MOST Web Documentation

Release 1.0.1

Healthcare Flows - CRS4

May 10, 2017

Contents

The MOST project aims to achieve an open, modular and scalable solution for the creation, execution and management of remote clinical consultations with direct interaction between specialists.

The project consists of a set of frameworks that deal with different aspects and technologies useful for the creation of telemedicine applications.

In addition to the frameworks, the project contains a number of helper modules, that should ease developers to build telemedicine applications focusing only on high value-added functionality.

CHAPTER 1

Frameworks

- Voip: a fast and lightweight library created for handling VOIP sessions;
- Streaming: a library for managing audio/video streams;
- Visualization: a library for providing mobile applications with visual widgets for interacting with A/V streams;
- Demographics: a Django application for patients management;
- Report: a library for managing clinical models;
- MedicalRecords: a frontend REST api for accessing to demographics and clinical data of patients;
- Demo: a DEMO Application showing the main features of the MOST frameworks.

CHAPTER 2

Helpers

- *MOST User*: a Django application for creation and management of administrative users;
- *Clinician User*: a Django application for creation and management of clinician users;
- *Task Group*: a Django application for creation and management of technical and clinician task groups;
- *Authentication*: a consumer/producer library for user authentication based on oauth2 protocol.

CHAPTER 3

Project TOC

Modules

The following are external framework of the project:

Helpers

Installation

Users Installation

Users is a Django application, indipendent from any Django project.

To use **Users** in your project, edit your settings.py file adding it to INSTALLED_APPS and properly setting AUTH_USER_MODEL, LOGIN_URL and LOGOUT_URL as follow:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # ...
    # your apps go here
    # ...
    'users',
)
# ...
AUTH_USER_MODEL = 'users.MostUser'
```

