B5 - Application Development

Action REAction

Automation platform of his digital life

MEMBRES : Thomas MARQUIS, Lorraine NETZER, Nicolas CLEMENCET,
Spyros RAPHAIL-KYPRIADIS, Tina ANDRIA MANDIMBY

Version : 4
23/03/2019

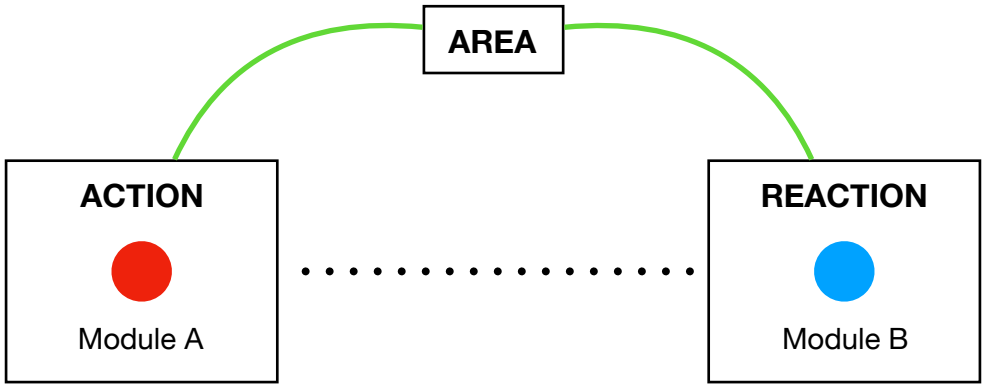
Sommaire

1) Fonctionnement de l'AREA

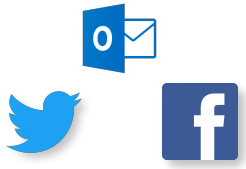
L'AREA est une application web qui permet des actions en ligne grâce à l'inter-connectivité de différents services web (FaceBook, Twitter, Outlook …). L'utilisateur peut se connecter ou s'inscrire sur des sites internets puis autoriser à l'application, l'accès aux différents services (modules). L'utilisateur peut donc créé des connexions entre ces services de la façon suivante :

1) Une action se produit sur un module A
2) Une réaction se produit sur un module B

En combinant ces 2 modules Action / REAction on obtient un **AREA**.



a - Environnement des technologies utilisées



Pour ce projet de liaison de services nous utilisons pour :

Server (API REST)	**NodeJs**
Mobile-client	**React Native**
Web-client	**NodeJs**

NodeJs est nécessaire pour developper un serveur. Utilisé dans un souci de simplicité, en fait plusieurs membres du groupe connaissent plutôt bien ce langage.

React Native pour sa praticité et sa simplicité de distribution (cross-plateforme) permettant une comptabilité Android / iOS.

Concernant le « ***User Management*** », les utilisateurs non authentifiés peuvent créer un compte sur une page de login. Une fois le compte créé, ils peuvent se connecter à l'application web.

Les utilisateurs peuvent directement gérer leurs connexions aux différents services (FaceBook, Twitter …) sur la page qui leur est dédié. Une fois que l'utilisateur se connecte et autorise à l'application AREA l'accès à son compte, les **tokens** d'accès seront sauvegardés dans une base de données qui permettra une connexion automatique à chaque utilisation de l'AREA.

Sur la page de gestion de l'AREA, l'utilisateur peut voir ce qui est possible, disponible ou bien indisponible sur l'application. En effet, les modules disponibles peuvent être créés et les modules indisponibles peuvent demander une autorisation d'accès. Par exemple pour FaceBook, le module ne sera pas disponible si l'utilisateur n'a pas autorisé l'application AREA à accéder aux informations de son compte FaceBook.

b - Description de l'API

Les **API REST** imitent la façon dont le web lui-même marche dans les échanges entre un client et un serveur.

API REST	
Cacheable (avec cache = mémoire)	Sans état
Orienté client - serveur	Système de couche
Interface uniforme	Code à la demande (optionnel)

Le principe du client - serveur définit les deux entités qui interagissent dans une API REST : un client et un serveur, les mêmes entités qui communiquent sur le web. Un client envoie une requête, et le serveur renvoie une réponse. Ce dernier doit avoir le plus d'informations possible sur le client, car il est important qu'ils soient capables de travailler indépendamment l'un de l'autre.

