ADE-Scheduler

Release 2.0

Jérome E. & Louis N. & Gilles P.

Feb 28, 2023

TUTORIALS:

1 Quickstart **3**
1.1 Installation 3
1.2 Documentation 3

2 Contributor Guide **73**

3 Index et tables **75**

Python Module Index **77**

Index **79**

This documentation is here to help contributors understanding how the project is built.

CHAPTER

ONE

QUICKSTART

1.1 Installation

To setup your ADE Scheduler project locally, we recommend you to follow our *Setup tutorial*.

1.2 Documentation

- **Tutorials**

Note: This section contains many tutorials on how ADE Scheduler is built and how to run the project on your local machine.

1.2.1 Setup tutorial

Everything you need in order to run this project

Table of content

- *Setup tutorial*
  - *(*New*) Docker setup*
    - *I. Install Docker*
    - *II. Pull Docker image*
    - *III. Build and run the ADE Scheduler server*
  - *1. Redis server*
    - *I. Install Redis*
    - *II. Check that it works*
  - *2. ADE Scheduler*
    - *New to Github ?*
    - *I. Clone the repository*
    - *II. Install NodeJS and NPM*

    * III. Install Python
    * IV. Create a virtual environment
    * V. Initialize the environment file
  - 3. Database
    * I. Lite setup
      · I-a. Install SQLite3
      · I-b. Create a database
    * II. Complete setup
      · II-a. Install PostgreSQL
      · II-b. Setup a password
      · II-c. Create a database
      · II-d. Setup read without password access
    * III. Link database in .flaskenv
    * IV. Populate the database
  - 4. ADE API
    * I. With credentials
    * II. Without credentials
  - 5. Email
    * I. With credentials
    * II. Without credentials
  - 6. Launch
  - 7. Contributing
    * I. Install pre-commit
    * II. Create a branch

This page contains all the required steps that you must follow in order to have a fully working development environment. Each tutorial can be found its in own section with more details on how to use particular tools.

Warning: It is important to follow this setup guide in the correct order of sections!

(*New*) Docker setup

A new setup alternative is the Docker setup. This setup is a lot more simple as it will install most of the requirements for you.

Note: This section will describe how to install and run ADE Scheduler under Docker on your system.

I. Install Docker

First, you must have Docker installed on your system. We recommend to install it from the official website: https://docs.docker.com/desktop/#download-and-install

II. Pull Docker image

Pull the Docker image:

```
$ docker pull gilponcelet/ade-scheduler
```

III. Build and run the ADE Scheduler server

Run the following in <repo>:

```
$ docker build -f Dockerfile -t ade-scheduler .
$ docker run --name ade-scheduler -it -p 5000:5000 -v <Path to ADE-Scheduler folder>:/
↪ADE-Scheduler ade-scheduler
$ docker start -i ade-scheduler       # To run the app
$ docker exec -it ade-scheduler bash  # To e.g. run `flask shell`
```

Warning: This tutorial is still under development! Any problem you might encounter is important for us, so please contact use whenever you need help :-)

1. Redis server

Note: A Redis server is used as a cache memory storage and will, among others, act as a buffer between the ADE API and ADE Scheduler in order to enhance performance and to reduce delay between repetitive requests.

I. Install Redis

The installation process is pretty simple and we recommend you to follow the official guide: https://redis.io/topics/quickstart

II. Check that it works

Before running into more troubles, make sure that Redis is correctly installed and that you can manually start a server from command line. Something like this should work:

Listing 1: Start this in one terminal

```
$ redis-server
do not kill this terminal
```

Listing 2: Then, do this in another terminal

```
$ redis-cli ping
you should receive `pong`
$ redis-cli shutdown
this kills the redis server
```

2. ADE Scheduler

Note: This section will describe every tools that is brought with the ADE Scheduler repository. While many tools can be installed separately, the Github of this project contains many tools and things to setup that can be done in a few clicks.

New to Github ?

If you are new to Github, please consider reading this tutorial first: https://guides.github.com/activities/hello-world/

I. Clone the repository

If you are not one of the official maintainers of this project, you should first fork this repository: https://github.com/ADE-Scheduler/ADE-Scheduler/fork

Then, clone the forked repository somewhere on your machine. We will always refer to the location of the project directory as `<repo>`.

II. Install NodeJS and NPM

First, you need NodeJS and NPM (Node Package Manager). It can be downloaded on the official site: https://nodejs.org/en/

Once it is installed, go to the `<repo>` directory and run:

```
$ npm install
```

This will install all the Javascript packages we use in the project.

Any javascript or HTML asset needs to be bundled by webpack before being used. To do so, run in `<repo>`:

```
$ npx webpack
```

Moreover, you can activate the file watcher for hot-reloading and map source files for easy debugging by running:

```
$  npx webpack --watch --devtool inline-source-map
```

If an error appears about the maximum amount of allowed watchers, you can fix this by running:

Listing 3: from https://stackoverflow.com/questions/53930305/nodemon-error-system-limit-for-number-of-file-watchers-reached

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf \
&& sudo sysctl -p
```

III. Install Python

The project can work with a variety of Python versions, but we recommend Python 3.9.X as this is the version that is currently used in production. It can be downloaded from the official site: https://www.python.org/

Before installing Python packages: Some external packages require external dependencies, and missing them will cause errors during the installation process. Here are a few: - OpenSSL: https://www.poftut.com/install-use-openssl-library-python-applications/ - OpenLDAP: https://stackoverflow.com/questions/4768446/i-cant-install-python-ldap

IV. Create a virtual environment

Event though this step is not mandatory, it is good practice and highly recommended to create a Python virtual environment. This can be done in a few lines:

```
$ cd <repo>
$ sudo apt install python3-virtualenv
or equivalent command if you are not on ubuntu
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r prod-requirement.txt # For production  (if you do not plan on modifying␣
↪code)
$ pip install -r dev-requirements.txt # For development (else)
```

> **Warning:** Whenever you want to run the project, you need to be in this virtual environment. You will then need to activate the environment each time you come back to work. Some IDEs such as PyCharm provide tools to automatically activate virtual environments.

V. Initialize the environment file

In this project, we use many configuration files. One of these files will carry sensible information such as password and it is your duty to create this file and to keep it secret.

Create a file named `.flaskenv` in the `<repo>` directory and fill the following lines in:

Listing 4: `<repo>/.flaskenv` content for development

```
FLASK_APP = path/to/<repo>/app.py
FLASK_ENV = development
FLASK_RUN_HOST = localhost
FLASK_RUN_PORT = 5000
TEMPLATES_AUTO_RELOAD = True
```

(continues on next page)

(continued from previous page)

```
FLASK_SECRET_KEY = <super_secret_key>
FLASK_SALT = <super_complex_salt>
```

Listing 5: `<repo>/.flaskenv` content for production

```
FLASK_APP = path/to/<repo>/app.py
FLASK_ENV = production

FLASK_SECRET_KEY = <super_secret_key>
FLASK_SALT = <super_complex_salt>
```

Lines will be added to this file in other tutorials.

3. Database

Note: For permanent or long-term storage, a SQL database is used. While many implementations exist, we choose to use PostgreSQL for its robustness against high activity. SQLite has proven to be a bottleneck in performances for ADE Scheduler (in production) but can still be used for development.

I. Lite setup

Note: If it is your first time working on this project, we highly recommend you to follow this setup. The complete setup is only required for maintainers working with the database hosted on the ADE Scheduler server.

I-a. Install SQLite3

You will need to install SQLite3 to perform operation on your database. To do so, refer to instructions related to your machine. For ubuntu:

```
$ sudo apt-get install sqlite3
```

I-b. Create a database

You will need to create an empty database, wherever you want, on your machine. Just create an empty file name `ade-database.db`.

Please, remember the location of the database, as you will need it for later in the tutorial. Now, you can directly jump to section *III*.

II. Complete setup

II-a. Install PostgreSQL

For this, we highly recommend you to follow this guide: https://www.postgresqltutorial.com/install-postgresql/

II-b. Setup a password

Even if it is not always required, it is good practice to setup a password for the `postgres` user:

Listing 6: Tutorial from: https://docs.boundlessgeo.com/suite/1.1.1/dataadmin/pgGettingStarted/firstconnect.html

```
$ sudo -u postgres psql postgres
$ \password
enter your password_psql and confirm
$ \q
```

II-c. Create a database

In order to manipulate databases, you need to create a database instance:

```
$ sudo su - postgres
$ createdb ade-database
$ exit
```

II-d. Setup read without password access

By default, ADE Scheduler tries to access the database without password. Here, we need to explicitly allow the program to access the newly created database without password:

Listing 7: You main need to replace *12* with your actual version if it differs

```
$ sudo {vim|geany|nano|...} /etc/postgresql/12/main/pg_hba.conf
and change `peer`/`md5` values to `trust`
$ sudo systemctl restart postgresql
```

III. Link database in .flaskenv

Now, you will need to tell the program where your database is located. To do so, add this line in your `<repo>/.flaskenv` file:

```
$ ADE_DB_PATH="sqlite:///<path/to>/ade_database.db"
for SQLite3 (warning, <path/to> may be "/home/..." so it will add 1 more "/")
$ ADE_DB_PATH="postgresql://postgres@localhost/ade-database"
for PostgreSQL
or, alternatively, you can use an other database you have
$ ADE_DB_PATH=<database URI>
```

IV. Populate the database

If your database is empty, you need to populate it with the correct tables and columns. This can be done using the client:

```
$ flask sql init
```

4. ADE API

Note: The ADE API is the interface allowing ADE Scheduler to obtain all the information it needs, such as events, courses, etc., from the ADE server at UCLouvain.

I. With credentials

If you are one of the maintainers, you should have received credentials in order to proceed requests to the ADE API. If you do not have such credentials, please read next section.

In the `<repo>/.flaskenv` file, make sure to write those lines and complete with your credentials:

```
ADE_URL = <url to get token>
ADE_DATA = <grant_type>
ADE_AUTHORIZATION = <authorization access>
```

II. Without credentials

Because we cannot share the credentials with everyone, but we want people to be able to have the best environment to contribute to ADE Scheduler, we created a fake API. To use this fake API, you will need to modify the `<repo>/.flaskenv`

```
ADE_URL = <leave empty>
ADE_DATA = <leave empty>
ADE_AUTHORIZATION = <leave empty>

ADE_FAKE_API = true
```

Next, you need to download a compressed folder containing the fake api as a folder containing pickled requests and a README file explaining everything that you can request with this API. The `fake_api` folder should be placed in the `<repo>` folder. Please contact us if something isn't working as expected.

Note: We know it is not the best way to provide you an access to the ADE API but we are still working on this and propositions are more than welcome!

5. Email

Note: In order to quickly receive feedback if anything goes wrong, ADE Scheduler will send you an email everytime an error occurs. This can be, of course, disabled to avoid email spamming. This email address can also be used to send account confirmation emails.

I. With credentials

Here, you will allow ADE Scheduler to send emails using your email address. You may need to allow third-party applications to let ADE Schedule use your email address. Please refer to your email provider documentation if so. If you do not want to use the emails, please read next section.

In the `<repo>/.flaskenv` file, make sure to write those lines and complete with your credentials:

```
MAIL_USERNAME = <email address>
MAIL_PASSWORD = <password>
```

II. Without credentials

In the `<repo>/.flaskenv` file, make sure to write those lines:

```
MAIL_USERNAME = <leave empty>
MAIL_PASSWORD = <leave empty>
MAIL_DISABLE = true
```

6. Launch

Almost there ! Once you proceeded to the above installation steps, you can now run the following command in your terminal to run the project. Make sure that you are still within the virtual environment.

```
$ flask run
```

Note: It is recommended to copy the above command lines (i.e., virtual environment activation, database initialization, redis-server startup, flask launch) in an executable script in order to facilitate the project launch for the next times. You can check the 'docker/entrypoint.sh' file for some inspiration.

Your terminal should now display a page url that you can access with your browser. This is your local version of the app!

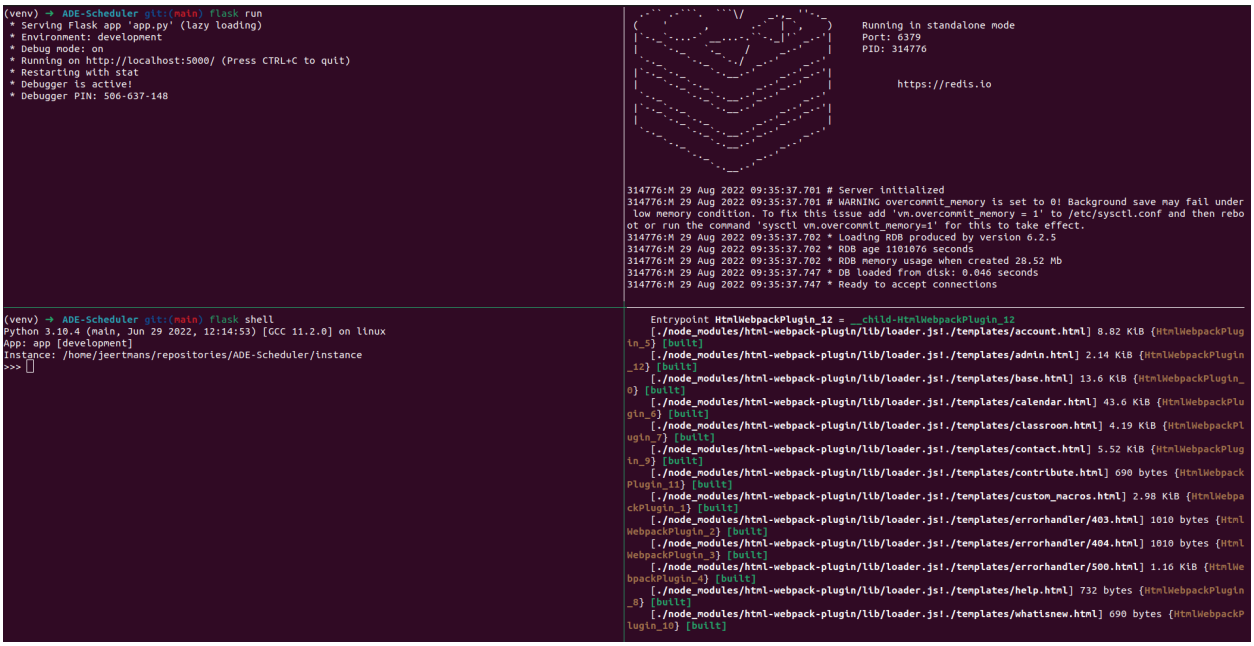

Fig. 1: Example of terminal configuration when everything is running properly. Here, you can see four terminal sessions: flask server (upper left), redis server (upper right), flask shell (lower left), and webpack (lower right). The "(venv)" indicates that the virtual environment is activated (required for left terminals only).

7. Contributing

Note: In order to keep the code as readable as possible, some guidelines are followed. Even though our coding style can never be perfect, it is good practice to follow a common syntax for all the project, as it will help newcomers getting easily into the project, as well debugging.

I. Install pre-commit

Please follow the tutorial on the official site: https://pre-commit.com/

Once it's done:

