Documentation Dashboard

Implémenter un nouveau service ou un nouveau widget

Dans le package Services se trouve 2 enums ServicesEnum et WidgetsEnum.

ServicesEnum permet de définir un service avec ses widgets tandis que WidgetsEnum permet d'implémenter dynamiquement de nouveaux widgets.

On va créer un nouveau service en renseignant un nom et une liste de widget dans ServicesEnum.

Par exemple pour définir un nouveau Service appelé Youtube avec 2 widgets :

```
YoutubeService(new Service("Youtube")
        .addWidgets(
                WidgetsEnum.YoutubeChannelInformation.getWidget(),
                WidgetsEnum.YoutubeVideoInformation.getWidget()
        )),
```

Maintenant on veut définir ce que font chacun de ces 2 widgets YoutubeChannelInformation et YoutubeVideoInformation dans WidgetsEnum.

Un widget contient un nom, une description, une route, une action ou une requête (suivant s'il y a du traitement à faire ou si on veut seulement récupérer le contenu d'une requête), un temps de rafraichissement, une taille et des paramètres supplémentaires facultatifs.

Ci-dessous on peut voir le widget YoutubeChannelInformation qui est créer avec une action et une requête vide.

```
YoutubeChannelInformation(
        new Widget(
                "Youtube Channel Information",
                "Get all informations of a specific yt channel",
                "",
                new YoutubeChannelInformation(),
                (e) -> {
                    String api = e.get("@1");
                    HttpRequestInitializer init = a -> {};

                    YouTube bd = new YouTube.Builder(new NetHttpTransport(), new JacksonFactory(), init)
                            .setApplicationName("subcount").build();

                    YouTube.Channels.List searchengine;
                    Channel channel;
                    ChannelStatistics stats;

                    try {
                        searchengine = bd.channels().list("statistics, brandingSettings, snippet");
                        searchengine.setForUsername(e.get("@2"));
                        searchengine.setKey(api);
                        channel = searchengine.execute().getItems().stream().findFirst().orElse(null);
                        stats = channel.getStatistics();

                    } catch (Exception a) {
                        return null;
                    }

                    JSONObject obj = new JSONObject();

                    obj.put("subs", stats.getSubscriberCount());
                    obj.put("views", stats.getViewCount());
                    obj.put("videos", stats.getVideoCount());
                    obj.put("thumbnail", channel.getSnippet().getThumbnails().getDefault());
                    obj.put("country", channel.getSnippet().getCountry());

                    System.out.println(obj.toString());

                    return obj;
                },
                10,
                new Widget.Positions(400, 160),
                new Widget.Params("Username", "string")
        )
),
```

Ci-dessous on peut voir le widget CityWeather qui prend une requête en paramètre et une action null :

```
CityWeather(
        new Widget(
                "CityWeather",
                "Get the weather of your city",
                "http://api.openweathermap.org/data/2.5/weather?q=@1&appid=60b927b88df31d77863d7e5834970f6c",
                new Weather(),
                null,
                30,
                new Widget.Positions(450, 200),
                new Widget.Params("city", "string")
        )
),
```

Implémenter une nouvelle requête

Dans cette partie on va voir comment créer une nouvelle requête.

Il suffit de créer une classe dans le package Routers et d'hériter de la class RoutingProperties, il faut appeler le constructeur en donnant une nouvelle route et implémenter la méthode view qui prend en paramètre l'échange http et renvoie un objet json.

Une fois la class créée il faut ajouter la route au serveur en appelant la méthode addRoute du ServerHandler contenu dans le ServerMain.

Par exemple la class Connection :

```java
public class Connection extends RoutingProperties {

    public Connection() {
        super("/connection");
    }

    @Override
    public JSONObject view(HttpExchange t) {

        JSONObject jsonObject = new JSONObject(convertStreamToString(t.getRequestBody()));

        String email = jsonObject.getString("email");
        String password = digest(jsonObject.getString("password"));

        JSONObject json = new JSONObject();

        Object u = UsersManager.getManager().connectUser(email, password);

        if (u instanceof String) {
            System.out.println("Connection failed");
            json.put("res", "false");
            json.put("message", u);
            return json;
        }
        else {
            json.put("res", "true");
            json.put("datas", ((User)u).getServicesDatas());
            return json;
        }

    }
}
```