Le fait d'être "sans état" signifie que le serveur n'a aucune idée de l'état du client entre deux requêtes. Du point de vue du serveur, chaque requête est une entité distincte des autres. Ensuite, le cache, pour les API REST, met en jeu le même principe que pour le reste d'Internet : un client doit être capable de garder en mémoire des informations sans avoir constamment besoin de demander tout au serveur.

Les réponses du serveur pour les API REST peuvent être délivrées dans de multiples formats **.JSON** (JavaScript Object Notation) est souvent utilisé, mais XML, CSV, ou même RSS sont aussi valables.

AUTHENTIFICATION

POST signup

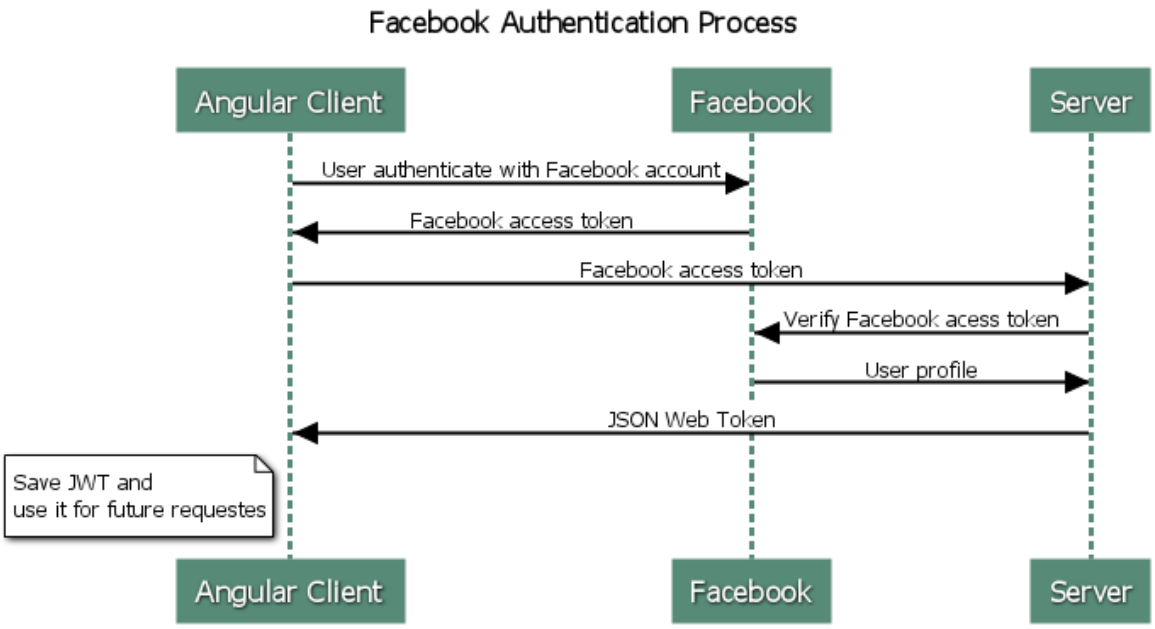
localhost:8080/api/auth/signup
> Return a bearer token { token: }

```
curl --location --request POST "localhost:8080/api/auth/signup" \
  --header "Content-Type: application/json" \
  --data "{
  \"email\": \"toto.tata@epitech.eu\",
  \"username\": \"user1\",
  \"password\":\"azerty1234\"
}"
```

POST signin

localhost:8080/api/auth/signin
> Return a bearer token { token: }

```
curl --location --request POST "localhost:8080/api/auth/signin" \
  --header "Content-Type: application/json" \
  --data "{
  \"email\": \"toto.tata@epitech.eu\",
  \"password\": \"azerty1234\"
}"
```



USER

Edit User
localhost:8080/api/users/

```
curl --location --request PATCH "localhost:8080/api/users/" \
  --header "Authorization: Bearer token" \
  --header "Content-Type: application/json" \
  --data "{
  \"email\": \"toto.tata@epitech.eu\",
  \"username\": \"user1\"
}"
```