```
$ cd <repo>
$ pre-commit install
```

II. Create a branch

If you plan to contribute, it is always better to work on a separate branch with a meaningful name. To do so, in `<repo>`:

```
$ git branch -b your-branch-name
```

1.2.2 Other tutorials

ADE API tutorials

Note: The ADE API is the interface allowing ADE Scheduler to obtain all the information it needs, such as events, courses, etc., from the ADE server at UCLouvain.

Table of content

- *ADE API tutorials*
  - *ADE API setup*
    - *I. With credentials*
    - *II. Without credentials*

ADE API setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need for the calendar to work properly. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

Warning: Make sure to never commit your `<repo>/.flaskenv` file on Github! It should stay secret and the `<repo>/.gitignore` file should prevent you from doing that.

I. With credentials

If you are one of the maintainers, you should have received credentials in order to proceed requests to the ADE API. If you do not have such credentials, please read next section.

In the `<repo>/.flaskenv` file, make sure to write those lines and complete with your credentials:

```
ADE_URL = <url to get token>
ADE_DATA = <grant_type>
ADE_AUTHORIZATION = <authorization access>
```

II. Without credentials

Because we cannot share the credentials with everyone, but we want people to be able to have the best environment to contribute to ADE Scheduler, we created a fake API. To use this fake API, you will need to modify the `<repo>/.flaskenv`

```
ADE_URL = <leave empty>
ADE_DATA = <leave empty>
ADE_AUTHORIZATION = <leave empty>

ADE_FAKE_API = true
```

Next, you need to download a compressed folder containing the fake api as a folder containing pickled requests and a README file explaining everything that you can request with this API. The `fake_api` folder should be placed in the `<repo>` folder. Please contact us if something isn't working as expected.

Note: We know it is not the best way to provide you an access to the ADE API but we are still working on this and propositions are more than welcome!

ADE Scheduler tutorials

Note: This section will describe every tools that is brought with the ADE Scheduler repository. While many tools can be installed separately, the Github of this project contains many tools and things to setup that can be done in a few clicks.

Table of content

- *ADE Scheduler tutorials*
  - *ADE Scheduler setup*
    - *New to Github ?*
    - *I. Clone the repository*
    - *II. Install NodeJS and NPM*
    - *III. Install Python*
    - *IV. Create a virtual environment*
    - *V. Initialize the environment file*
  - *1. Flask client*
  - *2. Flask shell*
  - *3. Adding new Python package*
  - *4. Adding new Javascript package*

ADE Scheduler setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need for the project. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

New to Github ?

If you are new to Github, please consider reading this tutorial first: https://guides.github.com/activities/hello-world/

I. Clone the repository

If you are not one of the official maintainers of this project, you should first fork this repository: https://github.com/ADE-Scheduler/ADE-Scheduler/fork

Then, clone the forked repository somewhere on your machine. We will always refer to the location of the project directory as `<repo>`.

II. Install NodeJS and NPM

First, you need NodeJS and NPM (Node Package Manager). It can be downloaded on the official site: https://nodejs.org/en/

Once it is installed, go to the `<repo>` directory and run:

```
$ npm install
```

This will install all the Javascript packages we use in the project.

Any javascript or HTML asset needs to be bundled by webpack before being used. To do so, run in `<repo>`:

```
$ npx webpack
```

Moreover, you can activate the file watcher for hot-reloading and map source files for easy debugging by running:

```
$  npx webpack --watch --devtool inline-source-map
```

If an error appears about the maximum amount of allowed watchers, you can fix this by running:

Listing 8: from https://stackoverflow.com/questions/53930305/nodemon-error-system-limit-for-number-of-file-watchers-reached

```
$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf \
&& sudo sysctl -p
```

III. Install Python

The project can work with a variety of Python versions, but we recommend Python 3.9.X as this is the version that is currently used in production. It can be downloaded from the official site: https://www.python.org/

Before installing Python packages: Some external packages require external dependencies, and missing them will cause errors during the installation process. Here are a few: - OpenSSL: https://www.poftut.com/install-use-openssl-library-python-applications/ - OpenLDAP: https://stackoverflow.com/questions/4768446/i-cant-install-python-ldap

IV. Create a virtual environment

Event though this step is not mandatory, it is good practice and highly recommended to create a Python virtual environment. This can be done in a few lines:

```
$ cd <repo>
$ sudo apt install python3-virtualenv
or equivalent command if you are not on ubuntu
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r prod-requirement.txt # For production  (if you do not plan on modifying␣
↪code)
$ pip install -r dev-requirements.txt # For development (else)
```

> **Warning:** Whenever you want to run the project, you need to be in this virtual environment. You will then need to activate the environment each time you come back to work. Some IDEs such as PyCharm provide tools to automatically activate virtual environments.

V. Initialize the environment file

In this project, we use many configuration files. One of these files will carry sensible information such as password and it is your duty to create this file and to keep it secret.

Create a file named `.flaskenv` in the `<repo>` directory and fill the following lines in:

Listing 9: `<repo>/.flaskenv` content for development

```
FLASK_APP = path/to/<repo>/app.py
FLASK_ENV = development
FLASK_RUN_HOST = localhost
FLASK_RUN_PORT = 5000
TEMPLATES_AUTO_RELOAD = True
```

(continues on next page)

(continued from previous page)

```
FLASK_SECRET_KEY = <super_secret_key>
FLASK_SALT = <super_complex_salt>
```

Listing 10: `<repo>/.flaskenv` content for production

```
FLASK_APP = path/to/<repo>/app.py
FLASK_ENV = production

FLASK_SECRET_KEY = <super_secret_key>
FLASK_SALT = <super_complex_salt>
```

Lines will be added to this file in other tutorials.

1. Flask client

This project comes with a variety of command line tools. You can list all the available commands with:

```
$ flask --help
```

2. Flask shell

Among all the command line tools, there is the Flask shell. This interactive shell enters a Python interactive shell with all the context of the ADE Scheduler application. It is a great tool for debugging purposes!

```
$ flask shell
```

3. Adding new Python package

Whenever you add a package to your Python environment that is required for the project, you will need to add it to the list of requirements so that other developers will know it. This can be done pretty easily with *pipreqs* utility:

```
$ pip3 install some_package
# Include this packages in one or multiple Python files
$ pipreqs --save-path prod-requirements.txt # For anything in app.py / backend / views
# or
$ pipreqs --save-path docs/requirements.txt # For anything related for automated docs
# or
$ pipreqs --save-path dev-requirements.txt  # For anything else: tests, linting, etc.
```

Warning: For the moment, *pipreqs* is not the best solution and might not detect every package that is required. To this end, please always manually check changes in the requirement files. Only keep changes related to your new package.

4. Adding new Javascript package

As of npm 5.0.0, packages are automatically added to `<repo>/packages.json` when you install them. So, simply do this:

```
$ npm install some_package
```

Contribution tutorials

Note: In order to keep the code as readable as possible, some guidelines are followed. Even though our coding style can never be perfect, it is good practice to follow a common syntax for all the project, as it will help newcomers getting easily into the project, as well debugging.

Table of content

- *Contribution tutorials*
  - *Contribution setup*
    - *I. Install pre-commit*
    - *II. Create a branch*
  - *1. Running and writing tests*
  - *2. Syntax linting with Black*
  - *3. Committing changes*
  - *4. Translations*
    - *4.1 Adding translations*
      - *4.1.a In Python files*
      - *4.1.b In HTML and Javascript files*
    - *4.2 Adding a new language*
  - *5. Documentation*
    - *5.1 Coding style*
    - *5.2 Building the docs*
    - *5.3 Additional readings*
  - *6. Profiling the Flask Application*
    - *6.1 Setup*
    - *6.2 Reading stats files*

Contribution setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need before contributing. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

I. Install pre-commit

Please follow the tutorial on the official site: https://pre-commit.com/

Once it's done:

```
$ cd <repo>
$ pre-commit install
```

II. Create a branch

If you plan to contribute, it is always better to work on a separate branch with a meaningful name. To do so, in `<repo>`:

```
$ git branch -b your-branch-name
```

1. Running and writing tests

At ADE Scheduler, we think that Continuous Integration is important and it is why we are using unit tests. While any pull request must be validated by an automated CI check, we highly recommend you to run the tests in your local machine if you made change in the code. In `<repo>`, simply run:

```
$ pytest
```

Whenever you modify or create a Python function, make sure there are sufficient unit tests in the `<repo>/tests/` folder to ensure its correctness.

2. Syntax linting with Black

In order to keep our code clean and uniform, we use Black. For the moment, you do not really to care about how to use it as it is automated with *3. Committing changes*. If, by any mean, pre-commit is not working, please always run Black tool in `<repo>` before committing changes:

```
$ black .
```

3. Committing changes

Before committing changes, you need to run in <repo>:

```
$ pre-commit run --all-files
```

4. Translations

Down below, we listed the most important things to know when using translation in this project. You can learn many others from the Flask-Megatutorial about babel.

4.1 Adding translations

If you add text which will be displayed on the website, you need to provide translations. The language used in the code is always English.

Once the code is written, you need to extract the strings to translate:

```
$ cd <repo>
$ pybabel extract -F translations/babel.cfg -k _l -o translations/messages.pot .
$ pybabel update -i translations/messages.pot -d translations
```

Then, manually enter the translations in the various `messages.po` files located in `<repo>/translations/<language>/LC_MESSAGES/`. When it's done, you can compile the new translations from <repo>:

```
$ pybabel compile -d translations
```

4.1.a In Python files

When you want a string to be translated, you need to do two things:

```
from flask_babel import gettext

x = gettext("string_to_translate")
```

4.1.b In HTML and Javascript files

Here, you only need to embed the string with mustaches:

```
{{ _('string_to_translate') }}
```

4.2 Adding a new language

You can easily add a new language (here *fr*) by executing this command from <repo>:

```
$ pybabel init -i translations/messages.pot -d translations -l fr
```

5. Documentation

5.1 Coding style

If you plan to contribute to ADE Scheduler, thank you! We really appreciate any help, from anyone. You code will never be rejected because of its coding style but, if you can follow the PEP8 guidelines, it will save us time!

Most PEP guidelines can be followed if you use the Black tool mentioned before.

5.2 Building the docs

This whole documentation is actually built using the Sphinx tool. If you happen to make modifications in the documentation, please try building the documentation before making a pull request.

```
$ cd <repo>/docs
$ pip install -r requirements.txt # Only run this once
$ sphinx-apidoc -o source/backend ../backend -f
$ sphinx-apidoc -o source/views ../views -f
$ make html
```

Then, simply open `build/html/index.html` in any browser to access the documentation.

5.3 Additional readings

Here are listed some interesting readings about Sphinx's documentation tool:

- https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html
- https://thomas-cokelaer.info/tutorials/sphinx/docstring_python.html
- https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html

6. Profiling the Flask Application

Performance is a crucial aspect in ADE Scheduler as it will impact users' experience. This section discusses how to profile the Flask Application to better understand some performance issues you might encounter.

6.1 Setup

Profiling is straighforward: you just need to add one line to your `.flaskenv` file.

Listing 11: `<repo>/.flaskenv`

```
PROFILE = True
```

And voilà! Once the application is (re-)started, it will now output the time taken by each request. It will also write the profile stats into files under the `profile` folder.

If you wish the tweak the profiling settings, you can modify the `app.py` file. More information can be found here.

6.2 Reading stats files

Interpreting `.prof` files can be hard and we recommend you to use a graphical tool to do so. There are plenty available for free and we use tuna. With this tool, it is straighforward to analyse the profiling stats.

```
$ cd <repo>
$ tuna profile/<path_to_file.prof>
```

Database tutorials

Note: For permanent or long-term storage, a SQL database is used. While many implementations exist, we choose to use PostgreSQL for its robustness against high activity. SQLite has proven to be a bottleneck in performances for ADE Scheduler (in production) but can still be used for development.

Table of content

- *Database tutorials*
  - *PostgreSQL setup*
    - *I. Lite setup*
      - *I-a. Install SQLite3*
      - *I-b. Create a database*
    - *II. Complete setup*
      - *II-a. Install PostgreSQL*
      - *II-b. Setup a password*
      - *II-c. Create a database*
      - *II-d. Setup read without password access*
    - *III. Link database in .flaskenv*
    - *IV. Populate the database*
  - *1. Recovering data from backup version*
    - *1.1 Reading the database on your local machine*

- *2. SQL client*
- *3. Modify the database structure*
- *4. Update or migrate the database*

PostgreSQL setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need for database manipulations. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

I. Lite setup

Note: If it is your first time working on this project, we highly recommend you to follow this setup. The complete setup is only required for maintainers working with the database hosted on the ADE Scheduler server.

I-a. Install SQLite3

You will need to install SQLite3 to perform operation on your database. To do so, refer to instructions related to your machine. For ubuntu:

```
$ sudo apt-get install sqlite3
```

I-b. Create a database

You will need to create an empty database, wherever you want, on your machine. Just create an empty file name `ade-database.db`.

Please, remember the location of the database, as you will need it for later in the tutorial. Now, you can directly jump to section *III*.

II. Complete setup

II-a. Install PostgreSQL

For this, we highly recommend you to follow this guide: https://www.postgresqltutorial.com/install-postgresql/

II-b. Setup a password

Even if it is not always required, it is good practice to setup a password for the `postgres` user:

Listing 12: Tutorial from: https://docs.boundlessgeo.com/suite/1.1.1/dataadmin/pgGettingStarted/firstconnect.html

```
$ sudo -u postgres psql postgres
$ \password
enter your password_psql and confirm
$ \q
```

II-c. Create a database

In order to manipulate databases, you need to create a database instance:

```
$ sudo su - postgres
$ createdb ade-database
$ exit
```

II-d. Setup read without password access

By default, ADE Scheduler tries to access the database without password. Here, we need to explicitly allow the program to access the newly created database without password:

Listing 13: You main need to replace *12* with your actual version if it differs

```
$ sudo {vim|geany|nano|...} /etc/postgresql/12/main/pg_hba.conf
and change `peer`/`md5` values to `trust`
$ sudo systemctl restart postgresql
```

III. Link database in .flaskenv

Now, you will need to tell the program where your database is located. To do so, add this line in your `<repo>/.flaskenv` file:

```
$ ADE_DB_PATH="sqlite:///<path/to>/ade_database.db"
for SQLite3 (warning, <path/to> may be "/home/..." so it will add 1 more "/")
$ ADE_DB_PATH="postgresql://postgres@localhost/ade-database"
for PostgreSQL
or, alternatively, you can use an other database you have
$ ADE_DB_PATH=<database URI>
```

IV. Populate the database

If your database is empty, you need to populate it with the correct tables and columns. This can be done using the client:

```
$ flask sql init
```

1. Recovering data from backup version

In order to provide robustness to ADE Scheduler's users, a backup of the database is done once in a while. This tutorial will show you how to use a precedent version of the database, in order to do some statistic about usage or to recover an old version of the database in case of problem.

1.1 Reading the database on your local machine

Here, we suppose that you have a copy of backup version of the database. Such copy can be obtained using `scp` command to transfer a backup version from the server to your local machine.

