Création du DAO

Les DAO sont responsables de la communication avec le backend. Il sont également responsables de transformer la donnée reçue du backend en données compréhensibles par l'application : c'est à dire en données définies dans le package `src/app/models/` .

Création du fichier DAO

Les fichiers DAO sont à mettre dans le package `src/app/DAO/` . Ils sont nommés de la manière suivante : `<nom>DAO.ts` .
EXEMPLE pour le DAO qui va chercher des activité, on le nomme `ActivityDAO.ts` .

Injecter les service DAOService et AuthService

Pour effectuer les requêtes HTTP vers le backend, il convient d'utiliser le service **DAOService**. Ce service contient l'adresse du backend et propose deux méthodes : **get()** et **post()** pour effectuer respectivement les requêtes HTTP GET et HTTP POST.*

Le service AuthService peut ne pas être injecté, si le DAO doit faire des requêtes de nécessitant pas d'être authentifié. Mais sur Crew Mobile Access, appart l'action de login, toutes les actions nécessites d'être authentifié. Ce service permet de mettre à disposition les propriétés getter **sessionKey** pour le JWT et **connectedPN** pour les infos sur l'utilisateurs actuellement authentifié. Si aucun utilisateur authentifié, **connectedPN** vaut null et **AuthService.isAuth** est valorisé à FALSE.

```
import { Injectable } from "@angular/core";
import { DAOService } from "../services/DAO.service";

@Injectable()
export class ActivityDAO {
    private apiBaseURL: string;

    constructor(
        private daoService: DAOService,
        private authService: AuthService
    ) {
        this.apiBaseURL = daoService.restApiBaseURL; // <== ON RÉCUPÈRE L'ADRESSE DU BACKEND DÉFINI DANS LE FICHIER DE
    }
}
```

L'implémentation du DAO

Le DAO doit pouvoir être injecté, d'où l'utilisation du décorateur Angular @Injectable.
Les méthodes du DAO doivent renvoyer une promesse contenant, en cas de succès, la donnée recherchée (ou rien si c'est une action de type POST). Etant donnée que le backend est susceptible de renvoyer des status codes HTTP divers, le DAOService se content d'empaqueter les réponses qu'il produit dans l'objet DAOResponse du package `src/app/models/`. Cet objet contient les propriétés suivantes :

- un boolean **isSuccess** qui est à TRUE si le code HTTP STATUS et inférieur à 400.
- une chaine de caractère **content** qui contient le contenu de la réponse.
- un nombre entier **statusCode** qui contient le HTTP Status code renvoyé par le backend.

Cela permet de laisser au DAO spécialisé de gérer les cas spécifiques en fonction des HTTP status code. Voir l'exemple dans la méthode getActivitiesBetweenTimestamps() ci-dessous :

```
import { Injectable } from "@angular/core";
import { DAOService } from "../services/DAO.service";
import { AuthService } from "../services/Auth.service";
import { Activity } from "../models/Activity.model";
import { DAOResponse } from "../models/DAOResponse.model";
import { Logger } from "../core/Logger";

@Injectable()
export class ActivityDAO {
    private apiBaseURL: string;
    private static URL_GET_ACTIVITIES_BETWEEN_TIMESTAMPS = "/getActivities";
    private static URL_GET_ACTIVITY_BY_ID = "/getActivity";
    private static URL_POST_EXPORT_PLANNING_PDF = "/pdf/generate-planning";
    private logger: Logger;

    constructor(
        private daoService: DAOService,
        private authService: AuthService
    ) {
        this.apiBaseURL = daoService.restApiBaseURL;
        this.logger = Logger.getInstance("ActivityDAO");
    }
```

```
/**
 * Retrieve activities between two timestamps and returns a list of them by calling the success callback of the returned Promise
 *
 * @param startDateTime
 * @param endDateTime
 */
getActivitiesBetweenTimestamps(startDateTime: number, endDateTime: number): Promise<Activity[]> {
    let _this = this;
    return new Promise((success, reject) => {
        _this.daoService.get(_this.apiBaseURL + ActivityDAO.URL_GET_ACTIVITIES_BETWEEN_TIMESTAMPS, {
            startDateTime,
            endDateTime,
            userID: _this.authService.connectedPN && _this.authService.connectedPN.userID || 0
        }).then((response: DAOResponse) => {
            if (response.isSuccess) {
                let activitiesJSON = JSON.parse(response.content);
                success(_this.createActivitiesListFromJSONResponse(activitiesJSON));
            } else {
                if (response.httpStatusCode === 404) { // not concidered has error but 404 happen when no activity were found
                    success(new Array());
                } else {
                    this.logger.error("Error when fetching activities from server. Server responsed with HTTP Status code " + re
                    reject()
                }
            }
        }).catch(() => {
            reject();
        });
    });
}

/* LES DEUX MÉTHODES CI-DESSOUS SONT UTILISÉES POUR TRANSFORMER LES DONNÉES REÇUES DU BACKEND EN DONNÉES UTILISÉES PAR LES SERVI
private createActivitiesListFromJSONResponse(activitiesJSON: any): Activity[] {
    let activities: Activity[] = new Array();
    for (let activityJSON of activitiesJSON.activity) {
        activities.push(this.mapActivityFromJson(activityJSON));
    }
    return activities;
}
```

```
private mapActivityFromJson(activityJSON: any) {
    return new Activity(
        activityJSON.acIataCode             || null,
        activityJSON.acRegistration         || null,
        activityJSON.actDescription         || null,
        activityJSON.activityID             || null,
        activityJSON.airlineCode            || null,
        activityJSON.arrAirpName            || null,
        activityJSON.arrAirport             || null,
        activityJSON.arrDate                || null,
        activityJSON.arrDateLtAirp          || null,
        activityJSON.arrDateLtBase          || null,
        activityJSON.color                  || null,
        activityJSON.crewComment            || null,
        activityJSON.crewFunctionName       || null,
        activityJSON.depAirpName            || null,
        activityJSON.depAirport             || null,
        activityJSON.depDate                || null,
        activityJSON.depDateLtAirp          || null,
        activityJSON.depDateLtBase          || null,
        activityJSON.dutyNatureCode         || null,
        activityJSON.dutyTypeCode           || null,
        activityJSON.dutyTypeCwa            || null,
        activityJSON.flightNumber           || null,
        activityJSON.hotelAddress           || null,
        activityJSON.hotelName              || null,
        activityJSON.hotelPhoneNumber       || null,
        activityJSON.isOperatingFlight      || null,
        activityJSON.isPositioningFlight    || null,
        activityJSON.offFltDutyFunct        || null,
        activityJSON.offFltDutyFunctName    || null,
        activityJSON.operatSuffix           || null,
        activityJSON.secondFunctionName     || null
    );
  }
}
```