Area API Documentation

Anas Buyumad, Antoine Foret, Gauthier Cler, Valentin Montagne and Vincent Dusautoir students of Epitech Lille are proud to present ~~THE MARAUDER'S MAP~~ the AREA Project

Its Goal

The Area is a project whose goal is to propose a **simple interface** allowing a user to **build modules** with pre-defined actions and reactions on different **services**.
These modules are based on an **action-reaction** pattern, meaning one can, for example, **post a facebook status** whenever something is **tweeted** on Twitter.

To achieve this, we put at disposition a list of pre-defined actions and reactions linked to services such as **Dropbox**, **Facebook** or even **Twitter**.
These modules can be created and configured on our **Area Platfom** or by using our **RESTFUL API**.

How To Use It

- Table of Contents

Registration

In order to **register** a new user, you have to execute a **POST** request on **/register**.

The **username** must be unique, please check-out the register parameters for more information.

You will receive a response containing your newly created **id** and **access token**.

Response Chart
200 OK -> Successful Registration
400 BAD REQUEST -> Badly Formatted Request
409 CONFLICT -> Conflicting Username

Login

In order to **login** as a user, you have to execute a **POST** request on **/login**.

Please check-out the login parameters for more information.

You will receive a response containing your **id** and **access token**.

Response Chart
200 OK -> Login Complete
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Username Or Password

Listing Users

In order to **list** users, you have to execute a **GET** request on **/users**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** to use this functionality.

You will receive a response containing a list of **users**.

Response Chart
200 OK -> List Of Users
403 FORBIDDEN -> Wrong Permissions

Sending A Mail

In order to **send a mail** to a user, you have to execute a **POST** request on **/mails**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** to use this functionality.
Please check-out the email parameters for more information.

You won't receive any response on completion.

Response Chart
200 OK -> Mail Sent
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Getting A User

In order to **retrieve** a user, you have to execute a **GET** request on **/user/:userId**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.

You will receive a response containing the requested **user** data.

Response Chart
200 OK -> User Information
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Removing A User

In order to **remove** a user, you have to execute a **DELETE** request on **/user/:userId**.
You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.

You won't receive any response on completion.

Response Chart
200 OK -> User Deleted
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Updating A User

In order to **update** a user, you have to execute a **POST** request on **/user/:userId**.
You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.

You won't receive any response on completion.

Response Chart
200 OK -> User Updated
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Associating An Account

In order to **associate** a service account to a user, you have to execute a **POST** request on **/user/:userId/{serviceName}**.
The **service name** can be either **dropbox**, **facebook** or **twitter**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.
Please check-out the dropbox parameters, facebook parameters or twitter parameters for more information.

You won't receive any response on completion.

Response Chart
200 OK -> Account Associated
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Disassociating An Account

In order to **disassociate** a service account to a user, you have to execute a **DELETE** request on **/user/:userId/{serviceName}**.
The **service name** can be either **dropbox**, **facebook** or **twitter**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.

You won't receive any response on completion.

Response Chart
200 OK -> Account Disassociated
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Enabling A Module

In order to **enable** a module linked to a user, you have to execute a **POST** request on **/user/:userId/modules**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality. Please check-out the module parameters for more information.

You won't receive any response on completion.

Response Chart
200 OK -> Module Enabled
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Disabling A Module

In order to **disable** a module linked to a user, you have to execute a **DELETE** request on **/user/:userId/modules**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality. Please check-out the module parameters for more information.

You won't receive any response on completion.

Response Chart
200 OK -> Module Disabled
400 BAD REQUEST -> Badly Formatted Request
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Listing Modules

In order to **list** the different modules enabled for a user, you have to execute a **GET** request on **user/:userId/modules**.

You have to give your **authentication token** in the **token** field of the headers.
Please note that you have to be **admin** or the **owner** of the id to use this functionality.

You won't receive any response on completion.

Response Chart
200 OK -> List Of Enabled Modules
403 FORBIDDEN -> Wrong Permissions
404 NOT FOUND -> User Not Found

Listeners

Below is the **list** of the different **listeners** available to the users that can be used to create **modules** along with their description.