Get User
localhost:8080/api/users/

```
curl --location --request GET "localhost:8080/api/users/" \
  --header "Authorization: Bearer token" \
  --data ""
```

Put User
localhost:8080/api/users

```
  curl --location --request PUT "localhost:8080/api/users" \
  --header "Authorization: Bearer token" \
  --header "Content-Type: application/json" \
  --data "{
  \"email\": \"TEST\",
  \"username\": \"TEST\",
  \"avatarUrl\": \"TEST\"
}"
```

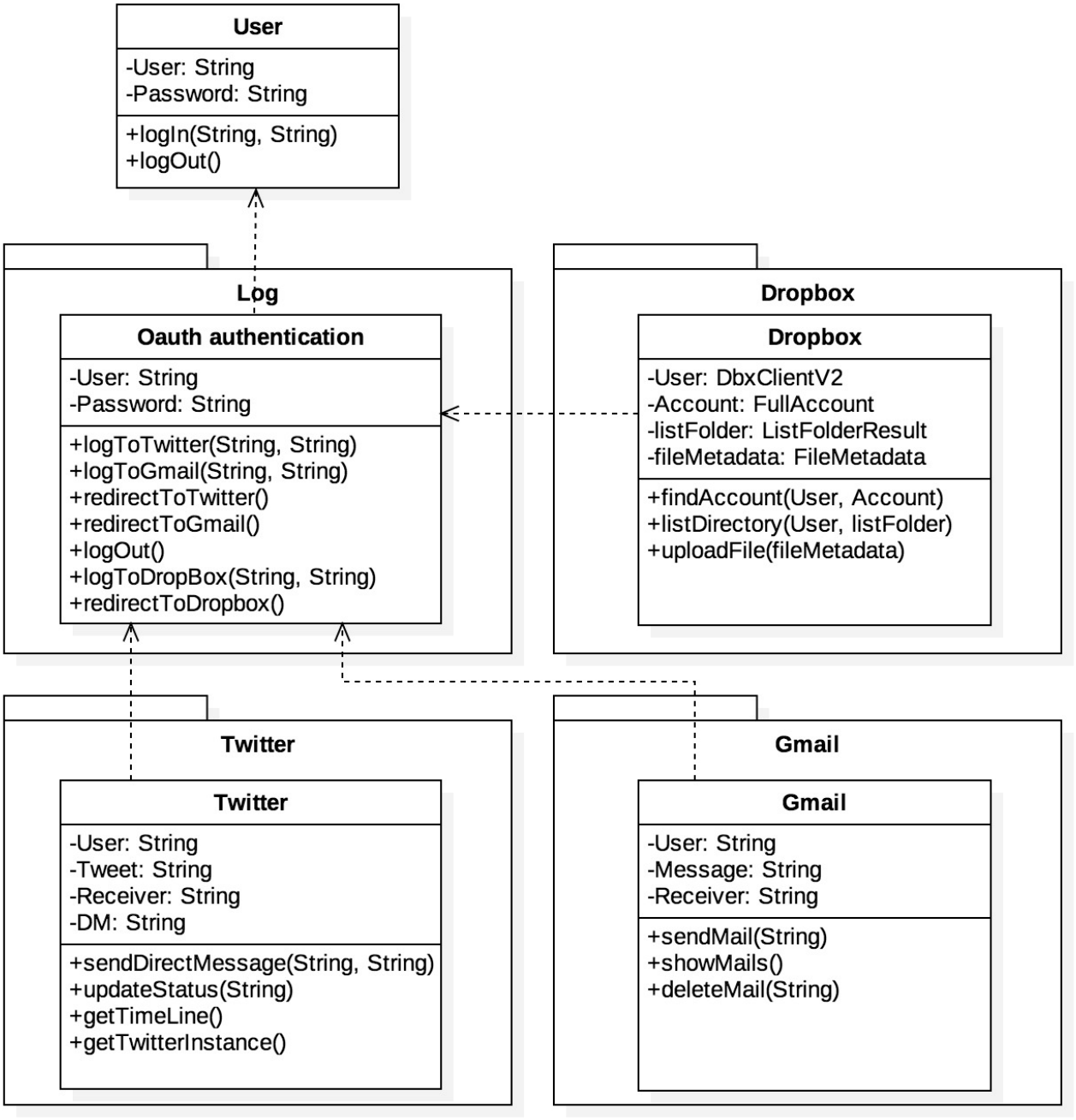
Get upload profile URL
localhost:8080/api/users/upload/profile

```
curl --location --request GET "localhost:8080/api/users/upload/profile" \
  --header "Authorization: Bearer token" \
  --data ""
```