```
drop old database
$ psql -c 'drop database "ade-database";' -U postgres --host=localhost
create new database
$ psql -c 'create database "ade-database";' -U postgres --host=localhost
$ gzip -d {db-backup}.sql.gz
un-zip database
$ psql -U postgres --host=localhost --dbname=ade-database < {db-backup}.sql
un-dump database
eventually enter you password_psql
```

2. SQL client

Once the project is setup, you can use the various commands in the shell in order to interact with the database:

```
$ flask db --help
$ flask sql --help
```

3. Modify the database structure

4. Update or migrate the database

Email tutorials

Note: In order to quickly receive feedback if anything goes wrong, ADE Scheduler will send you an email everytime an error occurs. This can be, of course, disabled to avoid email spamming. This email address can also be used to send account confirmation emails.

Table of content

- *Email tutorials*
  - *Email setup*
    - *I. With credentials*
    - *II. Without credentials*
  - *1. Disable error emails*

Email setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need for the feedback system to work properly. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

Warning: Make sure to never commit your `<repo>/.flaskenv` file on Github! It should stay secret and the `<repo>/.gitignore` file should prevent you from doing that.

I. With credentials

Here, you will allow ADE Scheduler to send emails using your email address. You may need to allow third-party applications to let ADE Schedule use your email address. Please refer to your email provider documentation if so. If you do not want to use the emails, please read next section.

In the `<repo>/.flaskenv` file, make sure to write those lines and complete with your credentials:

```
MAIL_USERNAME = <email address>
MAIL_PASSWORD = <password>
```

II. Without credentials

In the `<repo>/.flaskenv` file, make sure to write those lines:

```
MAIL_USERNAME = <leave empty>
MAIL_PASSWORD = <leave empty>
MAIL_DISABLE = true
```

1. Disable error emails

In the <repo>/.flaskenv file, add this line:

```
MAIL_SEND_ERRORS = false
```

Redis server tutorials

Note: A Redis server is used as a cache memory storage and will, among others, act as a buffer between the ADE API and ADE Scheduler in order to enhance performance and to reduce delay between repetitive requests.

Table of content

- *Redis server tutorials*
  - *Redis setup*
    - *I. Install Redis*
    - *II. Check that it works*
  - *1. Redis clients*
    - *1.1 Built-in client*
    - *1.2 Flask client*

Redis setup

Note: Before running into the other tutorials, you should go through this one in order to setup all you need for the redis server manipulations. As ADE Scheduler is hosted on a UNIX system machine, this tutorial will use the same commands as UNIX / LINUX system machines. No current support is provided for Windows users.

I. Install Redis

The installation process is pretty simple and we recommend you to follow the official guide: https://redis.io/topics/quickstart

II. Check that it works

Before running into more troubles, make sure that Redis is correctly installed and that you can manually start a server from command line. Something like this should work:

Listing 14: Start this in one terminal

```
$ redis-server
do not kill this terminal
```

Listing 15: Then, do this in another terminal

```
$ redis-cli ping
you should receive `pong`
$ redis-cli shutdown
this kills the redis server
```

1. Redis clients

There are 2 clients for the Redis server:

1.1 Built-in client

The built-in client is provided with Redis installation. Its purpose is to provide tools to modify or to read the content of the server. The documentation of this client can be found online: https://redis.io/documentation

Listing 16: Invoking the Redis client (a Redis server must be running!)

```
$ redis-cli
```

1.2 Flask client

The Flask client is defined by the *cli* module and its documentation can be found in the appropriate section.

```
$ flask redis --help
```

1.2.3 backend

ade_api module

class ade_api.**Client**(*credentials: Dict[str, Union[str, List[int]]]*)

Bases: *ade_api.DummyClient*

A client subclasses the DummyClient abstract class and implements access to ADE API using correct credentials. If you do not own such credentials, please use *FakeClient()*.

expire_in() → float

Returns the time remaining before the token expires.

Returns the time remaining in seconds, 0 if expired

Return type float

is_expired() → bool

Returns whether the current token is expired.

Returns True if the token is expired

Return type bool

renew_token()

Renews the current token requesting a new one.

request(***kwargs: Union[str, int]*) → requests.models.Response

Performs a request to the API with given parameters. The client should automatically renew its token if it has expired.

Parameters **kwargs** (*Request*) – set of key / value parameters that will be merged

Returns the response

Return type request.Response

Raises **HTTPError** – if the request is unsuccessful (oustide the 2XX-3XX range)

class ade_api.**DummyClient**(*credentials: Dict[str, Union[str, List[int]]]*)

Bases: object

A dummy client is an abstrat class providing just the specific requests, but not the access to the API. Access to the API is to be implemented.

Parameters

- **credentials** – all information needed to make requests to API
- **credentials** – ClientCredentials

Example

```
>>> import backend.credentials as credentials
>>> credentials = credentials.get_credentials(credentials.ADE_API_CREDENTIALS)
>>> client = Client(credentials)
```

expire_in() → float

Returns the time remaining before the token expires.

Returns the time remaining in seconds, 0 if expired

Return type float

get_activities(*resource_ids: List[str], project_id: str*) → requests.models.Response

Requests all activities (set of events) for a specific project.

Parameters

- **resource_ids** (*List[str]*) – the ids of all the resources the activities are requested
- **project_id** (*str*) – the id of the project

Returns the response

Return type request.Response

get_classrooms(*project_id: str*) → requests.models.Response

Requests all the classrooms for a specific project.

Parameters **project_id** (*str*) – the id of the project

Returns the response

Return type request.Response

get_course_resources(*project_id: str*) → Tuple[requests.models.Response, requests.models.Response]

Requests all the course resource for a specific project.

Parameters **project_id** (*str*) – the id of the project

Returns a tuple the response

Return type Tuple[request.Response, requests.Response]

get_project_ids() → requests.models.Response

Requests the project ids currently available; each year (project) corresponds to an id.

Returns the response

Return type request.Response

get_resource_ids(*project_id: str*) → requests.models.Response

Requests the ids of all the resource for a specific project.

Parameters **project_id** (*str*) – the id of the project

Returns the response

Return type request.Response

get_resources(*project_id: str*) → requests.models.Response

Requests all the resource for a specific project.

Parameters **project_id** (*str*) – the id of the project

Returns the response

Return type request.Response

is_expired() → bool
Returns whether the current token is expired.

Returns True if the token is expired

Return type bool

renew_token() → None
Renews the current token requesting a new one.

request(***kwargs: Union[str, int]*) → requests.models.Response
Performs a request to the API with given parameters. The client should automatically renew its token if it has expired.

Parameters **kwargs** (*Request*) – set of key / value parameters that will be merged

Returns the response

Return type request.Response

Raises **HTTPError** – if the request is unsuccessful (oustide the 2XX-3XX range)

exception ade_api.**ExpiredTokenError**
Bases: Exception

Exception that will occur if a token is expired.

class ade_api.**FakeClient**(*credentials: Dict[str, Union[str, List[int]]]*)
Bases: *ade_api.DummyClient*

A fake client subclasses the DummyClient abstract class and implements a fake access to ADE API. All the API responses you can cannot are stored in json file, which you can change if you want to.

FOLDER = 'fake_api'

expire_in() → float
Returns the time remaining before the token expires.

Returns the time remaining in seconds, 0 if expired

Return type float

is_expired() → bool
Returns whether the current token is expired.

Returns True if the token is expired

Return type bool

renew_token() → None
Renews the current token requesting a new one.

request(***kwargs: Union[str, int]*) → requests.models.Response
Performs a request to the API with given parameters. The client should automatically renew its token if it has expired.

Parameters **kwargs** (*Request*) – set of key / value parameters that will be merged

Returns the response

Return type request.Response

Raises **HTTPError** – if the request is unsuccessful (oustide the 2XX-3XX range)

ade_api.**get_response_path**(*request_name: str*) → str
Returns the path where the response should be located if using the fake ade api.

Parameters **request_name** (*str*) – the parameters used in the request, concatenated as in *FakeClient.request()*

Returns the path

Return type str

ade_api.**get_token**(*credentials: Dict[str, Union[str, List[int]]]*) → Tuple[str, int]

Requests a new token to the API.

Parameters

- **credentials** – all information needed to make requests to API
- **credentials** – ClientCredentials

Returns the token and its time to expiration in seconds

Return type Tuple[str, int]

ade_api.**load_response**(*request_name: str*) → requests.models.Response

Loads the response of a given request.

Parameters **request_name** (*str*) – the parameters used in the request, concatenated as in *FakeClient.request()*

Returns the response

Return type requests.Response

Raises **HTTPError** – if the response to the request could not be found in the file

ade_api.**parse_activity**(*activity: lxml.etree._Element*) → Tuple[List[backend.events.AcademicalEvent], str, str, str]

Parses an element from a request into a list of events and some activity information.

Parameters **activity** (*etree._Element*) – the activity element

Returns the events, the name, the id and the code of this activity

Return type Tuple[List[backend.events.AcademicalEvent], str, str, str]

ade_api.**parse_event**(*event: lxml.etree._Element*, *event_type: backend.events.AcademicalEvent*, *activity_name: str*, *activity_id: str*, *activity_code: str*) → backend.events.AcademicalEvent

Parses an element from a request into an academical event. An event is from an activity so information about this activity must be provided.

Parameters

- **event** (*etree._Element*) – the event element
- **event_type** (*Type[backend.events.AcademicalEvent]*) – the constructor used to initiate to event object
- **activity_name** (*str*) – the name of the activity
- **activity_id** (*str*) – the id of the activity
- **activity_code** (*str*) – the code of the activity

Returns the academical event

Return type backend.events.AcademicalEvent

ade_api.**response_to_classrooms**(*classrooms_response: requests.models.Response*) → List[backend.classrooms.Classroom]

Extracts an API response into list of classrooms.

Parameters **classrooms_response** (*requests.Response*) – a response from the API to the classrooms request

Returns all classrooms

Return type List[*Classroom*]

Example

```
>>> response = client.get_classrooms(9)  # project id for 2019-2020
>>> classrooms = response_to_classrooms(response)
```

ade_api.**response_to_course_resources**(*course_resources_response: Tuple[requests.models.Response, requests.models.Response]*) → pandas.core.frame.DataFrame

Extracts an API response into an dataframe containing all course resources.

Parameters **course_resources_response** (*Tuple[requests.Response, requests.Response]*) – a response from the API to the course resources request

Returns all the course resources

Return type pd.Dataframe

ade_api.**response_to_courses**(*activities_response: requests.models.Response*) → List[backend.courses.Course]

Extracts an API response into list of courses.

Parameters **activities_response** (*requests.Response*) – a response from the API to the activities request

Returns all courses present in the response

Return type List[Courses]

Example

```
>>> response = client.get_activities(['1234'], 9)  # project id for 2019-2020
>>> courses = response_to_courses(response)
```

ade_api.**response_to_events**(*activities_response: requests.models.Response, filter_func: Callable[[backend.events.AcademicalEvent], bool]*) → List[backend.events.AcademicalEvent]

Extracts an API response into list of events.

Parameters

- **activities_response** (*requests.Response*) – a response from the API to the activities request
- **filter_func** (*Callable[[backend.events.AcademicalEvent], bool]*) – a function to filter out events

Returns all events present in the response, optionnally filtered

Return type List[backend.events.AcademicalEvents]

Example

```
>>> response = client.get_activities(['1234'], 9)  # project id for 2019-2020
>>> events = response_to_events(response)
```

ade_api.**response_to_project_ids**(*project_ids_response: requests.models.Response*) → Dict[str, str]

Extracts an API response into an iterator of project ids and years.

Parameters **project_ids_response** (*requests.Response*) – a response from the API to the project_ids request

Returns all the project ids and years (year as key, id as value)

Return type Dict[str, int]

Example

```
>>> response = client.get_project_ids()
>>> ids_years = response_to_project_ids(response)
```

ade_api.**response_to_resource_ids**(*resource_ids_response*) → Dict[str, str]

Extracts an API response into an dictionary mapping a resource name to its ids.

Parameters **resource_ids_response** (*requests.Response*) – a response from the API to the resources or resource_ids request

Returns all the resources names and their ids

Return type Dict[str, str]

Example

```
>>> response = client.get_resource_ids(9)  # project id for 2019-2020
>>> resources_ids = response_to_resource_ids(response)
```

ade_api.**response_to_resources**(*resources_response: requests.models.Response*) → pandas.core.frame.DataFrame

Extracts an API response into an dataframe containing all resources.

Parameters **resources_response** (*requests.Response*) – a response from the API to the resources request

Returns all the resources

Return type pd.Dataframe

Example

```
>>> response = client.get_resources(9)  # project id for 2019-2020
>>> resources = response_to_resources(response)
```

ade_api.**response_to_root**(*response: requests.models.Response*) → lxml.etree._Element

Parses an API response into a tree structure.

Parameters **response** (*requests.Response*) – a response from the API

Returns the tree structure

Return type etree._ElementTree

ade_api.**room_to_classroom**(*room: lxml.etree._Element*) → backend.classrooms.Classroom

Parses the (class)room retrieved from API to a more convenient Classroom object.

Parameters **room** (*etree._Element*) – (class)room as a tree structure

Returns the classroom

Return type *Classroom*

ade_api.**save_response**(*response: requests.models.Response*, *request_name: str*)

Saves the response to a request into the file specified by the FakeClient class.

Parameters

- **response** (*requests.Response*) – the response to the request
- **request_name** (*str*) – the parameters used in the request, concatenated as in *FakeClient.request()*

classrooms module

class classrooms.**Address**(***kwargs: str*)

Bases: object

Address object made from informations retrieved via ADE API.

Parameters **kwargs** (*str*) – dict with minimal entries (can be None): - address1 - zipCode - city - country

Example

```
>>> informations = dict(address1='Rue Rose 42', zipCode='1300', city='Wavre',␣
↪country='Belgique')
>>> address = Address(**informations)
```

class classrooms.**Classroom**(***kwargs: Union[str,* classrooms.Address*]*)

Bases: object

Classroom object containing the address (as string or Address object), the name of the classroom and its id.

Parameters **kwargs** (*Union[str, Address]*) – dict with minimal entries: - address - name - id

Its main purpose is be used with *location()*.

location() → str

Returns the location (address) of this classroom.

Returns the location

Return type str

classrooms.**get_geo_locations**() → Dict

Returns the dictionary mapping each address to its geo-location.

Returns the geo-locations

Return type Dict

classrooms.**merge_classrooms**(*classrooms: Iterable[*classrooms.Classroom*]*) → *classrooms.Classroom*

Merges multiple classrooms into one.

Parameters **classrooms** (*Iterable[Classroom]*) – multiple classrooms

Returns the new classroom

Return type *Classroom*

Example