LISTENER NAME	DESCRIPTION
DropboxFileAddedEvent	File is created in Dropbox
DropboxFileRemovedEvent	File is removed in Dropbox
DropboxFolderCreatedEvent	Folder is create in Dropbox
FacebookAboutEvent	About is changed on Facebook
FacebookBirthdayEvent	Birthday is changed on Facebook
FacebookEmailEvent	Email is changed on Facebook
FacebookFirstNameEvent	First name is changed on Facebook
FacebookLastNameEvent	Last name is changed on Facebook
FacebookLocaleEvent	Locale is changed on Facebook
FacebookPictureEvent	Picture is changed on Facebook
FacebookQuotesEvent	Quotes is changed on Facebook
FacebookReligionEvent	Religion is changed on Facebook
FacebookStatusEvent	A new status is posted on Facebook

LISTENER NAME	DESCRIPTION
FacebookWebsiteEvent	Website is changed on Facebook
TwitterStatusEvent	A new status is tweeted on Tweeter

Handlers

Below is the **list** of the different **handlers** available to the users that can be used to create **modules** along with their description.

HANDLER NAME	DESCRIPTION
facebookOnTweet	Facebook action when a status is tweeted on Twitter
twitterOnFacebookEvent	Twitter action when an event occurs on Facebook
twitterOnFacebookBirthday	Twitter action when birthday changes on Facebook
twitterOnFacebookEmail	Twitter action when email changes on Facebook
twitterOnFacebookFirstName	Twitter action when first name changes on Facebook
twitterOnFacebookLastName	Twitter action when last name changes on Facebook
twitterOnFacebookPicture	Twitter action when picture changes on Facebook
twitterOnFacebookReligion	Twitter action when religion changes on Facebook
twitterOnFacebookQuote	Twitter action when quote changes on Facebook
twitterOnFacebookStatus	Twitter action when a new status is posted on Facebook
twitterOnFacebookWebsite	Twitter action when website changes on Facebook
twitterOnDropboxAddedFile	Twitter action when a file is added on Dropbox
twitterOnDropboxCreatedFolderFile	Twitter action when a folder is created on Dropbox
twitterOnDropboxRemovedFile	Twitter action when a file is removed on Dropbox
facebookOnDropboxAddedFile	Facebook action when a file is created on Dropbox
facebookOnDropboxCreatedFolderFile	Facebook action when a folder is created on Dropbox

HANDLER NAME	DESCRIPTION
facebookOnDropboxRemovedFile	Facebook action when a file is removed on Dropbox

How It Works

- Table of Contents

Controllers

> *I regulate actions defined by the responsibilities bestowed upon me.*

Security Controller

Is in charge of **securing** the access to the different **services** of the API.
Regulations will be made based on the ownership of the ressources queried along with the user's role.
Registration and **log-in** are also handled by this controller.

Users Controller

Handles every interaction with the different users.
Listing, **updating**, **sending mails** or even **deleting users** is done through this controller.

Modules Controller

Its goal is to allow the user to **associate** or **disassociate** itself with the different modules proposed by the API services.

Services Controllers (Dropbo x, Facebook, Twitter)

These controllers' purpose is to let the user **associate** or **disassociate** its account with the respective services.

Webhook Controllers (Dropbo x, Facebook)

These controllers' purpose is to **receive** information about subscribed changes of the associated service, and then **deliver** them to the associated pipeline.

Pipelines

> *I convey data to the proper receiving end.*

The pipelines are the backbone of the AREA projec t.

Indeed, their goal is to **receive** data about **new changes** from different services.
They **analyze** the data, **verify** its integrity, and then **associate** it to the proper user.
Once this is done, they **generate** an event containing both the data and the user it is associated to, which will then be treated by the different event listeners.

Events

> *It's happening !*

Dropbox Events

These event symbolize the **creation** and **deletion** of any file or folder.
They hold any associated data.

Facebook Events

These events symbolize any profile changes ranging from the **birthday** to the **religion** of a user, along with any **status** update.
They hold any associated data.

Twitter Events

These events symbolize any **status** changes that occured for a user.
They hold any associated data.

Listeners

> *I can hear the voice of all things.*

Event Factory

Its goal is to provide the interface needed for the creation of new events.
Creation and **invocation** of events is its duty.

Service Event Handlers

Any service has their associated **handler**, with **reactions** that will be **invoked** on a specific event ran by the event factory.

Models

> *I think, therefore I am.*

User

This model **represents** what a user is.
Every **attribute** such as its **username**, **password**, **tokens** and so on are stocked here, along with their associated methods.