```
LOGIN_URL = '/users/user/login/'
LOGOUT_URL = '/users/user/logout/'
```

Authentication Installation

Task Group

Task Group is a Django module for creation and management of technical and clinician task groups.

Two types of task groups are provided:

- **Service Provider**: group designed to manage the system;
- **Health Care Facility**: group with clinical skill.

An Health Care Facility can provide specialized consultation to other Health Care Facility.

APIs

POST /users/task_group/new/
Create new task group

Parameters

- **title** (*str*) – the task group title. Max length 100
- **description** (*str*) – the task group description. Max length 100
- **task_group_type** (*str*) – the task group type: 'SP' for Service Provider and 'HF' for Health Care Facilities
- **hospital** (*str*) – the task group hospital. Max length 100
- **users** (*array*) – the list of users that belong to task group
- **is_health_care_provider** (*boolean*) – True if the task group is health care provider, False otherwise.
- **is_active** (*boolean*) – True if the user is active, False otherwise.
- **related_task_groups** (*array*) – the list of task groups that may benefit of health care services

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the task group is successfully created. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the created task group data in json format

GET /users/task_group/search/
Get a list of task group matching a query string in fields: title, description or hospital

Parameters

- **query_string** (*str*) – the query string to search

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

  parameter boolean *success* True if task groups matching the query string are found. False otherwise

  parameter str *message* a feedback string that would be displayed to the connected user

  parameter str *errors* an error string that explains the raised problems

  parameter json *data* if success is True, it contains the a list of data of task groups matching the query string, in json format

POST /users/task_group/ (*task_group_id*) **/edit/**
Edit the information of the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

  parameter boolean *success* True if the task group is successfully found and updated. False otherwise

  parameter str *message* a feedback string that would be displayed to the connected user

  parameter str *errors* an error string that explains the raised problems

  parameter json *data* if success is True, it contains the updated data of task group identified by *task_group_id*, in json format

GET /users/task_group/list_available_states/
Get a list of available state of activation

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

  parameter boolean *success* True if states are successfully found. False otherwise

  parameter str *message* a feedback string that would be displayed to the connected user

  parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains an array of available activation states in json format

POST /users/task_group/ (*task_group_id*) **/set_active_state/** *active_state***/** Set the activation state *active_state* (active or inactive) to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found and updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *id* (for the task group id) and *is_active* (for the activation state), in json format

GET /users/task_group/ (*task_group_id*) **/is_provider/**
Investigate if the task group identified by *task_group_id* is health care provider

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the task group is successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *id* (for the task group id) and *is_health_care_provider* (for the health care provider state)

POST /users/task_group/ (*task_group_id*) **/set_provider/**
Set the task group identified by *task_group_id* as health care provider

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found and updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *id* (for the task group id) and *is_active* (for the activation state), in json format

POST /users/task_group/ (*task_group_id*) **/add_user/**
user_id/ Add the user identified by *user_id* to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *user_id* (for the user just added to the task group), in json format

POST /users/task_group/ (*task_group_id*) **/remove_user/**
user_id/ Remove the user identified by *user_id* from the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *user_id* (for the user just removed from the task group), in json format

GET /users/task_group/ (*task_group_id*) **/list_users/**
List all users that belong to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains an array of data of users that belong to the task group, in json format

POST /users/task_group/ (*task_group_id*) **/add_related_task_group/** *related_task_group_id***/** Add the related task group identified by *related_task_group_id* to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *related_task_group_id* (for the related task group just added to the task group), in json format

POST /users/task_group/ (*task_group_id*) **/remove_related_task_group/** *related_task_group_id***/** Remove the related task group identified by *related_task_group_id* from the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *related_task_group_id* (for the related task group just removed from the task group), in json format

GET /users/task_group/ (*task_group_id*) **/list_related_task_groups/** List all related task groups that belong to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains an array of data of related task groups that belong to the task group, in json format

GET /users/task_group/ (*task_group_id*) **/has_clinicians/**
Investigate if the task group identified by *task_group_id* has clinician users

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if clinician users are successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *clinicians_count* (for the number of clinician user that belong to task group)

GET /users/task_group/ (*task_group_id*) **/list_clinicians/**
List all related clinician users that belong to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains an array of data of clinician users that belong to the task group, in json format

GET /users/task_group/ (*task_group_id*) **/has_clinician_provider/**
Investigate if the task group identified by *task_group_id* has health care provider clinician users

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if health care providers are successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *task_group_id* (for the task group id) and *clinicians_count* (for the number of health care provider clinician user that belong to task group)

GET /users/task_group/ (*task_group_id*) **/list_clinician_providers/**
List all health care provider clinician users that belong to the task group identified by *task_group_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if task group is found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains an array of data of health care provider clinician users that belong to the task group, in json format

Examples

Run the following login API before run others:

```
# -*- coding: utf-8 -*-

# API: /users/user/login/

from helper import *


USER_DATA = {
    'username': 'admintest',
    'password': 'admintest',
}

user = compose_post_request('/users/user/login/', USER_DATA)
print_response_data('user', user)
```

TaskGroup module provides the following **REST API** (run login api before run the following):

- /users/task_group/search/
- /users/task_group/new/
- /users/task_group/(?P<task_group_id>\d+)/get_task_group_info/
- /users/task_group/(?P<task_group_id>\d+)/edit/
- /users/task_group/list_available_states/

- /users/task_group/(?P<task_group_id>\d+)/set_active_state/(?P<active_state>\w+)/
- /users/task_group/(?P<task_group_id>\d+)/is_provider/
- /users/task_group/(?P<task_group_id>\d+)/set_provider/
- /users/task_group/(?P<task_group_id>\d+)/add_user/(?P<user_id>\d+)/
- /users/task_group/(?P<task_group_id>\d+)/remove_user/(?P<user_id>\d+)/
- /users/task_group/(?P<task_group_id>\d+)/list_users/
- /users/task_group/(?P<task_group_id>\d+)/add_related_task_group/(?P<related_task_group_id>\d+)/
- /users/task_group/(?P<task_group_id>\d+)/remove_related_task_group/(?P<related_task_group_id>\d+)/
- /users/task_group/(?P<task_group_id>\d+)/list_related_task_groups/
- /users/task_group/(?P<task_group_id>\d+)/has_clinicians/
- /users/task_group/(?P<task_group_id>\d+)/list_clinicians/
- /users/task_group/(?P<task_group_id>\d+)/has_clinician_provider/
- /users/task_group/(?P<task_group_id>\d+)/list_clinician_providers/

