L'export PDF (cotés application MOBILE/WEB)

1 - Créer un DAO pour invoquer le end point sur le backend (Keop bridge)

Voir comment créer un DAO

2 - Créer le service faisant appel au DAO

Voir comment créer un service

3 - Si la génération sur le backend a réussi, le service peut utiliser le DAO spécialisé dans le téléchargement des fichiers `FileDAO`

Exemple :

```
import { ActivityDAO } from "../DAO/ActivityDAO";
import { FileDAO } from "../DAO/FileDAO";
...
_this.activityDAO.exportPlanningToPDF(fromDateTimestamp, toDateTimestamp, planningTimestamp)
    .then((urlPDFFile: string) => {
        _this.logger.info("Planning PDF has been successfuly generated. The file is now being downloaded...");
        _this.fileDAO.getFileFromURL(urlPDFFile, fileDownloaded => {
        if (fileDownloaded) {
            success(fileDownloaded);
        } else {
            _this.logger.error("File could not be downloaded");
            reject();
        }
    });
}).catch(() => {
    _this.logger.error("Something happened when exporting planning to PDF");
    reject();
});
```

Pour créer un nouveau template (KEOPS Bridge)

1 - Créer le template HTML

Les templates doivent être mis dans le dossier `src/main/resources/templates`. Les template sont des fichier html contenant les **Expression Language** du moteur template **Thymeleaf**.

Doc thymeleaf 3.0

2 - Ajouter des ressources

Les ressources (CSS, Images, font) doivent être mis dans le dossier `src/main/resources/assets`
NOTE : Attention, il n'y a pas de préprocessing donc pas de compatibilité avec le SASS.

3 - Référencer le nouveau template dans l'enumération

Afin de rendre le listing des template plus lisible, il convient de référencer le nouveau fichier template dans l'enum `KeopsBridge/src/main/java/app/keops/bridge/common/PDFService/PDFTemplates`

Pour cela il suffit d'ajouter un attribut et de passer en paramètre le nom du fichier template.

Exemple : J'ai ajouté le fichier template `exemplePDFTemplate.html`, je doit ajouter la ligne suivante dans l'enum : `Exemple("exemplePDFTemplate.html");`

4 - Créer le contexte

Le contexte permet de définir les données qui doivent être mappée dans le template. Si dans un cas particulier on ne souhaite pas ajouter de donnée au template, on peut utiliser un contexte prédéfini nommé `EmptyContextBuilder`.

Pour créer un nouveau contextBuilder, il faut créer une classe héritant de `AbstractPDFContextBuilder` et de la placer dans le package `KeopsBridge/src/main/java/app/keops/bridge/common/PDFService`.
La classe doit avoir l'annotation `@Component` de **Spring** afin de pouvoir être injecté.

La seule obligation de cette classe est d'implémenter (rédéfinir) la méthode `prepareContent()` qui doit renvoyer un objet (classe interne de `AbstractPDFContextBuilder`) qui contient les données à être mappées.

IMPORTANT il est possible au besoins de créer des DTO permettant d'adapter la structure de donnée si celle du **domain** n'est pas suffisante. Dans l'exemple ci-desssous le DTO `planningDTO` est utilisé.
Les DTO utilisés par le service de génération PDF doivent être mis dans le package `KeopsBridge/src/main/java/app/keops/bridge/common/PDFService/DTO`.