2) Sequence & Class Diagram

Comme pour le projet *Dashboard*, tous les services découlent d'une classe *Service* et tous les widgets découlent d'une classe *Widget*. Chaque service a une clé d'API, une url de base ainsi que ses widgets. Chaque widget a une clé d'API, une url de base, un taux de rafraîchissement (défaut 60sec) et une fonction UpdateInfos.

Toutes les informations sont récupérées via des **API REST**. Une fonction *RequestApi* prenant en paramètre l'URL complète de la requête et une fonction callback permet de faire ces requêtes.



3) Modules de l'AREA

L'AREA est composé de 2 modules. Le but de ces modules est de permettre l'authentification de l'utilisateur via une simple connexion **OAuth1 / OAuth2**. Cette connexion à un service permet de récupérer un token.

Chaque utilisateur possède un token propre à chaque module.

Pour cela, il est implémenté dans un module : *`IService Interface`*.

```
public interface IService
{

  public String login(String username, String password);
  public String login(String code, Module module);

}
```

Cette méthode est utilisée pour une simple authentification :

```
public String login(String username, String password);
```

Celle-ci pour une authentification **OAuth2** :

```
public String login(String code, Module module);
```

La classe *Service* doit être nommée : « **Nom du service** » **+ Service**.

```
public class    FacebookService implements IService
```

a- Fixture des modules

Le projet AREA utilise une base de données *MongoDB*. A chaque ajout de module, le développeur doit implémenter une installation précise afin de l'intégrer à la base de données sous la forme d'un modèle module.

```
public void     init()
{
  this.add(new Module(moduleName, image path, description,
    url retrieve code, url get token));
}
```

L'installation doit se faire dans l'implémentation du init() dans la class *ModuleFixture*.

Le constructeur de la classe *Module* prend un string moduleName, un string image path, un string description, un string url retrieve code ainsi qu'un string url get token.

4) Les Actions / Réactions de l'AREA

L'AREA est donc composé d'une **ACTION** et d'une **REACTION**. Il utilise les tokens de l'authentification pour appeler le Service API.

Une action est implémentée via *`IAction Interface`* :

```
public interface    IAction
{
  ErrorCode     run();
  Object      getData();
}
```

La méthode run() est le point de départ de l'action. Cela va exécuter l'appel au Service API et créé Object en réponse. Puis va retourner ErrorCode Success si l'exécution fonctionne sans erreur ou que ErrorCode Auth n'est plus accès au token.

La méthode getData() va chercher la data de Object créé pendant la méthode run().

Concernant la réaction, celle-ci est implémentée via *`IReaction Interface`* :

```
public interface    IReaction
{
  public ErrorCode  run(Object data);
}
```

La méthode run() est le point de départ de la réaction. Cela va prendre Object data pendant la création de action run() et va exécuter les appels au Service API puis envoyer Object data. Ensuite cela va retourner ErrorCode Success si l'exécution fonctionne sans erreur ou que ErrorCode Auth n'est plus accès au token.

a- Fixture de l'AREA

Le projet AREA utilise une base de données *MongoDB*. A chaque ajout d'AREA, le développeur doit implémenter une installation précise afin de l'intégrer à la base de données sous la forme d'un modèle AREA.

```
public void     init()
{
  this.add(new Area(action name, reaction name, module action name,
    module reaction name, description));
}
```

L'installation doit se faire dans l'implémentation du init() dans la classe *AreaFixture*.

Le constructeur de la classe *Area* prend un string action name, un string réaction name, un string module action name, un string module reaction name ainsi qu'un string description.

5) Installation & Déploiement

Pour l'installation de l'AREA il est nécessaire d'avoir *Docker*.

ÉTAPE 1 > *sudo docker-compose up --build*

ÉTAPE 2 > *Lancer [AREA](http://localhost/8080)*