

If the user is not valid, then returns a rejection message. In general, it's the same format for each entry point.

```
{
"message": "Invalid username or password"
}
```

If a token expired is used, then the message received is

```
{
    "message": "Expired token",
    "status": "error"
}
```

- `/user/` (POST): Creates a new user. Data to provide

```
{
"email": "email to be used as identification",
"name": "name of the user",
"surname": "surname of the user",
"password": "xxxxx"
}
```

Returns a json with the confirmation message and the corresponding data (the same user data with the unique id for the user), or an error message. The provided password isn't sent.

```
{
"message": {
    "status": "success",
    "message": "User created",
        "user": {
```

```
            "email": "email to be used as identification",
            "name": "name of the user",
            "surname": "surname of the user",
            "uid": "unique identifyer associated to the user",
            "id": an integer, also unique identifyer
        }
    }
}
```

- `/user/` (GET, DELETE):

GET: Retrieves user information based on the provided user uid. The user information is extracted from the access token. The returned json is:

```
{
"message": {
    "name": "name of the user",
    "surname": "surname of the user",
    "email": "email to be used as identification",
    "uid": "unique identifier associated to the user",
    "id": an integer, also unique identifyer,
    "registration_time": "date of the user registration"
},
"token": "the same token provided or an updated one if the expiration time is near"
}
```

DELETE: Deletes a user based on the provided user uid. The user information is extracted from the access token. That means that the user deleted is the identified by the access token. The response if the user has been removed is:

```
{
"message": {
    "status": "success",
    "message": "User deleted"
},
"token": "the same token provided or an updated one if the expiration time is near"
}
```

- `/user/` (PUT, PATCH): Updates the user information based on the provided user ID.

A json with the fields to be changes has to be sent. Also, an access token. The updatable fields are name, surname, and email.

If the update is correct, then the message obtained is

```
{
"message": {
    "status": "success",
    "message": "User updated",
        "user": {
        "name": "name of the user",
        "surname": "surname of the user",
        "email": "email to be used as identification",
        "uid": "unique identifier associated to the user",
        "id": an integer, also unique identifyer,
        "registration_time": "date of the user registration"
    }
```

```
},
"token": "the same token provided or an updated one if the expiration time is near"
}
```

- `/user/list/<string:table>/` (GET): Retrieves a list of elements related to the user from the specified table (e.g., groups, projects, ….).

```
{
"message": [
    {
    item 1,
    },
    {
    item 2,
    },
    {
    item 3,
    }
    .
    .
    .
],
"token": "the same token provided or an updated one if the expiration time is near"
}
```

The possible values are the different experiments, the samples, the groups or the projects.

The option `groups` has the following values:

```
{
"relation": "owner" | "member",
"group_id": unique identifier of the group an integer,
"name": "group name",
"description": "group description"
}
```

The option `project` has the following values:

```
{
"project_id": unique identifier of the project, an integer,
"name": "project name",
"description": "project description"
}
```

The option `sample`, and also the other experiment tables, has the corresponding fields described in their document. Also, the following fields are provided (Note: 0|1 means a boolean value):

```
"public": 0|1,                                  is public
"owned": 0|1,                                   the user is the owner
"shared_by_group": 0|1,                         the user has access by a group
"shared_by_others": 0|1,                        the user has access to a element of other user
"access_mode": "readwrite"|"read",
"group_relation": null | "owner" | "member",
"group_id": null | id of the group,
"group_name": null | "name of the group",
```

Group related entry points

- `/group/` (POST): Creates a new group. The values to provide are

```
{
"name": "group name",
"description": "group description"
}
```

- `/group/` (GET, DELETE):

GET: Retrieves group information based on the provided group ID.
DELETE: Deletes a group based on the provided group ID.

- `/group/<int:group_id>/` (PUT, PATCH): Updates group information based on the provided group ID.
- `/group/list/<string:table>/` (GET): Retrieves a list of elements related to the user from the specified table (e.g., groups, experiments, projects, samples).
- `/group/invitation/<int:group>/` (PUT, PATCH, DELETE): Accepts or rejects an invitation to join a group.
- `/group/invite/<int:invited>/<int:group>/` (POST): Invites a user to join a group.

Project related entry points

- `/project/` (POST): Creates a new project. The values to provide are

```
{
"name": "project name",
"description": "project description"
}
```

- `/project/` (GET, DELETE):

GET: Retrieves project information based on the provided project ID.
DELETE: Deletes a project based on the provided project ID.

- `/project/<int:project_id>/` (PUT, PATCH): Updates project information based on the provided project ID.
- `/project/list/<string:table>/` (GET): Retrieves a list of elements related to the project from the specified table (e.g., tasks, milestones).

Genomic sequences parts

Those are the entry points provided to manage the different genomic sequence steps. All of them have an url starting by `/<string:step>/` this is the type of genomic sequence to be considered.

- `/<string:step>/<int:step_id>/share/group/` (PUT, PATCH): Shares a a genomic sequence step, `step`, with a group. Requires `step_id` and `group_id`.
- `/<string:step>/<int:step_id>/share/group/<int:group_id>/` (DELETE): Stops sharing the genomic sequence step with a group. Requires a `step_id` and a `group_id`.
- `/<string:step>/` (POST): Uploads a genomic sequence step. Requires a json with the corresponding data and also a `source_id`, the genomic sequence step origin of the current.

- `/<string:step>/<int:step_id>/` (PUT, PATCH): Updates some or the whole genomic sequence step data. The corresponding files are handled with a different entry point
- `/<string:step>/<int:step_id>/<input_type>/` (PUT, PATCH): Uploads a genomic sequence step file, requires a key, `file`. The value `input_type` describes the field associated with the file to be uploaded.
- `/<string:step>/<int:step_id>/` (GET): Retrieves the data corresponding to genomic sequence step.
- `/<string:step>/<int:step_id>/<input_type>/` (GET): Retrieves a genomic sequence step file. The value `input_type` describes the field associated with the file to be downloaded