EXEMPLE :

```
@Component
public class PlanningPDFContextBuilder extends AbstractPDFContextBuilder {
        private final Logger logger = LoggerFactory.getLogger(this.getClass());

        private static final String KEY_PLANNING = "planning";
        private static final String KEY_FONTS_DECLARATION = "fontsDeclaration";

private PlanningDTO planningDTO;
private ResourceLoaderService resourceLoader;

public PlanningPDFContextBuilder(PlanningDTO planningDTO) {
        this.planningDTO = planningDTO;
        this.resourceLoader = new ResourceLoaderService();
}
```

```
@Override
protected BindingContent prepareContent() {
        HashMap<String, Object> objectsToMap = new HashMap<String, Object>();

        objectsToMap.put(KEY_PLANNING, this.planningDTO);
        String fontsDeclarationString = "";

        try {
                String dinproFontFilePath = resourceLoader
                                .getResourceAbsolutePathString("/assets/font/DINPro_Light_tr.woff");
                String montserratFontFilePath = resourceLoader
                                .getResourceAbsolutePathString("/assets/font/Montserrat_Light.woff");
                fontsDeclarationString = "@font-face {\r\n" + //
                                "           font-family: \"Montserrat\";\r\n" + //
                                "           src: url(\"" + montserratFontFilePath + "\") format(\"woff\");\r\n" + //
                                "       }\r\n" + //
                                "       @font-face {\r\n" + //
                                "           font-family: \"DINpro\";\r\n" + //
                                "           src: url(\"" + dinproFontFilePath + "\") format(\"woff\");\r\n" + //
                                "       }"; //
                } catch (Exception e) {
                        logger.warn("Error while building the PDF template context. A font file could not be loaded : "
                                        + e.getMessage());
                }

                objectsToMap.put(KEY_FONTS_DECLARATION, fontsDeclarationString);

                return new BindingContent(objectsToMap, //
                                "templates/fragments/activityTemplateFragment.html", //
                                "assets/css/PDFStyle.css", //
                                "assets/img/head.png", //
                                "assets/img/duration_icon.png", //
                                "assets/img/servings_icon.png");
        }
}
```

5 - Ajouter le context builder à la beanFactory

Afin de pouvoir concerver les bénéfices du framework d'injection de dépendance de **Spring**, il convient de passer par la beanFactory.

Pour cela il suffit d'ajouter une méthode permettant de créer l'instance du context builder nouvellement créé dans la classe `BeansFactory` du package `PDFService`.

EXEMPLE :

```
 @Bean
@Scope(value = "prototype")
public PlanningPDFContextBuilder makePlanningPDFContextBuilder(PlanningDTO planningDTO) {
    return new PlanningPDFContextBuilder(planningDTO);
}
```

6 - Créer le end point

Le end point doit être ajouté dans un contrôleur REST. Le end point doit retourner la chaine de caractère renvoyée par le service **PDFGenerator** (voir l'exemple ci-dessous).

Le service réclame 3 paramètres :

- Une valeur de l'enum PDFTemplate
- Une chaine de caractère pour le nom du fichier PDF à générer
- Le PDFContextBuilder associé (optionnel). Ne pas renseigner ce paramètre pour un contexte vide.
    - ATTENTION, il faut passer par la BeanFactory du service PDF pour instancier le PDFContextBuilder.

```
@POST
	@Path("/pdf/generate-planning/{crewCode}")
	@Produces(MediaType.TEXT_PLAIN)
	@SuppressWarnings("deprecation")
public String resetPassword(@PathParam("crewCode") String crewCode, @RequestBody ExportPDFTimestamps exportPDFTimestamps) th
	String generatedPDFFileURL = "";

	GetActivitiesResponse activitiesResponse = this.getActivities(exportPDFTimestamps.getFromTimestamp(), exportPDFTimes
	if (activitiesResponse == null) {
		return null;
	}
	PlanningDTO planningDTO = this.mapperDTO.makePlanningDTOFromActivityResponse(activitiesResponse, exportPDFTimestamps

	Date date = new Date(exportPDFTimestamps.getPlanningTimestamp());
	MONTHS_NAME_SHORTEN monthNames[] = MONTHS_NAME_SHORTEN.values();
	String fileOutputName = "planning-" + monthNames[date.getMonth()] + "-" + (date.getYear() + 1900) + ".pdf";

	generatedPDFFileURL = pdfService.generatePDFFromHTML(crewCode, PDFTemplates.planning, fileOutputName,
			this.beansFactory.makePlanningPDFContextBuilder(planningDTO));

	return generatedPDFFileURL;
}
```