```
>>> c1 = Classroom(address1, 'classA', 1)
>>> c2 = Classroom(address2, 'classB', 2)
>>> c3 = merge_classrooms((c1, c2))
```

classrooms.**prettify_classrooms**(*classrooms: pandas.core.frame.DataFrame, sleep: float = 0*) → pandas.core.frame.DataFrame

Returns the classrooms dataframe in a pretty format, useful when need to display.

The function will request, for every different address, a geo-localisation, so it can take some times. If too many requests are done, Nominatim will not like it so prefer to put a time.sleep between each request.

Parameters

- **classrooms** (*pd.DataFrame*) – the classrooms with fields defined in backend.resources.py
- **sleep** (*float*) – the sleep duration between each address request

Returns the classrooms in a prettier format and with geo-location information

Return type pd.DataFrame

`classrooms.save_geo_locations(geo_locations: Dict)`

Saves the dictionary of geo-locations into a json file with pretty indent. This file can be manually edited without causing any problem.

Parameters **geo_locations** (*Dict*) – the geo-locations

cookies module

Provides utils to store and retrieve cookies, as well as a secure mechanism to score oauth tokens in cookies.

`cookies.get_oauth_token() → Optional[dict]`

`cookies.set_cookie(key: str, value: str, resp, **kwargs: Any)`

`cookies.set_oauth_token(token: dict, resp)`

courses module

`class courses.Course(code: str, name: str, weight: float = 1, activities: Optional[pandas.core.frame.DataFrame] = None)`

Bases: `object`

A course aims to represent one or more courses. It contains its events and is represented with a name and a code.

Parameters

- **code** (*str*) – the code of the course
- **name** (*str*) – the full name of the course
- **weight** (*float*) – the weight attributed to the course
- **activities** (*Optional[pd.Dataframe]*) – a structure of all the events indexed by code, type and id

Example

```
>>> course = Course('LMECA2732', 'Robotics')
```

`add_activity(events: List[backend.events.AcademicalEvent])`

Adds an activity to the current course's activities. An activity is a set of events with the same id.

Parameters **events** (*List[AcademicalEvent]*) – list of academical events coming from the same activity

`get_activities(view: Optional[Union[List[str], Set[str], Dict[int, str]]] = None, reverse: bool = False) → pandas.core.frame.DataFrame`

Returns a table of all activities that optionally match correct ids.

Parameters

- **view** (*Optional[View]*) – if present, list of ids or dict {week_number : ids}
- **reverse** (*bool*) – if True, the activities in View will be removed

Returns table containing all the activities and their events

Return type pd.DataFrame

get_events(***kwargs*) → Iterable[backend.events.AcademicalEvent]

Returns a list of events that optionally matches correct ids.

Parameters **kwargs** (*Any*) – parameters that will be passed to *Course.get_activities()*

Returns list of events

Return type Iterable[*AcademicalEvent*]

get_summary() → Dict[str, Set[str]]

Returns the summary of all activities in the course.

Returns dict of activity codes, ordered by activity type (CM, TP, etc.)

Return type Dict[str, Set[str]]

set_weights(*percentage: float = 50, event_type: Optional[backend.events.AcademicalEvent] = None*)

Modifies this course's events weight.

Parameters

- **percentage** (*float*) – the "priority" required for this course in (0-100)%, default is 50%
- **event_type** (*Optional[AcademicalEvent]*) – if present, modify the weight of a certain type of event only

courses.**generate_empty_dataframe**()

courses.**merge_courses**(*courses: Iterable[courses.Course], code: str = '0000', name: str = 'merged', weight: float = 1, views: Optional[Dict[str, Union[List[str], Set[str], Dict[int, str]]]] = None, **kwargs: Any*) → *courses.Course*

Merges multiple courses into one.

Parameters

- **courses** (*Iterable[Courses]*) – multiple courses
- **code** (*str*) – the new code
- **name** (*str*) – the new name
- **weight** (*float*) – the new weight
- **views** (*Optional[Dict[str, View]]*) – map of views that will be passed to *Course.get_activities()*
- **kwargs** (*Any*) – additional parameters that will be passed to *Course.get_activities()*

Returns the new course

Return type *Course*

events module

class events.**AcademicalEvent**(*name: str*, *begin: datetime.datetime*, *end: datetime.datetime*, *professor: backend.professors.Professor*, *classrooms: Optional[Iterable[backend.classrooms.Classroom]] = None*, *id: Optional[str] = None*, *weight: Union[int, float] = 5*, *code: Optional[str] = None*, *prefix: Optional[str] = None*, *note: Optional[str] = None*)

Bases: *events.CustomEvent*

An academical event is an object used to represent any event in the academical calendar.

It subclasses CustomEvent.

Parameters

- **name** (*str*) – the name of the event
- **begin** (*datetime*) – the start of the event
- **end** (*datetime*) – the end of the event
- **professor** (Professor) – the professor(s) in charge of this event
- **classrooms** (*Optional[List[Classroom]]*) – all the classrooms were this event takes place
- **id** (*Optional[str]*) – the id of the event
- **weight** (*Union[int, float]*) – the weight attributed to the event
- **code** (*Optional[str]*) – code of the course related to this event
- **prefix** (*Optional[str]*) – the prefix used for to describe the type of event
- **note** – a note to be added to the event description

KEYWORDS = ()

get_id() → str

Returns the id of this event.

Returns the id of the event

Return type str

json(*color=''*)

Returns the event as a json-like format.

Parameters **color** (*str*) – the color of the event

Returns a dictionary containing relevant information

Return type Dict[str, Any]

classmethod matches(*string*)

class events.**CustomEvent**(*weight: Union[int, float] = 5*, ***kwargs: Any*)

Bases: ics.event.Event

Subclass of ics.Event, implementing more methods useful to know if two events are conflicting.

Parameters

- **weight** (*Union[int, float]*) – the weight of this event
- **kwargs** (*Any*) – parameters passed to ics.Event() constructor, but should as least contain : - name: str - begin: datetime - end: datetime - location: str - description: str

DEFAULT_COLOR = '#9e742f'

get_week() → int

Returns the week of this event in the gregorian calendar, starting at 0 for the first week.

Returns the week number relative to gregorian calendar numbering

Return type int

intersects(*other:* events.CustomEvent) → bool

Returns whether two events intersect each other.

Parameters **other** (CustomEvent) – the event to compare with

Returns true if both events intersect

Return type bool

json(*color: Optional[str] = None*) → Dict[str, Any]

Returns the event as a json-like format.

Parameters **color** (*str*) – the color of the event

Returns a dictionary containing relevant information

Return type Dict[str, Any]

overlap(*other:* events.CustomEvent) → float

If both events intersect, returns the product of the weights.

Parameters **other** (CustomEvent) – the event to compare with

Returns self.weight * other.weight if intersect, else 0

Return type float

set_weight(*weight: float*)

changes the weight of the event.

Parameters **weight** (*float*) – the weight

class events.**EventCM**(***kwargs*)

Bases: *events.AcademicalEvent*

KEYWORDS = ('cm', 'cours', 'magistral')

PREFIX = 'CM: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.**EventEXAM**(**kwargs)

Bases: *events.AcademicalEvent*

KEYWORDS = ('partiel', 'ecrit', 'oral', 'interro', 'test')

PREFIX = 'EXAM: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.**EventEXTERN**(**kwargs)

Bases: *events.AcademicalEvent*

KEYWORDS = ('extern',)

PREFIX = 'EXT: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

classmethod from_event(*event: ics.event.Event*, *code: str*) → *events.EventEXTERN*

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.**EventLABO**(**kwargs)

Bases: *events.AcademicalEvent*

KEYWORDS = 'labo'

PREFIX = 'LABO: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.EventORAL(**kwargs)

Bases: events.AcademicalEvent

KEYWORDS = ('oral',)

PREFIX = 'ORAL: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.EventOTHER(**kwargs)

Bases: events.AcademicalEvent

KEYWORDS = ('autre', 'info', 'monitorat', 'consult', 'copie')

PREFIX = 'OTHER: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.**EventTP**(***kwargs*)

Bases: *events.AcademicalEvent*

KEYWORDS = ('tp', 'td', 'ape', 'app')

PREFIX = 'TP: '

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

class events.**RecurringCustomEvent**(*end_recurrence*, *freq*, ***kwargs*)

Bases: *events.CustomEvent*

Subclass of CustomEvent, representing a recurring event, according to iCalendar's rules.

Parameters

- **end_recurrence** (*datetime*) – the end of the recurrence
- **freq** (*Iterable[str]*) – the frequency of the recurrence
- **kwargs** (*Any*) – parameters passed to *CustomEvent()* constructor

DEFAULT_COLOR = '#8a7451'

alarms: List[BaseAlarm]

attendees: Set[Attendee]

categories: Set[str]

created: Optional[ArrowLike]

description: Optional[str]

json(*color=None*)

Returns the event as a json-like format.

Parameters **color** (*str*) – the color of the event

Returns a dictionary containing relevant information

Return type Dict[str, Any]

last_modified: Optional[ArrowLike]

location: Optional[str]

organizer: Optional[Organizer]

transparent: Optional[bool]

uid: str

url: Optional[str]

events.**extract_code**(*course_id: str*) → str

Extracts a code from a course id.

Parameters **course_id** (*str*) – str given by ADE API to represent the id

Returns The code of the course. None if nothing matched the pattern required

Return type str

events.**extract_datetime**(*date: str*, *start: str*, *end: str*) → Tuple[datetime.datetime, datetime.datetime]

Parses infos to return the start and end time of an event.

Parameters

- **date** (*str*) – the date matching %d/%m/%Y format
- **start** (*str*) – the start hour matching %H:%M format
- **end** (*str*) – the end hour matching %H:%M format

Returns datetime objects (start date, end date)

Return type Tuple[datetime, datetime]

events.**extract_type**(*course_type: str*, *course_id: str*) → Type[*events.AcademicalEvent*]

Extract the type of Academical event from course type or course id. Sometimes, information from ADE API is wrong…

Parameters

- **course_type** (*str*) – string given by ADE API to represent the type
- **course_id** (*str*) – string given by ADE API to represent the id

Returns the type of the event

Return type Type[*AcademicalEvent*]

events.**pretty_date_formatter**(*arrow*) → str

events.**pretty_formatter**(*arrow*) → str

events.**pretty_hour_formatter**(*arrow*) → str

events.**remove_accents**(*input_str: str*) → str

events.**remove_prefix**(*s: str*, *prefix: str*) → str

events.**sanitize_string**(*input_str: str*) → str

manager module

exception manager.**ExternalCalendarAlreadyExistsError**

Bases: Exception

Exception that will occur if someone tries to create a calendar with a code already taken.

class manager.**Manager**(*client: backend.ade_api.Client*, *server: backend.servers.Server*, *database: flask_sqlalchemy.SQLAlchemy*, *ttl: Dict*)

Bases: object

The manager ensures that data is accessible and provides access to it.

Data can either be found in the server, in the database or obtained from the ADE API. At initialization, each source is checked to ensure that they are working properly.

Parameters

- **client** (ade_api.Client) – the client providing access to ADE API
- **server** (*server.Server*) – the server providing temporary memory
- **database** (*md.SQLAlchemy*) – the database
- **ttl** (*Dict*) – a dictionary mapping keys in server to their default time-to-live value

code_exists(*code*, *project_id: Optional[str] = None*) → bool

Checks if a given code exists in the database for a given project id

delete_external_calendar(*id: int*)

get_classrooms(*project_id: Optional[str] = None*, *search_dict: Optional[Dict[str, str]] = None*, *return_json: bool = False*)

get_codes_matching(*pattern: str*, *project_id: Optional[str] = None*) → List[str]

get_course_resources(*project_id: Optional[str] = None*) → pandas.core.frame.DataFrame

Returns the course resources.

Parameters **project_id** (*str*) – the project id

Returns the courses resources

Return type pd.DataFrame

get_courses(**codes: str*, *project_id: Optional[str] = None*) → List[backend.courses.Course]

Returns the courses with given codes as a list. Order of courses is consistent with initial order of the codes.

Parameters

- **codes** (*str*) – the code(s) of the course(s)
- **project_id** (*str*) – the project id

Returns the list of courses

Return type List[crs.Course]

get_default_project_id() → str

Returns the default project id.

Returns the default project id

Return type str

get_events_in_classroom(*classroom_id: str*, *project_id: Optional[str] = None*) → List[backend.events.AcademicalEvent]

get_external_calendars(*user: backend.models.User*) → List[backend.models.ExternalCalendar]

get_link(*schedule_id*)

get_plots() → List[Tuple[str, dict]]

Returns all the (key, plot) pairs stored in the server. Plots are json dictionary generated using Plotly.

Returns the pairs of (key, plot) that were stored in the server

Return type List[Tuple[str, dict]]

get_project_ids(*year: Optional[str] = None*) → Optional[Union[List[Dict[str, str]], str]]

Returns the project ids. If year is specified, only the project id of this year is returned.

Parameters **year** (*str*) – the year, '2019-2020' format

Returns the list of ids and years or the id of one year or None if no id was found

Return type Union[List[Dict[str, str]], str, None]

get_resource_ids(**codes: str*, *project_id: Optional[str] = None*) → Iterator[str]

Returns the resource ids of each code.

Parameters

- **codes** (*str*) – the code(s) (name(s)) of the resources
- **project_id** (*str*) – the project id

Returns the resource ids

Return type Iterator[str]

get_resources(*project_id: Optional[str] = None*) → pandas.core.frame.DataFrame

Returns the resources.

Parameters **project_id** (*str*) – the project id

Returns the resources

Return type pd.DataFrame

get_schedule(*link*)

save_ics_url(*code: str*, *name: str*, *url: str*, *description: str*, *user: backend.models.User*, *approved: bool*)

save_schedule(*user: backend.models.User*, *schedule: backend.schedules.Schedule*, *uuid*)

Saves a schedule binding it to a user into the database.

Parameters

- **user** (*md.User*) – the user
- **schedule** (*schd.Schedule*) – the schedule

Returns the scheduler, with its id updated…

update_classrooms(*drop_empty: List[str] = ['address1']*)

Updates the classrooms contained in the server for all project ids.

update_course_resources()

Updates the course resources contained in the server for all project ids.

update_project_ids()

Updates the project ids.

update_resource_ids()

Updates the resource ids contained in the server for all project ids.

update_resources()
Updates the resources contained in the server for all project ids.

exception manager.**ScheduleNotFountError**
Bases: Exception

Exception that will occur if a schedule is marked as saved but is not in the database.

mixins module

class mixins.**AnonymousUser**
Bases: flask_login.mixins.AnonymousUserMixin

has_role(*role*)
Returns *False*.

class mixins.**UserMixin**
Bases: flask_login.mixins.UserMixin

add_role(*role*)
Add *role* to this user.

has_role(*role*)
Checks if the user has a given role. *role* can be a string or a Role instance

models module

class models.**ApiUsage**(*url*, *response*)
Bases: sqlalchemy.orm.decl_api.Model

datetime

id

speed

status

url

class models.**ExternalCalendar**(*code*, *name*, *url*, *description*, *user*, *approved=False*)
Bases: sqlalchemy.orm.decl_api.Model

approved

code

description

id

name

update_url(*url*)

url

user_id

class models.**GUID**(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Platform-independent GUID type.

Uses PostgreSQL's UUID type, otherwise uses CHAR(32), storing as stringified hex values.

impl
alias of `sqlalchemy.sql.sqltypes.CHAR`

load_dialect_impl(*dialect*)
Return a `TypeEngine` object corresponding to a dialect.

This is an end-user override hook that can be used to provide differing types depending on the given dialect. It is used by the `TypeDecorator` implementation of `type_engine()` to help determine what type should ultimately be returned for a given `TypeDecorator`.

By default returns `self.impl`.

process_bind_param(*value: Union[None, uuid.UUID, str]*, *dialect*) → Union[None, str]
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying `TypeEngine` object, and from there to the DBAPI `execute()` method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** – the `Dialect` in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying `TypeEngine` object, originally from the DBAPI cursor method `fetchone()` or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** – the `Dialect` in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

exception models.**LevelAccessDenied**
Bases: `Exception`

class models.**Link**(*schedule*)
Bases: `sqlalchemy.orm.decl_api.Model`

choice

id

link

schedule

schedule_id

class models.**OldUser**(***kwargs*)

Bases: backend.mixins.UserMixin, sqlalchemy.orm.decl_api.Model

autosave

confirmed_at

email

classmethod get_emails()

id

property is_active

last_schedule_id

password

schedules

class models.**Role**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

description

id

name

users

class models.**Schedule**(*data*, *user=None*)

Bases: sqlalchemy.orm.decl_api.Model

Table used to store Schedules in the database.

data

get_link()

id

last_modified_by

link

old_users

update_data(*data*)

Warning: the address of data must be different that of self.data For example: >>> data = schedule.data >>> data.label = "new_label" >>> schedule.update(data) … does not work ! Instead, do: >>> data = copy(schedule.data) >>> data.label = "new_label" >>> schedule.update(data) For more information, see: https://docs.sqlalchemy.org/en/13/orm/extensions/mutable.html

update_label(*label*)

update_last_modified_by(*uuid*)

users

exception models.**ScheduleDoNotMatchError**(*database_id*, *data_id*)

Bases: Exception

Exception that will occur if a user tries to update a schedule's data with a non-matching ID.

class models.**Usage**(*data*)

Bases: sqlalchemy.orm.decl_api.Model

blueprint

datetime

endpoint

id

path

remote_addr

speed

status

track_var

truncate_string(*key*, *value*)

ua_browser

ua_language

ua_platform

ua_version

url

url_args

username

view_args

class models.**User**(***kwargs*)

Bases: backend.mixins.UserMixin, sqlalchemy.orm.decl_api.Model

add_schedule(*schedule*, *level=0*)

autosave

created_at

email

fgs

first_name

classmethod get_emails()

get_id()

Returns the user's identification attribute

get_schedule(*id*)

Return the schedule in this user's schedule list matching the given ID. None if no match is found.

get_schedules()

Equivalent to User.schedules, but sorts the schedules according to their IDs to ensure similar behavior throughout the SQL databases.

id

last_name

last_schedule_id

last_seen_at

remove_schedule(*schedule:* models.Schedule)

Removes a schedule from the schedules list. If user owns this schedule, deletes the schedule for all users.

Parameters **schedule** – the schedule

roles

schedules

set_autosave(*autosave*)

set_last_schedule_id(*schedule_id*)

share_schedule_with_emails(*schedule*, **emails: str*, *level=1*)

models.**query_to_dataframe**(*query: flask_sqlalchemy.BaseQuery*, **args: Any*, ***kwargs: Any*) → pandas.core.frame.DataFrame

Parses a SQL query from the database into a dataframe.

Parameters

- **query** (*BaseQuery*) – the query to be read
- **args** (*Any*) – positional arguments to be passed to `pandas.read_sql()`
- **kwargs** (*Any*) – keyword arguments to be passed to `pandas.read_sql()`

Returns a dataframe of table

Return type pd.DataFrame

models.**reformat_status_in_dataframe**(*df: pandas.core.frame.DataFrame*)

Modifies in-place a dataframe in order to replace status (int) to status (str) with format 2XX, 4XX, 5XX, etc.

Parameters **df** (*pd.DataFrame*) – the dataframe containing a status column

models.**table_to_dataframe**(*table: sqlalchemy.orm.decl_api.Model*, **args: Any*, ***kwargs: Any*) → pandas.core.frame.DataFrame

Parses a table from the database into a dataframe.

Parameters

- **table** (*db.Model*) – the table to be parsed
- **args** (*Any*) – positional arguments to be passed to `pandas.read_sql()`
- **kwargs** (*Any*) – keyword arguments to be passed to `pandas.read_sql()`

Returns a dataframe of table

Return type pd.DataFrame

professors module

class professors.**Professor**(*name: str*, *email: Optional[str] = None*)

Bases: `object`

A professor is one or many people in charge of a given academical event.

Parameters

- **name** (*str*) – the name(s)
- **email** (*Optional[str]*) – the email(s)

professors.**merge_professors**(*professors: Iterable[*professors.Professor*]*) → *professors.Professor*

Merges multiple professors into one.

Parameters **professors** (*Iterable[Professor]*) – multiple professors

Returns the new professor

Return type *Professor*

Example