```
# -*- coding: utf-8 -*-

# API: /users/task_group/search/

from helper import *

QUERY_STRING = 'CRS'

task_groups = compose_get_request('/users/task_group/search/', QUERY_STRING)
print_response_data('task_group', task_groups)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/new/

from helper import *

TASK_GROUP_DATA = {
    'title': 'Notebook group 7',
    'description': 'Example task group, create by notebook user interface',
    'task_group_type': 'HF',
    'is_active': True,
    'is_health_care_provider': True,
}

task_group = compose_post_request('/users/task_group/new/', TASK_GROUP_DATA)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/get_task_group_info/

from helper import *
```

```
TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/get_task_group_info/' % TASK_
↪GROUP_ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/edit/

from helper import *

TASK_GROUP_ID = 2
TASK_GROUP_DATA = {
    "description": "Generici",
    "hospital": "Clinica d'esempio",
    "id": "2",
    "is_active": True,
    "is_health_care_provider": False,
    "task_group_type": "HF",
    "title": "Clinica"
}

task_group = compose_post_request('/users/task_group/%d/edit/' % TASK_GROUP_ID, TASK_
↪GROUP_DATA)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/list_available_states/

from helper import *

task_group = compose_get_request('/users/task_group/list_available_states/')
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/set_active_state/(?P<active_state>
↪\w+)/

from helper import *

TASK_GROUP_ID = 2
TASK_GROUP_ACTIVATION_STATE = 'inactive'

task_group = compose_post_request('/users/task_group/%d/set_active_state/%s/' % (TASK_
↪GROUP_ID, TASK_GROUP_ACTIVATION_STATE))
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/is_provider/

from helper import *
```

```
TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/is_provider/' % TASK_GROUP_ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/set_provider/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_post_request('/users/task_group/%d/set_provider/' % TASK_GROUP_
↪ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/add_user/(?P<user_id>\d+)/

from helper import *

TASK_GROUP_ID = 2
USER_ID = 3

task_group = compose_post_request('/users/task_group/%d/add_user/%d/' % (TASK_GROUP_
↪ID, USER_ID))
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/remove_user/(?P<user_id>\d+)/

from helper import *

TASK_GROUP_ID = 2
USER_ID = 3

task_group = compose_post_request('/users/task_group/%d/remove_user/%d/' % (TASK_
↪GROUP_ID, USER_ID))
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/list_users/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/list_users/' % TASK_GROUP_ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/add_related_task_group/(?P<related_
↪task_group_id>\d+)/

from helper import *

TASK_GROUP_ID = 2
RELATED_TASK_GROUP_ID = 26

task_group = compose_post_request('/users/task_group/%d/add_related_task_group/%d/' %␣
↪(TASK_GROUP_ID, RELATED_TASK_GROUP_ID))
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/remove_related_task_group/(?P<related_
↪task_group_id>\d+)/

from helper import *

TASK_GROUP_ID = 2
RELATED_TASK_GROUP_ID = 26

task_group = compose_post_request('/users/task_group/%d/remove_related_task_group/%d/
↪' % (TASK_GROUP_ID, RELATED_TASK_GROUP_ID))
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/list_related_task_groups/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/list_related_task_groups/' %␣
↪TASK_GROUP_ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/has_clinicians/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/has_clinicians/' % TASK_GROUP_
↪ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/list_clinicians/
```

```
from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/list_clinicians/' % TASK_GROUP_
↪ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/has_clinician_provider/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/has_clinician_provider/' %␣
↪TASK_GROUP_ID)
print_response_data('task_group', task_group)
```