UserModule

This model **represents** what a module is : a **connection** between a **handler** and a **listener**, making the creation of a **link** between an action and a reaction its goal.
Every **attribute** such as the **user**, **handlers** and **listeners** it is associated to are stocked here, along with their associated methods.

Log

This model **represents** what a log is : a representation in time of an **action**. Every **attribute** such as the **type**, **id** and **timestamp** are stocked here, along with their associated methods.

Parameters

> *I am what describes your nature.*

Register Parameters

Defines what is required when a user executes a registration request.
A **username**, **password** and an **email** are mandatory.

```
{
  "username":   "username",
  "password":   "password",
  "email":      "your.email@domain.com
}
```

Login Parameters

Defines what is required when a user executes a login request.
A **username** and a **password** are mandatory.

```
{
  "username":   "username",
  "password":   "password"
}
```

Email Parameters

Defines what is required when a user wishes to send a mail.
A **subject** and a **message** are mandatory.

```
{
  "subject":    "subject",
  "message":    "message"
}
```

User Update Parameters

Defines what is required when a user wishes to update information about itself.
A **password** is mandatory, and a list of **roles** is optional.

```
{
  "password":   "password",
  "roles":      ["USER", "ADMIN", "..."]
}
```

Module Parameters

Defines what is required when a users wishes to link actions and reactions with a module.
A **listener** and a **handler** are mandatory.

```
{
  "listener":   "listener",
  "handler":    "handler"
}
```

Dropbox Parameters

Defines what is required when a users wishes to associate its Dropbox account.
An **authorization token** is mandatory.

```
{
  "accessToken":    "token"
}
```

Facebook Parameters

Defines what is required when a users wishes to associate its Facebook account.
An **authorization token** is mandatory.

```
{
  "accessToken":    "token"
}
```

Twitter Parameters

Defines what is required when a users wishes to associate its Twitter account.
An **authorization token** and a **secret access token** are mandatory.

```
{
  "accessToken":        "token",
  "accessTokenSecret":  "secretToken"
}
```

Responses

> *You wanted to know ?*

Token Response

Is what is sent following a **login** or **register** request.
It represents a user by its **id** and **token** fields that are used and required for many API operations.

```
{
  "id":     "id",
  "token":  "token"
}
```

User Response

Is what is sent follow a **user** or **list users** request. It represents a user by all its associated fields that are used and required for many API operations.

```
{
"id": 1,
"username": "username",
"password": "password",
"email": "your@email@domain.com" ,
"token": "token",
"roles": [
    "USER",
    "ADMIN",
    "..."
    ],
"facebookAccessToken" : "facebookToken",
"twitterAccessToken" : "twitterToken",
"twitterAccessTokenSecret" : "twitterSecretToken",
"dropboxAccessToken" : "dropboxToken",
"dropboxCursor" : "dropboxCursor",
"twitterID": "twitterID",
"facebookUUID" : "facebookUUID",
"dropboxID": "dropboxID",
"valid": true
}
```

Repositories

> *I hold the knowledge.*

Users Repository

Stores every user and provides a way to interact with any of them.
A multitude of methods are available, such as **finding a user** by its **id** or a **service token**, **logging-in**, **creating**, **updating** or **removing** a user, **sending an e-mail** and so on so that you can manipulate a user correctly.

Logs Repository

Holds a trace of every **log** (action) in **time** and provides a way to interact with any of them.
Its main goal is to **prevent** the **duplication** of actions, as webhooks and such services tend to send the same changes multiple times to ensure it was received.

UsersModule Repository

Stores every module created by a certain user.
Gathering **listeners** and **handlers**, it creates a way to **interact** and **configure** what a specific user wants to listen and react to.

Helpers

> *What can I do for you ?*

The different helpers provide methods or mechanisms to help you in your daily life.

Parameters Helper

Provides a method to help you **translate** a **JSON** object into its associated **class**.

Serializer Helper

Provides a method to help you **translate** a **class** into its associated **JSON** object

---

Header Helper

Provides **assistance** for every **configuration** that may be needed with the **headers** of any **request**.

Email Helper

Provides methods to help you **interact** with a **user** and **e-mails**.

Dropbox Helper

Provides methods to help you **interact** with a **Dropbox account** associated to a **user**.

Facebook Helper

Provides methods to help you **interact** with a **Facebook account** associated to a **user**.

Twitter Helper

Provides methods to help you **interact** with a **Twitter account** associated to a **user**.