```
>>> p1 = Professor('Jean Moulin', 'jean.moulin@mail.com')
>>> p2 = Professor('Marc Potier', 'marc.potier@mail.com')
>>> p3 = merge_professors((p1, p2))
```

resources module

class resources.**INDEX**

Bases: `object`

The INDEX class only contains static string values.

These values are attributes in the events returned by ADE API. Remembering all the names can be quite difficult so this class is here to help with it, while providing robustness if any of these attributes might change.

ADDRESS = 'address1'

AVAILABLE_QUANTITY = 'availableQuantity'

CITY = 'city'

CODE = 'code'

CODE_X = 'codeX'

CODE_Y = 'codeY'

CODE_Z = 'codeZ'

COLOR = 'color'

CONSUMER = 'consumer'

COUNTRY = 'country'

CREATION = 'creation'

DURATION_IN_MINUTES = 'durationInMinutes'

EMAIL = 'email'

FACILITY_TYPE = 'type'

FATHER_ID = 'fatherId'

FATHER_NAME = 'fatherName'

FAX = 'fax'

FIRST_SLOT = 'firstSlot'

FIRST_WEEK = 'firstWeek'

FIST_DAY = 'firstDay'

ID = 'id'

INFO = 'info'

IP_ADDRESS = 'address2'

IS_GROUP = 'isGroup'

JOB_CATEGORY = 'jobCategory'

LAST_DAY = 'lastDay'

LAST_SLOT = 'lastSlot'

LAST_UPDATE = 'lastUpdate'

LAST_WEEK = 'lastWeek'

LEVEL_ACCESS = 'levelAccess'

MANAGER = 'manager'

NAME = 'name'

NB_EVENTS_PLACES = 'nbEventsPlaced'

NUMBER = 'number'

OWNER = 'owner'

PATH = 'path'

PHONE = 'telephone'

SIZE = 'size'

STATE = 'state'

TIMEZONE = 'timezone'

TYPE = 'category'

URL = 'url'

ZIP_CODE = 'zipCode'

class resources.**Resource**(***kwargs*)

Bases: object

A resource is anything that can be requested from ADE API: a teacher, a classroom, an activity…

Parameters **kwargs** (*Any*) – attribute(s) of this resource

class resources.**TYPES**

Bases: object

The TYPES class only contains static string values.

These values are possible resource types returned by ADE API. Remembering all the names can be quite difficult so this class is here to help with it, while providing robustness if any of these attributes might change.

CLASSROOM = 'classroom'

COURSE = 'category5'

COURSE_COMBO = 'trainee'

TEACHER = 'instructor'

schedules module

class schedules.**Schedule**(*project_id: str*, *schedule_id: typing.Optional[int] = None*, *label: str = l'New schedule'*)

Bases: object

A schedule is essentially a combination of courses stored as a master course, from which some events can be removed.

Parameters **project_id** – the schedule id matching this of the Database it is currently saved in. This parameter is automatically set when the schedule is saved for the first time.

add_course(*codes: Union[Iterable[str], str]*) → List[str]

Adds one or many courses to the schedule.

Parameters **codes** (*Union[Iterable[str], str]*) – the codes of the course added

Returns all the new codes added to the schedule

Return type List[str]

add_custom_event(*event: backend.events.CustomEvent*)

Adds a custom event to the schedule.

Parameters **event** (CustomEvent (or RecurringCustomEvent)) – the event to add

add_filter(*code: str*, *filter: Union[Iterable[str], str]*)

compute_best(*n_best: int = 5*, *safe_compute: bool = True*) → List[Iterable[backend.events.CustomEvent]]

Computes best schedules trying to minimize conflicts selecting, for each type of event, one event.

Parameters

- **n_best** (*int*) – number of best schedules to produce
- **safe_compute** (*bool*) – if True, ignore all redundant events at same time period

Returns the n_best schedules, but maybe less if cannot find n_best different schedules

Return type List[Iterable[evt.CustomEvent]]

get_courses() → List[backend.courses.Course]

Returns all the courses of this schedule as a list.

Returns the courses

Return type List[*Course*]

get_custom_event(*id: str*) → Optional[backend.events.CustomEvent]

Returns the custom event matching given id, or None if not found.

Parameters **id** (*str*) – the unique id of the event

Returns the custom event

Return type Optional[*CustomEvent*]

get_custom_event_color(*event: Optional[backend.events.CustomEvent] = None*, *id: Optional[str] = None*) → Optional[str]

Returns the color of a given custom event

Parameters

- **event** (*Optional[CustomEvent]*) – the event to remove
- **id** (*Optional[str]*) – the unique id of the event

Returns the color

Return type Optional[str]

get_events(*json: bool = False*, *schedule_number: int = 0*) → List[ics.event.Event]

Extracts all the events matching ids in the filtered_subcodes list.

Parameters

- **json** (*bool*) – whether or not the events are to be returned in a JSON format
- **schedule_number** (*int*) – the # of the schedule, 0 for main and 1 for best one, 2 for second best, etc.

Returns the events

Return type List[events]

get_ics_file(*schedule_number: int = 0*)

Returns the .ics (iCalendar) representation of this Schedule.

Parameters **schedule_number** (*int*) – the # of the schedule, 0 for main and 1 for best one, 2 for second best, etc.

Returns iCalendar-formatted schedule

Return type str

get_min_max_time_slots() → Tuple[str, str]

get_option(*option: str*) → bool

get_summary() → Dict[str, Dict[str, Set[str]]]

Returns the summary of all activities within the schedule.

Returns dict of course summaries

Return type Dict[str, Dict[str, Set[str]]]

is_empty()

remove_course(*code: str*)

Removes a course from the schedule.

Parameters **code** (*str*) – the code of the course to remove

remove_custom_event(*event: Optional[backend.events.CustomEvent] = None*, *id: Optional[str] = None*)

Removes a custom event from the schedule. If this event is present multiple times in the schedule, only delete the first occurrence.

Parameters

- **event** (*Optional[CustomEvent]*) – the event to remove
- **id** (*Optional[str]*) – the unique id of the event

remove_filter(*code: str*, *filter: Union[Iterable[str], str]*)

reset_best_schedules()

reset_color_palette()

reset_filters(*code*)

set_custom_event_attributes(*id: str*, ***kwargs: str*)

Changes the custom event's attributes.

Parameters

- **id** (*str*) – the unique id of the event
- **kwargs** (*str*) – the attributes and their value

set_option(*option: str*, *value: bool*)

schedules.**default_dict_any_to_set**() → collections.defaultdict

Create a collections.defaultdict object mapping each key to a set.

Returns the dictionary

Return type collections.defaultdict

schedules.**default_options**() → collections.defaultdict

Create a collections.defaultdict object mapping each key to a boolean. Default value is false.

Returns the dictionary

Return type collections.defaultdict

schedules.**evaluate_week**(*week: Iterable[Iterable[backend.events.CustomEvent]]*, *fts: Optional[Iterable[backend.events.CustomEvent]] = None*) → float

Evaluates how much a given week contains conflicts.

Parameters

- **week** (*Iterable[Iterable[evt.CustomEvent]]*) – events in a week as provided by *Schedule.compute_best()*
- **fts** (*Iterable[evt.CustomEvent]*) – additional events to take into account

Returns the sum of all the conflicts

Return type float

security module

security.**login**()

security.**logout**()

servers module

class servers.**Server**(**arg*, ***kwargs*)

Bases: `redis.client.Redis`

Subclass of Redis object, aiming to simplify the use of the server to few basic commands.

Parameters

- **args** – arguments passed to parent constructor
- **kwargs** – keyword arguments passed to parent constructor

Example

```
>>> s = Server(host='localhost', port=6379)
```

contains(**keys: str*) → int

Returns the number of keys that exist.

Parameters **keys** (*str*) – key(s) to be checked

Returns the number of keys that exist

Return type int

get_multiple_values(**keys: str*, *prefix: str = ''*, ***kwargs*) → Tuple[Dict[str, Any], List[str]]

Returns all the values corresponding the given keys. If key does not match any value, the key is returned explicitly tell that it is missing. An optional prefix can be added to every key.

Parameters

- **keys** (*str*) – the keys
- **prefix** (*str*) – the prefix to be added to each key

Returns a tuple containing all values found and all keys which did not match

Return type Tuple[Dict[str, Any], List[str]]

get_multiple_values_expired(**keys: str*, *prefix: str = ''*) → Tuple[Dict[str, Optional[bool]]]

Returns, for each key, wether a expire notification was issued or not, and None is returned if the case the key does not exist. An optional prefix can be added to every key.

Parameters

- **keys** (*str*) – the keys
- **prefix** (*str*) – the prefix to be added to each key

Returns a tuple containing all values found and all keys which did not match

Return type Tuple[Dict[str, Optional[bool]]]

get_value(*key: str*, *hmap: Optional[str] = None*) → Any

Returns the value with corresponding key stored in the server.

Parameters

- **key** (*str*) – the key
- **hmap** (*str*) – if present, will look for value stored in hash-map with this name

Returns the object stored in the server, None if not object matching the key

Return type Any

Example

```
>>> s.get_value('apple')
{'weight': 400, 'unit': 'g'}
```

is_running() → bool

Checks whether the server is running.

Returns True if the server is running

Return type bool

run()

Runs the server. Since redis is an externally run server, it cannot be started from the python code properly.

set_value(*key: str*, *value: Any*, *expire_in: Optional[Dict[str, int]] = None*, *notify_expire_in: Optional[Dict[str, int]] = None*, *hmap: bool = False*)

Store a pair key / value in the server, with an optional expiration time.

Parameters

- **key** (*str*) – the key
- **value** (*Any*) – any object that can be dumped (see pickle.dumps)
- **expire_in** (*Optional[Dict[str, int]]*) – dictionary of keyword arguments passed used to create a datetime.timedelta object
- **hmap** (*bool*) – True if the value passed is a hash-map

Example