```
# -*- coding: utf-8 -*-

# API: /users/task_group/(?P<task_group_id>\d+)/list_clinician_providers/

from helper import *

TASK_GROUP_ID = 2

task_group = compose_get_request('/users/task_group/%d/list_clinician_providers/' %␣
↪TASK_GROUP_ID)
print_response_data('task_group', task_group)
```

Now you can run logout API:

```
# -*- coding: utf-8 -*-

# API: /users/user/logout/

from helper import *

response_content = compose_get_request('/users/user/logout/')
print_response_data('user', response_content)
```

MOST User

MOST User is a Django module for creation and management of users.

Four types of task groups are provided:

- **Administrative**: user designed to perform administrative duties;
- **Technician**: user designed to perform technical duties;
- **Clinician**: user designed to perform clinical duties;
- **Student**: user designed to interact with clinicians and see clinical reports.

APIs

POST /users/user/new/
Create new user.

Parameters

- **username** (*str*) – the user nickname. Max length 30
- **first_name** (*str*) – the user first name. Max length 50
- **last_name** (*str*) – the user last name. Max length 50
- **email** (*str*) – the user email. Max length 250
- **birth_date** (*datetime*) – the user birth date
- **is_staff** (*boolean*) – True if the user is staff member, False otherwise. Default to True
- **is_active** (*boolean*) – True if the user is active, False otherwise. Default to True
- **is_admin** (*boolean*) – True if the user has administrator privileges. Default to False
- **numeric_password** (*int*) – the user pin. String of 4 numbers
- **user_type** (*str*) – the user type: 'AD' for Administrative, 'TE' for Technician, 'CL' for Clinician, 'ST' for Student
- **gender** (*str*) – the user gender: 'M' for Male, 'F' for Female, 'U' for Unknown
- **phone** (*str*) – the user phone number. Max length 20
- **mobile** (*str*) – the user mobile phone number. Max length 20
- **certified_email** (*str*) – the user legal mail. Max length 255

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully created. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the created user data in json format

POST /users/user/login/
Log a user in the system

Parameters

- **username** (*str*) – the user nickname
- **password** (*str*) – the user password

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully logged in. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the logged user data in json format

GET /users/user/logout/
Log a user out of the system

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully logged out. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

GET /users/user/ (*user_id*) **/get_user_info/**
Get the information of the user identified by *user_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the data of user identified by *user_id*, in json format

GET /users/user/search/
Get a list of users matching a query string in fields: username, last_name, first_name, email or certified_email

Parameters

- **query_string** (*str*) – the query string to search

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if users matching the query string are found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the a list of data of users matching the query string, in json format

POST /users/user/ (*user_id*) **/edit/**

Edit the information of the user identified by *user_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully found and updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the updated data of user identified by *user_id*, in json format

POST /users/user/ (*user_id*) **/deactivate/**

Deactivate the user identified by *user_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully deactivated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *id* (for the user id) and *is_active* (for the activation state):

POST /users/user/ (*user_id*) **/activate/**

Activate the user identified by *user_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the user is successfully activated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *id* (for the user id) and *is_active* (for the activation state)

Examples

Run the following login API before run others:

```
# -*- coding: utf-8 -*-

# API: /users/user/login/

from helper import *


USER_DATA = {
    'username': 'admintest',
    'password': 'admintest',
}

user = compose_post_request('/users/user/login/', USER_DATA)
print_response_data('user', user)
```

MostUser module provides the following **REST API** (run login api before run the following):

- `/users/user/new/`
- `/users/user/(?P<user_id>\d+)/get_user_info/`
- `/users/user/search/`
- `/users/user/(?P<user_id>\d+)/edit/`
- `/users/user/(?P<user_id>\d+)/deactivate/`
- `/users/user/(?P<user_id>\d+)/activate/`

```
# -*- coding: utf-8 -*-

# API: /users/user/new/

from helper import *


USER_DATA = {
    'username': 'mario.rossi',
    'first_name': 'Mario',
    'last_name': 'Rossi',
    'email': 'mario.rossi@most.crs4.it',
    'birth_date': '1980-07-08',
    'is_active': True,
    'is_admin': False,
```