```
>>> s.set_value('apple', {'weight': 400, 'unit': 'g'}, expire_in={'hours': 10})
```

shutdown()

Shuts the server down.

servers.**parse_redis_ttl_config**(*conf: Mapping[str, str]*) → Dict[str, Dict[str, int]]

Parses a config mapping (from a config file for example) into a usable TTL config. Each key, value pair holds the name of the resource stored in the server and the parameters to setup the default expiry duration.

Parameters **conf** (*Mapping[str, str]*) – the config mapping

Returns the TTL config mapping

Return type Dict[str, Dict[str, int]]

track_usage module

class track_usage.**ParsedUserAgent**(*string: str*)

Bases: werkzeug.user_agent.UserAgent

property browser

property platform

property version

track_usage.**after_request**(*response*)

track_usage.**before_request**()

uclouvain_apis module

class uclouvain_apis.**ADE**
Bases: *uclouvain_apis.API*

ENDPOINT = 'ade/v0'

class uclouvain_apis.**API**
Bases: object

class property AUTHORIZE_URL
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

BASE_URL = 'https://gw.api.uclouvain.be'

ENDPOINT = ''

class property TOKEN_URL
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

class property authorize
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

classmethod get(*url*, ***kwargs*)

class property token
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

class property url
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

class uclouvain_apis.**My**
Bases: *uclouvain_apis.API*

ENDPOINT = 'my/v0'

class uclouvain_apis.**MyADE**
Bases: *uclouvain_apis.API*

```
ENDPOINT = 'myADE/v1'
```

1.2.4 views

account module

class account.**AccountDecoder**(*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *strict=True*, *object_pairs_hook=None*)

Bases: `json.decoder.JSONDecoder`

Subclass of json decoder made for the account-specific JSON decodings.

decode(*obj: Any*, *w: Optional[Any] = None*) → str

Return the Python representation of `s` (a `str` instance containing a JSON document).

class account.**AccountEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

Subclass of json decoder made for the account-specific JSON encodings.

default(*obj: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

account.**add_external_calendar**()

account.**autosave**()

account.**before_account_request**()

account.**delete_external_calendar**(*id*)

account.**delete_schedule**(*id*)

account.**get_data**()

account.**index**()

account.**load_schedule**(*id*)

account.**save**()

account.**update_label**(*id*)

admin module

admin.**get_data**()

admin.**index**()

api module

class api.**ApiDecoder**(*, *object_hook=None*, *parse_float=None*, *parse_int=None*, *parse_constant=None*, *strict=True*, *object_pairs_hook=None*)

Bases: `json.decoder.JSONDecoder`

Subclass of json decoder made for the calendar-specific JSON decodings.

decode(*obj: Any*, *w: Optional[Any] = None*) → str

Return the Python representation of `s` (a `str` instance containing a JSON document).

class api.**ApiEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

Subclass of json decoder made for the calendar-specific JSON encodings.

default(*obj: Any*) → Any

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

api.**before_api_request**()

api.**get_events**()

API endpoint to fetch the schedule matching the various arguments.

The request accepts 3 arguments:

- **view: if set to True, shows the schedule in ADE-Scheduler's schedule viewer.** By default, view = False the data is returned in JSON format.
- **year: defines the academical year. Accepted format is YYYY-YYYY where the second year is the first + 1.** e.g.: 2020-20201
- **code: the various code of the courses to be added to the schedule.** e.g.: LELEC2885, LMECA2170, LEPL1104,… To specify a list of codes, use the following format: /schedule?code=CODE_1&code=CODE_2&code=CODE_3 and so on.
- **filtered events: the IDs of the events you want to filter.** Say the course which code is "CODE" has two TPs with IDs "TP: CODE-Q1A" and "TP: CODE-Q1B" and you want to filter out the events of "TP: CODE-Q1A", you can specify such a filter by specifying: /schedule?code=CODE&CODE=TP: CODE-Q1A&…

/!To make sure to avoid any problems, respect the case-sensitiveness of the codes and event IDs. Generally, those are uppercase. /!

Example: https://ade-scheduler.info.ucl.ac.be/api/events?year=2020-2021&code=LMECA2170&code=LEPL1104&view=true

or, with filtered events: https://ade-scheduler.info.ucl.ac.be/api/events?year=2020-2021&code=LEPL1104&LEPL1104=TP: LEPL1104_Q2B-APE&view=true

`api.schedule_shield()`
API route for badges on the GitHub Repository, per shields.io

`api.user_shield()`
API route for badges on the GitHub Repository, per shields.io

calendar module

Calendar printing functions

Note when comparing these calendars to the ones printed by cal(1): By default, these calendars have Monday as the first day of the week, and Sunday as the last (the European convention). Use setfirstweekday() to set the first day of the week (0=Monday, 6=Sunday).

`class calendar.Calendar(firstweekday=0)`
Bases: `object`

Base calendar class. This class doesn't do any formatting. It simply provides data to subclasses.

`property firstweekday`

`getfirstweekday()`

`itermonthdates(year, month)`
Return an iterator for one month. The iterator will yield datetime.date values and will always iterate through complete weeks, so it will yield dates outside the specified month.

`itermonthdays(year, month)`
Like itermonthdates(), but will yield day numbers. For days outside the specified month the day number is 0.

`itermonthdays2(year, month)`
Like itermonthdates(), but will yield (day number, weekday number) tuples. For days outside the specified month the day number is 0.

`itermonthdays3(year, month)`
Like itermonthdates(), but will yield (year, month, day) tuples. Can be used for dates outside of datetime.date range.

`itermonthdays4(year, month)`
Like itermonthdates(), but will yield (year, month, day, day_of_week) tuples. Can be used for dates outside of datetime.date range.

`iterweekdays()`
Return an iterator for one week of weekday numbers starting with the configured first one.

`monthdatescalendar(year, month)`
Return a matrix (list of lists) representing a month's calendar. Each row represents a week; week entries are datetime.date values.

`monthdays2calendar(year, month)`
Return a matrix representing a month's calendar. Each row represents a week; week entries are (day number, weekday number) tuples. Day numbers outside this month are zero.

monthdayscalendar(*year*, *month*)
Return a matrix representing a month's calendar. Each row represents a week; days outside this month are zero.

setfirstweekday(*firstweekday*)

yeardatescalendar(*year*, *width=3*)
Return the data for the specified year ready for formatting. The return value is a list of month rows. Each month row contains up to width months. Each month contains between 4 and 6 weeks and each week contains 1-7 days. Days are datetime.date objects.

yeardays2calendar(*year*, *width=3*)
Return the data for the specified year ready for formatting (similar to yeardatescalendar()). Entries in the week lists are (day number, weekday number) tuples. Day numbers outside this month are zero.

yeardayscalendar(*year*, *width=3*)
Return the data for the specified year ready for formatting (similar to yeardatescalendar()). Entries in the week lists are day numbers. Day numbers outside this month are zero.

class calendar.**HTMLCalendar**(*firstweekday=0*)
Bases: *calendar.Calendar*

This calendar returns complete HTML pages.

cssclass_month = 'month'

cssclass_month_head = 'month'

cssclass_noday = 'noday'

cssclass_year = 'year'

cssclass_year_head = 'year'

cssclasses = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']

cssclasses_weekday_head = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']

formatday(*day*, *weekday*)
Return a day as a table cell.

formatmonth(*theyear*, *themonth*, *withyear=True*)
Return a formatted month as a table.

formatmonthname(*theyear*, *themonth*, *withyear=True*)
Return a month name as a table row.

formatweek(*theweek*)
Return a complete week as a table row.

formatweekday(*day*)
Return a weekday name as a table header.

formatweekheader()
Return a header for a week as a table row.

formatyear(*theyear*, *width=3*)
Return a formatted year as a table of tables.

formatyearpage(*theyear*, *width=3*, *css='calendar.css'*, *encoding=None*)
Return a formatted year as a complete HTML page.

exception calendar.**IllegalMonthError**(*month*)
Bases: ValueError

exception calendar.**IllegalWeekdayError**(*weekday*)
Bases: ValueError

class calendar.**LocaleHTMLCalendar**(*firstweekday=0*, *locale=None*)
Bases: *calendar.HTMLCalendar*

This class can be passed a locale name in the constructor and will return month and weekday names in the specified locale. If this locale includes an encoding all strings containing month and weekday names will be returned as unicode.

formatmonthname(*theyear*, *themonth*, *withyear=True*)
Return a month name as a table row.

formatweekday(*day*)
Return a weekday name as a table header.

class calendar.**LocaleTextCalendar**(*firstweekday=0*, *locale=None*)
Bases: *calendar.TextCalendar*

This class can be passed a locale name in the constructor and will return month and weekday names in the specified locale. If this locale includes an encoding all strings containing month and weekday names will be returned as unicode.

formatmonthname(*theyear*, *themonth*, *width*, *withyear=True*)
Return a formatted month name.

formatweekday(*day*, *width*)
Returns a formatted week day name.

class calendar.**TextCalendar**(*firstweekday=0*)
Bases: *calendar.Calendar*

Subclass of Calendar that outputs a calendar as a simple plain text similar to the UNIX program cal.

formatday(*day*, *weekday*, *width*)
Returns a formatted day.

formatmonth(*theyear*, *themonth*, *w=0*, *l=0*)
Return a month's calendar string (multi-line).

formatmonthname(*theyear*, *themonth*, *width*, *withyear=True*)
Return a formatted month name.

formatweek(*theweek*, *width*)
Returns a single week in a string (no newline).

formatweekday(*day*, *width*)
Returns a formatted week day name.

formatweekheader(*width*)
Return a header for a week.

formatyear(*theyear*, *w=2*, *l=1*, *c=6*, *m=3*)
Returns a year's calendar as a multi-line string.

prmonth(*theyear*, *themonth*, *w=0*, *l=0*)
Print a month's calendar.

prweek(*theweek*, *width*)
Print a single week (no newline).

pryear(*theyear*, *w=0*, *l=0*, *c=6*, *m=3*)
Print a year's calendar.

calendar.**calendar**(*theyear*, *w=2*, *l=1*, *c=6*, *m=3*)
Returns a year's calendar as a multi-line string.

calendar.**firstweekday**()

calendar.**isleap**(*year*)
Return True for leap years, False for non-leap years.

calendar.**leapdays**(*y1*, *y2*)
Return number of leap years in range [y1, y2). Assume y1 <= y2.

calendar.**month**(*theyear*, *themonth*, *w=0*, *l=0*)
Return a month's calendar string (multi-line).

calendar.**monthcalendar**(*year*, *month*)
Return a matrix representing a month's calendar. Each row represents a week; days outside this month are zero.

calendar.**monthrange**(*year*, *month*)
Return weekday (0-6 ~ Mon-Sun) and number of days (28-31) for year, month.

calendar.**prcal**(*theyear*, *w=0*, *l=0*, *c=6*, *m=3*)
Print a year's calendar.

calendar.**prmonth**(*theyear*, *themonth*, *w=0*, *l=0*)
Print a month's calendar.

calendar.**setfirstweekday**(*firstweekday*)

calendar.**timegm**(*tuple*)
Unrelated but handy function to calculate Unix timestamp from GMT.

calendar.**weekday**(*year*, *month*, *day*)
Return weekday (0-6 ~ Mon-Sun) for year, month (1-12), day (1-31).

calendar.**weekheader**(*width*)
Return a header for a week.

classroom module

classroom.**before_classroom_request**()

classroom.**get_data**()

classroom.**get_occupation**(*id*)

classroom.**index**()

contact module

contact.**index**()

contribute module

`contribute.index()`

help module

`help.index()`

security module

`security.login()`

`security.logout()`

utils module

`utils.autoload_schedule()`

`utils.autosave_schedule(response)`

`utils.init_session()`

whatisnew module

`whatisnew.index()`

- **Backend**

Note: This module implements all the computationally intensive and complex functions, in a more or less optimized manner, in order to provide complete services to end users.

- **Views**

Note: This module makes the interface between the backend and the frontend. All requests are handled by this module.

1.2.5 command-line tools

Here are listed the different command-line tools which can be used directly in a command shell. **Warning**: only the commands we have created are documented here, but many more exist !

Documentation is generated using sphinx-click but does not offer a pretty rendering. For a more detailed documentation, please use `flask --help` directly in the command shell.

cli_api_usage module

api_usage

Performs operations on the ApiUsage table.

```
api_usage [OPTIONS] COMMAND [ARGS]...
```

plot-requests-hist

```
api_usage plot-requests-hist [OPTIONS]
```

Options

--latest <latest>
Only shows data from latest days.

cli_client module

client

Performs operation on client.

```
client [OPTIONS] COMMAND [ARGS]...
```

expire-in

Returns current token expiration time.

```
client expire-in [OPTIONS]
```

renew

Renews client's token.

```
client renew [OPTIONS]
```

cli_external_calendars module

cli_mails module

cli_plots module

plots

Performs operations with Plotly backend

```
plots [OPTIONS] COMMAND [ARGS]...
```

generate

```
plots generate [OPTIONS]
```

Options

-s, --select <select>
Generate a plot for a given function. By default, will generate all plots.

Options all | api-usage-requests-hist | ics-requests-hist | platforms-pie | requests-per-blueprint-hist | unique-ip-addresses-per-day | users-hist | users-emails-pie | users-last-seen | views-per-blueprint-hist

cli_redis module

redis

Performs operations on Redis.

```
redis [OPTIONS] COMMAND [ARGS]...
```

count

Counts the number of keys matching a given pattern present in Redis

```
redis count [OPTIONS]
```

Options

-p, --pattern <pattern>
Pattern of the keys. By default, selects all the keys.

--session, --no-session
Display the amount of user sessions.

--code, --no-code
Display the amount of unique codes.

expire

Apply an expiry time to keys in Redis.

```
redis expire [OPTIONS]
```

Options

-p, --pattern <pattern>
Pattern of the keys. By default, selects all the keys.

-t, --time <time>
Apply an expiry time. If no expiry is specified, simply deletes the keys.

-r, --random <random>
Apply a random expiry time, between provided lower and upper bounds.

flush

Resets Redis, deleting all keys.

```
redis flush [OPTIONS]
```

update

Updates Redis' tables.

```
redis update [OPTIONS]
```

cli_roles module

cli_schedules module

schedules

Performs operations with the schedules

```
schedules [OPTIONS] COMMAND [ARGS]...
```

stats

Returns some statistics about schedules.

```
schedules stats [OPTIONS]
```

Options

-m <m>
Number of most common codes, all by default.

-n <n>
Number of most common prefixes, all by default.

cli_sql module

sql

Performs operations on the SQL database.

```
sql [OPTIONS] COMMAND [ARGS]...
```

dump

Dumps the database.

```
sql dump [OPTIONS]
```

Options

-o, --output <output>
Output file.

fix

```
sql fix [OPTIONS]
```

init

Initialization of the SQL database.

```
sql init [OPTIONS]
```

load

Loads the database from a dumpfile. Tested with PostgreSQL, MySQL & SQLite.

```
sql load [OPTIONS]
```

Options

-i, --input <input>
Input file.

cli_usage module

usage

Performs operations on the Usage table.

```
usage [OPTIONS] COMMAND [ARGS]...
```

plot-ics-requests-hist

```
usage plot-ics-requests-hist [OPTIONS]
```

plot-platforms-pie

```
usage plot-platforms-pie [OPTIONS]
```

plot-requests-per-blueprint-hist

```
usage plot-requests-per-blueprint-hist [OPTIONS]
```

Options

--latest <latest>
Only shows data from latest days.

plot-unique-ip-addresses-per-day

```
usage plot-unique-ip-addresses-per-day [OPTIONS]
```

plot-views-per-blueprint-hist

```
usage plot-views-per-blueprint-hist [OPTIONS]
```

Options

--latest <latest>
Only shows data from latest days.

stats

```
usage stats [OPTIONS]
```

cli_users module

users

Performs actions on users.

```
users [OPTIONS] COMMAND [ARGS]...
```

count

Count the number of current users.

```
users count [OPTIONS]
```

plot-users-emails-pie

```
users plot-users-emails-pie [OPTIONS]
```

plot-users-hist

```
users plot-users-hist [OPTIONS]
```

plot-users-last-seen

```
users plot-users-last-seen [OPTIONS]
```

stats

Returns some statistics about users.

```
users stats [OPTIONS]
```

Note: This module contains all the command-line tools built to interact with ADE-Scheduler's data such database statistics.

CHAPTER

TWO

CONTRIBUTOR GUIDE

We really encourage people who want to contribute in any way to do so ! There are many ways to help us:

- Pointing a problem
- Asking for a new feature
- Fixing a problem
- Adding language translations
- Fixing typos in the code

and so on…

Please take a look at our Contribution tutorials.

CHAPTER

THREE

INDEX ET TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

account, 59
ade_api, 29
admin, 60
api, 60

c

calendar, 61
classroom, 64
classrooms, 35
contact, 64
contribute, 65
cookies, 36
courses, 36

e

events, 38

h

help, 65

m

manager, 44
mixins, 46
models, 46

p

professors, 51

r

resources, 51

s

schedules, 53
security, 65
servers, 56

t

track_usage, 57

u

uclouvain_apis, 58
utils, 65

w

whatisnew, 65

INDEX

Symbols

--code
    redis-count command line option, 67
--input
    sql-load command line option, 70
--latest
    api_usage-plot-requests-hist command line option, 66
    usage-plot-requests-per-blueprint-hist command line option, 70
    usage-plot-views-per-blueprint-hist command line option, 71
--no-code
    redis-count command line option, 67
--no-session
    redis-count command line option, 67
--output
    sql-dump command line option, 69
--pattern
    redis-count command line option, 67
    redis-expire command line option, 68
--random
    redis-expire command line option, 68
--select
    plots-generate command line option, 67
--session
    redis-count command line option, 67
--time
    redis-expire command line option, 68
-i
    sql-load command line option, 70
-m
    schedules-stats command line option, 69
-n
    schedules-stats command line option, 69
-o
    sql-dump command line option, 69
-p
    redis-count command line option, 67
    redis-expire command line option, 68
-r
    redis-expire command line option, 68
-s
    plots-generate command line option, 67
-t
    redis-expire command line option, 68

A

AcademicalEvent (*class in events*), 38
account
    module, 59
AccountDecoder (*class in account*), 59
AccountEncoder (*class in account*), 59
add_activity() (*courses.Course method*), 36
add_course() (*schedules.Schedule method*), 53
add_custom_event() (*schedules.Schedule method*), 53
add_external_calendar() (*in module account*), 59
add_filter() (*schedules.Schedule method*), 53
add_role() (*mixins.UserMixin method*), 46
add_schedule() (*models.User method*), 49
Address (*class in classrooms*), 35
ADDRESS (*resources.INDEX attribute*), 51
ADE (*class in uclouvain_apis*), 58
ade_api
    module, 29
admin
    module, 60
after_request() (*in module track_usage*), 57
alarms (*events.EventCM attribute*), 39
alarms (*events.EventEXAM attribute*), 40
alarms (*events.EventEXTERN attribute*), 40
alarms (*events.EventLABO attribute*), 40
alarms (*events.EventORAL attribute*), 41
alarms (*events.EventOTHER attribute*), 41
alarms (*events.EventTP attribute*), 42
alarms (*events.RecurringCustomEvent attribute*), 42
AnonymousUser (*class in mixins*), 46
api
    module, 60
API (*class in uclouvain_apis*), 58
api_usage-plot-requests-hist command line option
    --latest, 66
ApiDecoder (*class in api*), 60

ApiEncoder (*class in api*), 60
ApiUsage (*class in models*), 46
approved (*models.ExternalCalendar attribute*), 46
attendees (*events.EventCM attribute*), 39
attendees (*events.EventEXAM attribute*), 40
attendees (*events.EventEXTERN attribute*), 40
attendees (*events.EventLABO attribute*), 40
attendees (*events.EventORAL attribute*), 41
attendees (*events.EventOTHER attribute*), 41
attendees (*events.EventTP attribute*), 42
attendees (*events.RecurringCustomEvent attribute*), 42
authorize (*uclouvain_apis.API property*), 58
AUTHORIZE_URL (*uclouvain_apis.API property*), 58
autoload_schedule() (*in module utils*), 65
autosave (*models.OldUser attribute*), 48
autosave (*models.User attribute*), 49
autosave() (*in module account*), 59
autosave_schedule() (*in module utils*), 65
AVAILABLE_QUANTITY (*resources.INDEX attribute*), 51

B

BASE_URL (*uclouvain_apis.API attribute*), 58
before_account_request() (*in module account*), 59
before_api_request() (*in module api*), 60
before_classroom_request() (*in module classroom*), 64
before_request() (*in module track_usage*), 57
blueprint (*models.Usage attribute*), 49
browser (*track_usage.ParsedUserAgent property*), 57

C

calendar
    module, 61
Calendar (*class in calendar*), 61
calendar() (*in module calendar*), 63
categories (*events.EventCM attribute*), 39
categories (*events.EventEXAM attribute*), 40
categories (*events.EventEXTERN attribute*), 40
categories (*events.EventLABO attribute*), 40
categories (*events.EventORAL attribute*), 41
categories (*events.EventOTHER attribute*), 41
categories (*events.EventTP attribute*), 42
categories (*events.RecurringCustomEvent attribute*), 42
choice (*models.Link attribute*), 47
CITY (*resources.INDEX attribute*), 51
classroom
    module, 64
Classroom (*class in classrooms*), 35
CLASSROOM (*resources.TYPES attribute*), 53
classrooms
    module, 35
Client (*class in ade_api*), 29
code (*models.ExternalCalendar attribute*), 46
CODE (*resources.INDEX attribute*), 51
code_exists() (*manager.Manager method*), 44
CODE_X (*resources.INDEX attribute*), 51
CODE_Y (*resources.INDEX attribute*), 51
CODE_Z (*resources.INDEX attribute*), 51
COLOR (*resources.INDEX attribute*), 51
compute_best() (*schedules.Schedule method*), 53
confirmed_at (*models.OldUser attribute*), 48
CONSUMER (*resources.INDEX attribute*), 51
contact
    module, 64
contains() (*servers.Server method*), 56
contribute
    module, 65
cookies
    module, 36
COUNTRY (*resources.INDEX attribute*), 51
Course (*class in courses*), 36
COURSE (*resources.TYPES attribute*), 53
COURSE_COMBO (*resources.TYPES attribute*), 53
courses
    module, 36
created (*events.EventCM attribute*), 39
created (*events.EventEXAM attribute*), 40
created (*events.EventEXTERN attribute*), 40
created (*events.EventLABO attribute*), 41
created (*events.EventORAL attribute*), 41
created (*events.EventOTHER attribute*), 41
created (*events.EventTP attribute*), 42
created (*events.RecurringCustomEvent attribute*), 42
created_at (*models.User attribute*), 49
CREATION (*resources.INDEX attribute*), 51
cssclass_month (*calendar.HTMLCalendar attribute*), 62
cssclass_month_head (*calendar.HTMLCalendar attribute*), 62
cssclass_noday (*calendar.HTMLCalendar attribute*), 62
cssclass_year (*calendar.HTMLCalendar attribute*), 62
cssclass_year_head (*calendar.HTMLCalendar attribute*), 62
cssclasses (*calendar.HTMLCalendar attribute*), 62
cssclasses_weekday_head (*calendar.HTMLCalendar attribute*), 62
CustomEvent (*class in events*), 38

D

data (*models.Schedule attribute*), 48
datetime (*models.ApiUsage attribute*), 46
datetime (*models.Usage attribute*), 49
decode() (*account.AccountDecoder method*), 59
decode() (*api.ApiDecoder method*), 60
default() (*account.AccountEncoder method*), 59
default() (*api.ApiEncoder method*), 60

DEFAULT_COLOR (*events.CustomEvent attribute*), 38
DEFAULT_COLOR (*events.RecurringCustomEvent attribute*), 42
default_dict_any_to_set() (*in module schedules*), 55
default_options() (*in module schedules*), 55
delete_external_calendar() (*in module account*), 59
delete_external_calendar() (*manager.Manager method*), 44
delete_schedule() (*in module account*), 59
description (*events.EventCM attribute*), 39
description (*events.EventEXAM attribute*), 40
description (*events.EventEXTERN attribute*), 40
description (*events.EventLABO attribute*), 41
description (*events.EventORAL attribute*), 41
description (*events.EventOTHER attribute*), 41
description (*events.EventTP attribute*), 42
description (*events.RecurringCustomEvent attribute*), 42
description (*models.ExternalCalendar attribute*), 46
description (*models.Role attribute*), 48
DummyClient (*class in ade_api*), 29
DURATION_IN_MINUTES (*resources.INDEX attribute*), 51

E

email (*models.OldUser attribute*), 48
email (*models.User attribute*), 49
EMAIL (*resources.INDEX attribute*), 51
endpoint (*models.Usage attribute*), 49
ENDPOINT (*uclouvain_apis.ADE attribute*), 58
ENDPOINT (*uclouvain_apis.API attribute*), 58
ENDPOINT (*uclouvain_apis.My attribute*), 58
ENDPOINT (*uclouvain_apis.MyADE attribute*), 58
evaluate_week() (*in module schedules*), 55
EventCM (*class in events*), 39
EventEXAM (*class in events*), 39
EventEXTERN (*class in events*), 40
EventLABO (*class in events*), 40
EventORAL (*class in events*), 41
EventOTHER (*class in events*), 41
events
    module, 38
EventTP (*class in events*), 42
expire_in() (*ade_api.Client method*), 29
expire_in() (*ade_api.DummyClient method*), 30
expire_in() (*ade_api.FakeClient method*), 31
ExpiredTokenError, 31
ExternalCalendar (*class in models*), 46
ExternalCalendarAlreadyExistsError, 44
extract_code() (*in module events*), 43
extract_datetime() (*in module events*), 43
extract_type() (*in module events*), 43

F

FACILITY_TYPE (*resources.INDEX attribute*), 51
FakeClient (*class in ade_api*), 31
FATHER_ID (*resources.INDEX attribute*), 51
FATHER_NAME (*resources.INDEX attribute*), 52
FAX (*resources.INDEX attribute*), 52
fgs (*models.User attribute*), 49
first_name (*models.User attribute*), 49
FIRST_SLOT (*resources.INDEX attribute*), 52
FIRST_WEEK (*resources.INDEX attribute*), 52
firstweekday (*calendar.Calendar property*), 61
firstweekday() (*in module calendar*), 64
FIST_DAY (*resources.INDEX attribute*), 52
FOLDER (*ade_api.FakeClient attribute*), 31
formatday() (*calendar.HTMLCalendar method*), 62
formatday() (*calendar.TextCalendar method*), 63
formatmonth() (*calendar.HTMLCalendar method*), 62
formatmonth() (*calendar.TextCalendar method*), 63
formatmonthname() (*calendar.HTMLCalendar method*), 62
formatmonthname() (*calendar.LocaleHTMLCalendar method*), 63
formatmonthname() (*calendar.LocaleTextCalendar method*), 63
formatmonthname() (*calendar.TextCalendar method*), 63
formatweek() (*calendar.HTMLCalendar method*), 62
formatweek() (*calendar.TextCalendar method*), 63
formatweekday() (*calendar.HTMLCalendar method*), 62
formatweekday() (*calendar.LocaleHTMLCalendar method*), 63
formatweekday() (*calendar.LocaleTextCalendar method*), 63
formatweekday() (*calendar.TextCalendar method*), 63
formatweekheader() (*calendar.HTMLCalendar method*), 62
formatweekheader() (*calendar.TextCalendar method*), 63
formatyear() (*calendar.HTMLCalendar method*), 62
formatyear() (*calendar.TextCalendar method*), 63
formatyearpage() (*calendar.HTMLCalendar method*), 62
from_event() (*events.EventEXTERN class method*), 40

G

generate_empty_dataframe() (*in module courses*), 37
get() (*uclouvain_apis.API class method*), 58
get_activities() (*ade_api.DummyClient method*), 30
get_activities() (*courses.Course method*), 36
get_classrooms() (*ade_api.DummyClient method*), 30
get_classrooms() (*manager.Manager method*), 44

get_codes_matching() (*manager.Manager method*), 44
get_course_resources() (*ade_api.DummyClient method*), 30
get_course_resources() (*manager.Manager method*), 44
get_courses() (*manager.Manager method*), 44
get_courses() (*schedules.Schedule method*), 53
get_custom_event() (*schedules.Schedule method*), 53
get_custom_event_color() (*schedules.Schedule method*), 54
get_data() (*in module account*), 59
get_data() (*in module admin*), 60
get_data() (*in module classroom*), 64
get_default_project_id() (*manager.Manager method*), 44
get_emails() (*models.OldUser class method*), 48
get_emails() (*models.User class method*), 49
get_events() (*courses.Course method*), 37
get_events() (*in module api*), 60
get_events() (*schedules.Schedule method*), 54
get_events_in_classroom() (*manager.Manager method*), 44
get_external_calendars() (*manager.Manager method*), 44
get_geo_locations() (*in module classrooms*), 35
get_ics_file() (*schedules.Schedule method*), 54
get_id() (*events.AcademicalEvent method*), 38
get_id() (*models.User method*), 49
get_link() (*manager.Manager method*), 45
get_link() (*models.Schedule method*), 48
get_min_max_time_slots() (*schedules.Schedule method*), 54
get_multiple_values() (*servers.Server method*), 56
get_multiple_values_expired() (*servers.Server method*), 56
get_oauth_token() (*in module cookies*), 36
get_occupation() (*in module classroom*), 64
get_option() (*schedules.Schedule method*), 54
get_plots() (*manager.Manager method*), 45
get_project_ids() (*ade_api.DummyClient method*), 30
get_project_ids() (*manager.Manager method*), 45
get_resource_ids() (*ade_api.DummyClient method*), 30
get_resource_ids() (*manager.Manager method*), 45
get_resources() (*ade_api.DummyClient method*), 30
get_resources() (*manager.Manager method*), 45
get_response_path() (*in module ade_api*), 31
get_schedule() (*manager.Manager method*), 45
get_schedule() (*models.User method*), 49
get_schedules() (*models.User method*), 49
get_summary() (*courses.Course method*), 37
get_summary() (*schedules.Schedule method*), 54
get_token() (*in module ade_api*), 32
get_value() (*servers.Server method*), 56
get_week() (*events.CustomEvent method*), 39
getfirstweekday() (*calendar.Calendar method*), 61
GUID (*class in models*), 46

H

has_role() (*mixins.AnonymousUser method*), 46
has_role() (*mixins.UserMixin method*), 46
help
    module, 65
HTMLCalendar (*class in calendar*), 62

I

id (*models.ApiUsage attribute*), 46