```
    'numeric_password': 1234,
    'user_type': 'CL',
    'gender': 'M',
    'phone': '070789456',
    'mobile': '888987654',
}

user = compose_post_request('/users/user/new/', USER_DATA)
print_response_data('user', user)
```

```
# -*- coding: utf-8 -*-

# API: /users/user/(?P<user_id>\d+)/get_user_info/

from helper import *

USER_ID = 1

user = compose_get_request('/users/user/%d/get_user_info/' % USER_ID)
print_response_data('user', user)
```

```
# -*- coding: utf-8 -*-

# API: /users/user/search/

from helper import *

QUERY_STRING = 'test'

users = compose_get_request('/users/user/search/', QUERY_STRING)
print_response_data('user', users)
```

```
# -*- coding: utf-8 -*-

# API: /users/user/(?P<user_id>\d+)/edit/

from helper import *

USER_ID = 1
USER_DATA = {
    'username': 'valeria',
    'first_name': 'Valeria',
    'last_name': 'Lecca',
    'email': 'valeria.lecca@most.crs4.it',
    'birth_date': '1980-06-11',
    'is_active': True,
    'is_admin': True,
    'numeric_password': 1234,
    'user_type': 'TE',
    'gender': 'F',
    'phone': '070789456',
    'mobile': '888987654',
}

user = compose_post_request('/users/user/%d/edit/' % USER_ID, USER_DATA)
print_response_data('user', user)
```

```
# -*- coding: utf-8 -*-

# API: /users/user/(?P<user_id>\d+)/deactivate/

from helper import *

USER_ID = 9

user = compose_get_request('/users/user/%d/deactivate/' % USER_ID)
print_response_data('user', user)
```

```
# -*- coding: utf-8 -*-

# API: /users/user/(?P<user_id>\d+)/activate/

from helper import *

USER_ID = 9

user = compose_post_request('/users/user/%d/activate/' % USER_ID)
print_response_data('user', user)
```

Now you can run logout API:

```
# -*- coding: utf-8 -*-

# API: /users/user/logout/

from helper import *

response_content = compose_get_request('/users/user/logout/')
print_response_data('user', response_content)
```

Clinician User

Clinician User is a Django module for creation and management of clinicians.

Two types of clinician users are provided:

- **Doctor**: user designed to provide specialized consultation to other Clinician Users (Doctor or Operator);
- **Operator**: user designed to consume specialized consultation of an other Clinician User (Doctor).

APIs

POST /users/clinician_user/new/
Create new clinician user.

Parameters

- **user** (*int*) – the user id of the related user
- **clinician_type** (*str*) – the clinician user type: 'DR' for Doctor or 'OP' for Operator
- **specialization** (*str*) – the clinician user specialization. Max length 50

- **is_health_care_provider** (*boolean*) – True if the clinician user is health care provider, False otherwise

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the clinician user is successfully created. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the created user data in json format

GET /users/clinician_user/ (*user_id*) **/is_provider/**

Investigate if the clinician user, with related user identified by *user_id*, is health care provider

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the clinician user is successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *user_id* (for the related user id) and *is_health_care_provider* (for the health care provider state)

POST /users/clinician_user/ (*user_id*) **/set_provider/**

Set the clinician user, with related user identified by *user_id*, health care provider state to True

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the clinician user is successfully updated. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the keys *user_id* (for the related user id) and *is_health_care_provider* (for the health care provider state)

GET /users/clinician_user/search/
Get a list of clinician users matching a query string in fields: username, last_name, first_name, email, certified_email or specialization

Parameters

- **query_string** (*str*) – the query string to search

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if clinician users matching the query string are found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the a list of data of clinician users matching the query string, in json format

GET /users/clinician_user/ (*user_id*) **/get_user_info/**
Get the information of the clinician user, with related user identified by *user_id*

Request Headers

- Authorization – login required

Response Headers

- Content-Type – application/json

parameter boolean *success* True if the clinician user is successfully found. False otherwise

parameter str *message* a feedback string that would be displayed to the connected user

parameter str *errors* an error string that explains the raised problems

parameter json *data* if success is True, it contains the data of clinician user, with related user identified by *user_id*, in json format

Examples

Run the following login API before run others:

```
# -*- coding: utf-8 -*-

# API: /users/user/login/

from helper import *


USER_DATA = {
    'username': 'admintest',
    'password': 'admintest',
}
```

```
user = compose_post_request('/users/user/login/', USER_DATA)
print_response_data('user', user)
```

ClinicianUser module provides the following **REST API** (run login api before run the following):

- /users/clinician_user/(?P<user_id>\d+)/is_provider/
- /users/clinician_user/(?P<user_id>\d+)/set_provider/
- /users/clinician_user/search/
- /users/clinician_user/(?P<user_id>\d+)/get_user_info/

```
# -*- coding: utf-8 -*-

# API: /users/clinician_user/(?P<user_id>\d+)/is_provider/

from helper import *

USER_ID = 2

clinician_user = compose_get_request('/users/clinician_user/%d/is_provider/' % USER_
↪ID)
print_response_data('clinician_user', clinician_user)
```

```
# -*- coding: utf-8 -*-

# API: /users/clinician_user/(?P<user_id>\d+)/set_provider/

from helper import *

USER_ID = 2

clinician_user = compose_post_request('/users/clinician_user/%d/set_provider/' % USER_
↪ID)
print_response_data('clinician_user', clinician_user)
```

```
# -*- coding: utf-8 -*-

# API: /users/clinician_user/search/

from helper import *

QUERY_STRING = 'test'

clinician_user = compose_get_request('/users/clinician_user/search/', QUERY_STRING)
print_response_data('clinician_user', clinician_user)
```

```
# -*- coding: utf-8 -*-

# API: /users/clinician_user/(?P<user_id>\d+)/get_user_info/

from helper import *

USER_ID = 2

clinician_user = compose_get_request('/users/clinician_user/%d/get_user_info/' % USER_
↪ID)
```

```
print_response_data('clinician_user', clinician_user)
```

Now you can run logout API:

```
# -*- coding: utf-8 -*-

# API: /users/user/logout/

from helper import *

response_content = compose_get_request('/users/user/logout/')
print_response_data('user', response_content)
```

Authentication

Authentication is performed using OAuth2 protocol. The implementation is based on django-oauth2-providers Django application so for generale documentation overview refer to the official documentation. Here we only explain the difference of the MOST implementation.

MOST implementation requires also the *taskgroup* field to get an access_token in the REST call.

REST API

POST /oauth2/access_token/

It authenticates the client providing an access token which can be used in subsequent REST calls. The request data are:

- client_id: the id of an OAuth2 client as configured in database
- client_secret: the client secret,
- grant_type: password|pincode,
- username: the username,
- password|pincode: the password or the pincode depending on the grant_type value,
- taskgroup: the id of the taskgroup of the user

Request Headers

- Content-Type – application/json

Response Headers

- Content-Type – application/json

Example of correct response

```
{"access_token": "25735f2de89eff6fa7576b7f8ca5efb952a23ef7",
 "expires_in": 31535999,
 "refresh_token": "e2a310b568aba5f92bf8683715f862b98b1d714d",
 "scope": "read",
 "token_type": "Bearer"
 }
```

License

```
/*!
 * Project MOST - Moving Outcomes to Standard Telemedicine Practice
 * http://most.crs4.it/
 *
 * Copyright 2014, CRS4 srl. (http://www.crs4.it/)
 * Dual licensed under the MIT or GPL Version 2 licenses.
 * See license-GPLv2.txt or license-MIT.txt
 */
```

GPL2: https://www.gnu.org/licenses/gpl-2.0.txt

MIT: http://opensource.org/licenses/MIT

Authors

Code author: Francesco Cabras <francesco.cabras@crs4.it>

Code author: Valeria Lecca <valeria.lecca@crs4.it>

Code author: Stefano Leone Monni <stefano.monni@crs4.it>

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

HTTP Routing Table

/oauth2

/users