id (*models.ExternalCalendar attribute*), 46
id (*models.Link attribute*), 47
id (*models.OldUser attribute*), 48
id (*models.Role attribute*), 48
id (*models.Schedule attribute*), 48
id (*models.Usage attribute*), 49
id (*models.User attribute*), 49
ID (*resources.INDEX attribute*), 52
IllegalMonthError, 62
IllegalWeekdayError, 62
impl (*models.GUID attribute*), 47
INDEX (*class in resources*), 51
index() (*in module account*), 59
index() (*in module admin*), 60
index() (*in module classroom*), 64
index() (*in module contact*), 64
index() (*in module contribute*), 65
index() (*in module help*), 65
index() (*in module whatisnew*), 65
INFO (*resources.INDEX attribute*), 52
init_session() (*in module utils*), 65
intersects() (*events.CustomEvent method*), 39
IP_ADDRESS (*resources.INDEX attribute*), 52
is_active (*models.OldUser property*), 48
is_empty() (*schedules.Schedule method*), 54
is_expired() (*ade_api.Client method*), 29
is_expired() (*ade_api.DummyClient method*), 31
is_expired() (*ade_api.FakeClient method*), 31
IS_GROUP (*resources.INDEX attribute*), 52
is_running() (*servers.Server method*), 57
isleap() (*in module calendar*), 64
itermonthdates() (*calendar.Calendar method*), 61
itermonthdays() (*calendar.Calendar method*), 61
itermonthdays2() (*calendar.Calendar method*), 61
itermonthdays3() (*calendar.Calendar method*), 61
itermonthdays4() (*calendar.Calendar method*), 61
iterweekdays() (*calendar.Calendar method*), 61

J

JOB_CATEGORY (*resources.INDEX attribute*), 52
json() (*events.AcademicalEvent method*), 38
json() (*events.CustomEvent method*), 39
json() (*events.RecurringCustomEvent method*), 42

K

KEYWORDS (*events.AcademicalEvent attribute*), 38
KEYWORDS (*events.EventCM attribute*), 39
KEYWORDS (*events.EventEXAM attribute*), 40
KEYWORDS (*events.EventEXTERN attribute*), 40
KEYWORDS (*events.EventLABO attribute*), 40
KEYWORDS (*events.EventORAL attribute*), 41
KEYWORDS (*events.EventOTHER attribute*), 41
KEYWORDS (*events.EventTP attribute*), 42

L

LAST_DAY (*resources.INDEX attribute*), 52
last_modified (*events.EventCM attribute*), 39
last_modified (*events.EventEXAM attribute*), 40
last_modified (*events.EventEXTERN attribute*), 40
last_modified (*events.EventLABO attribute*), 41
last_modified (*events.EventORAL attribute*), 41
last_modified (*events.EventOTHER attribute*), 41
last_modified (*events.EventTP attribute*), 42
last_modified (*events.RecurringCustomEvent attribute*), 42
last_modified_by (*models.Schedule attribute*), 48
last_name (*models.User attribute*), 49
last_schedule_id (*models.OldUser attribute*), 48
last_schedule_id (*models.User attribute*), 49
last_seen_at (*models.User attribute*), 50
LAST_SLOT (*resources.INDEX attribute*), 52
LAST_UPDATE (*resources.INDEX attribute*), 52
LAST_WEEK (*resources.INDEX attribute*), 52
leapdays() (*in module calendar*), 64
LEVEL_ACCESS (*resources.INDEX attribute*), 52
LevelAccessDenied, 47
Link (*class in models*), 47
link (*models.Link attribute*), 47
link (*models.Schedule attribute*), 48
load_dialect_impl() (*models.GUID method*), 47
load_response() (*in module ade_api*), 32
load_schedule() (*in module account*), 59
LocaleHTMLCalendar (*class in calendar*), 63
LocaleTextCalendar (*class in calendar*), 63
location (*events.EventCM attribute*), 39
location (*events.EventEXAM attribute*), 40
location (*events.EventEXTERN attribute*), 40
location (*events.EventLABO attribute*), 41
location (*events.EventORAL attribute*), 41
location (*events.EventOTHER attribute*), 41
location (*events.EventTP attribute*), 42
location (*events.RecurringCustomEvent attribute*), 42
location() (*classrooms.Classroom method*), 35
login() (*in module security*), 55, 65
logout() (*in module security*), 55, 65

M

manager
    module, 44
Manager (*class in manager*), 44
MANAGER (*resources.INDEX attribute*), 52
matches() (*events.AcademicalEvent class method*), 38
merge_classrooms() (*in module classrooms*), 35
merge_courses() (*in module courses*), 37
merge_professors() (*in module professors*), 51
mixins
    module, 46
models
    module, 46
module
    account, 59
    ade_api, 29
    admin, 60
    api, 60
    calendar, 61
    classroom, 64
    classrooms, 35
    contact, 64
    contribute, 65
    cookies, 36
    courses, 36
    events, 38
    help, 65
    manager, 44
    mixins, 46
    models, 46
    professors, 51
    resources, 51
    schedules, 53
    security, 55, 65
    servers, 56
    track_usage, 57
    uclouvain_apis, 58
    utils, 65
    whatisnew, 65
month() (*in module calendar*), 64
monthcalendar() (*in module calendar*), 64
monthdatescalendar() (*calendar.Calendar method*), 61
monthdays2calendar() (*calendar.Calendar method*), 61
monthdayscalendar() (*calendar.Calendar method*), 62
monthrange() (*in module calendar*), 64
My (*class in uclouvain_apis*), 58
MyADE (*class in uclouvain_apis*), 58

N

name (*models.ExternalCalendar attribute*), 46
name (*models.Role attribute*), 48
NAME (*resources.INDEX attribute*), 52
NB_EVENTS_PLACES (*resources.INDEX attribute*), 52
NUMBER (*resources.INDEX attribute*), 52

O

old_users (*models.Schedule attribute*), 48
OldUser (*class in models*), 48
organizer (*events.EventCM attribute*), 39
organizer (*events.EventEXAM attribute*), 40
organizer (*events.EventEXTERN attribute*), 40
organizer (*events.EventLABO attribute*), 41
organizer (*events.EventORAL attribute*), 41
organizer (*events.EventOTHER attribute*), 41
organizer (*events.EventTP attribute*), 42
organizer (*events.RecurringCustomEvent attribute*), 42
overlap() (*events.CustomEvent method*), 39
OWNER (*resources.INDEX attribute*), 52

P

parse_activity() (*in module ade_api*), 32
parse_event() (*in module ade_api*), 32
parse_redis_ttl_config() (*in module servers*), 57
ParsedUserAgent (*class in track_usage*), 57
password (*models.OldUser attribute*), 48
path (*models.Usage attribute*), 49
PATH (*resources.INDEX attribute*), 52
PHONE (*resources.INDEX attribute*), 52
platform (*track_usage.ParsedUserAgent property*), 57
plots-generate command line option
    --select, 67
    -s, 67
prcal() (*in module calendar*), 64
PREFIX (*events.EventCM attribute*), 39
PREFIX (*events.EventEXAM attribute*), 40
PREFIX (*events.EventEXTERN attribute*), 40
PREFIX (*events.EventLABO attribute*), 40
PREFIX (*events.EventORAL attribute*), 41
PREFIX (*events.EventOTHER attribute*), 41
PREFIX (*events.EventTP attribute*), 42
prettify_classrooms() (*in module classrooms*), 35
pretty_date_formatter() (*in module events*), 43
pretty_formatter() (*in module events*), 43
pretty_hour_formatter() (*in module events*), 43
prmonth() (*calendar.TextCalendar method*), 63
prmonth() (*in module calendar*), 64
process_bind_param() (*models.GUID method*), 47
process_result_value() (*models.GUID method*), 47
Professor (*class in professors*), 51
professors
    module, 51
prweek() (*calendar.TextCalendar method*), 63
pryear() (*calendar.TextCalendar method*), 63

Q

query_to_dataframe() (*in module models*), 50

R

RecurringCustomEvent (*class in events*), 42
redis-count command line option
    --code, 67
    --no-code, 67
    --no-session, 67
    --pattern, 67
    --session, 67
    -p, 67
redis-expire command line option
    --pattern, 68
    --random, 68
    --time, 68
    -p, 68
    -r, 68
    -t, 68
reformat_status_in_dataframe() (*in module models*), 50
remote_addr (*models.Usage attribute*), 49
remove_accents() (*in module events*), 43
remove_course() (*schedules.Schedule method*), 54
remove_custom_event() (*schedules.Schedule method*), 54
remove_filter() (*schedules.Schedule method*), 54
remove_prefix() (*in module events*), 43
remove_schedule() (*models.User method*), 50
renew_token() (*ade_api.Client method*), 29
renew_token() (*ade_api.DummyClient method*), 31
renew_token() (*ade_api.FakeClient method*), 31
request() (*ade_api.Client method*), 29
request() (*ade_api.DummyClient method*), 31
request() (*ade_api.FakeClient method*), 31
reset_best_schedules() (*schedules.Schedule method*), 55
reset_color_palette() (*schedules.Schedule method*), 55
reset_filters() (*schedules.Schedule method*), 55
Resource (*class in resources*), 52
resources
    module, 51
response_to_classrooms() (*in module ade_api*), 32
response_to_course_resources() (*in module ade_api*), 33
response_to_courses() (*in module ade_api*), 33
response_to_events() (*in module ade_api*), 33
response_to_project_ids() (*in module ade_api*), 33
response_to_resource_ids() (*in module ade_api*), 34

response_to_resources() (*in module ade_api*), 34
response_to_root() (*in module ade_api*), 34
Role (*class in models*), 48
roles (*models.User attribute*), 50
room_to_classroom() (*in module ade_api*), 34
run() (*servers.Server method*), 57

S

sanitize_string() (*in module events*), 43
save() (*in module account*), 59
save_geo_locations() (*in module classrooms*), 36
save_ics_url() (*manager.Manager method*), 45
save_response() (*in module ade_api*), 34
save_schedule() (*manager.Manager method*), 45
Schedule (*class in models*), 48
Schedule (*class in schedules*), 53
schedule (*models.Link attribute*), 47
schedule_id (*models.Link attribute*), 48
schedule_shield() (*in module api*), 61
ScheduleDoNotMatchError, 48
ScheduleNotFountError, 46
schedules
    module, 53
schedules (*models.OldUser attribute*), 48
schedules (*models.User attribute*), 50
schedules-stats command line option
    -m, 69
    -n, 69
security
    module, 55, 65
Server (*class in servers*), 56
servers
    module, 56
set_autosave() (*models.User method*), 50
set_cookie() (*in module cookies*), 36
set_custom_event_attributes() (*schedules.Schedule method*), 55
set_last_schedule_id() (*models.User method*), 50
set_oauth_token() (*in module cookies*), 36
set_option() (*schedules.Schedule method*), 55
set_value() (*servers.Server method*), 57
set_weight() (*events.CustomEvent method*), 39
set_weights() (*courses.Course method*), 37
setfirstweekday() (*calendar.Calendar method*), 62
setfirstweekday() (*in module calendar*), 64
share_schedule_with_emails() (*models.User method*), 50
shutdown() (*servers.Server method*), 57
SIZE (*resources.INDEX attribute*), 52
speed (*models.ApiUsage attribute*), 46
speed (*models.Usage attribute*), 49
sql-dump command line option
    --output, 69
    -o, 69
sql-load command line option
    --input, 70
    -i, 70
STATE (*resources.INDEX attribute*), 52
status (*models.ApiUsage attribute*), 46
status (*models.Usage attribute*), 49

T

table_to_dataframe() (*in module models*), 50
TEACHER (*resources.TYPES attribute*), 53
TextCalendar (*class in calendar*), 63
timegm() (*in module calendar*), 64
TIMEZONE (*resources.INDEX attribute*), 52
token (*uclouvain_apis.API property*), 58
TOKEN_URL (*uclouvain_apis.API property*), 58
track_usage
    module, 57
track_var (*models.Usage attribute*), 49
transparent (*events.EventCM attribute*), 39
transparent (*events.EventEXAM attribute*), 40
transparent (*events.EventEXTERN attribute*), 40
transparent (*events.EventLABO attribute*), 41
transparent (*events.EventORAL attribute*), 41
transparent (*events.EventOTHER attribute*), 41
transparent (*events.EventTP attribute*), 42
transparent (*events.RecurringCustomEvent attribute*), 43
truncate_string() (*models.Usage method*), 49
TYPE (*resources.INDEX attribute*), 52
TYPES (*class in resources*), 52

U

ua_browser (*models.Usage attribute*), 49
ua_language (*models.Usage attribute*), 49
ua_platform (*models.Usage attribute*), 49
ua_version (*models.Usage attribute*), 49
uclouvain_apis
    module, 58
uid (*events.EventCM attribute*), 39
uid (*events.EventEXAM attribute*), 40
uid (*events.EventEXTERN attribute*), 40
uid (*events.EventLABO attribute*), 41
uid (*events.EventORAL attribute*), 41
uid (*events.EventOTHER attribute*), 41
uid (*events.EventTP attribute*), 42
uid (*events.RecurringCustomEvent attribute*), 43
update_classrooms() (*manager.Manager method*), 45
update_course_resources() (*manager.Manager method*), 45
update_data() (*models.Schedule method*), 48
update_label() (*in module account*), 59
update_label() (*models.Schedule method*), 48
update_last_modified_by() (*models.Schedule method*), 48

update_project_ids() (*manager.Manager method*), 45
update_resource_ids() (*manager.Manager method*), 45
update_resources() (*manager.Manager method*), 45
update_url() (*models.ExternalCalendar method*), 46
url (*events.EventCM attribute*), 39
url (*events.EventEXAM attribute*), 40
url (*events.EventEXTERN attribute*), 40
url (*events.EventLABO attribute*), 41
url (*events.EventORAL attribute*), 41
url (*events.EventOTHER attribute*), 42
url (*events.EventTP attribute*), 42
url (*events.RecurringCustomEvent attribute*), 43
url (*models.ApiUsage attribute*), 46
url (*models.ExternalCalendar attribute*), 46
url (*models.Usage attribute*), 49
URL (*resources.INDEX attribute*), 52
url (*uclouvain_apis.API property*), 58
url_args (*models.Usage attribute*), 49
Usage (*class in models*), 48
usage-plot-requests-per-blueprint-hist command line option
    --latest, 70
usage-plot-views-per-blueprint-hist command line option
    --latest, 71
User (*class in models*), 49
user_id (*models.ExternalCalendar attribute*), 46
user_shield() (*in module api*), 61
UserMixin (*class in mixins*), 46
username (*models.Usage attribute*), 49
users (*models.Role attribute*), 48
users (*models.Schedule attribute*), 48
utils
    module, 65

V

version (*track_usage.ParsedUserAgent property*), 57
view_args (*models.Usage attribute*), 49

W

weekday() (*in module calendar*), 64
weekheader() (*in module calendar*), 64
whatisnew
    module, 65

Y

yeardatescalendar() (*calendar.Calendar method*), 62
yeardays2calendar() (*calendar.Calendar method*), 62
yeardayscalendar() (*calendar.Calendar method*), 62

Z

ZIP_CODE (*resources.INDEX attribute*), 52