Rapport de projet (groupe 2)

# Tâches effectuées :

Authentification :

* Authentification par le biais d’une adresse mail unique et mot de passe hashé en bdd page de connexion qui renvoie au dashboard.
* Déconnexion qui supprime la session utilisateur
* Mise en place de trois rôles ( ADMIN, USER, CONTRIBUTOR)
* Gestion des accès aux fonctionnalités du site par le biais des rôles

Travaux effectués avec la fonctionnalité native de Symfony. Il y a possibilité de le modifier, et l’adapter facilement et rapidement en cas de besoin. *Ne pas oublier de continuer la gestion des accès au fur et à mesure de l’avancement du projet !!*

Log :

Mise en place d’un historique de toutes les actions (CRUD) effectuées sur la base de données.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Action** | **Date** | **Identifiant de l'objet** | **Type de l'objet** | **Version** | **Donnée(s)** | **Utilisateur** |

Un administrateur a la possibilité d’annuler une action effectuée par un utilisateur lambda comme la modification, ou la suppression d’une donnée qui serait irréversible sans ce système.

Lors de la suppression :

* La donnée est conservée avec une date de suppression qui lui est affectée.
* Elle n’apparaît plus sur le site car un filtre est appliqué lors de la lecture de la base de données

Un listener vient supprimer définitivement les données au bout d’un temps X définit par l’administrateur.

Un second listener vient supprimer la table des log au bout d’un temps X.

Mise en place d’un menu de configuration, ou l’utilisateur mets à jour des paramètres comme la durée de vie des logs.

Travaux effectués avec l’extension de Doctrine (GEDMO, Loggable, Softdelete).

Import :

Mise en place de l’importation de données utilisateur dans la bdd de MariaDB depuis un fichier CSV et respectant un unique modèle de champs ( Nom, Prenom, Ville, CodePostal, NewsLetter , Date de naissance ,Institution).

Le système détecte par le bais de regex la correspondance des champs entre le fichier et la bdd.

Le système indique les doublons, la présence des institutions non créées ou les colonnes qui ne peuvent être nulle. Si une erreur survient, aucune donnée n’est insérée , et l’utilisateur est prié de corriger les erreurs et de refaire la manipulation.

Travaux effectués avec Spout. Il reste la prise en charge des formats XLSX et ODT . De plus il faut rajouter la gestion et la prise en charge de différents modèles.

Export :

L’exportation des données des utilisateurs dans la base de données s’effectue à l’aide d’un bouton d’export situé au niveau du listing des participants. Ce bouton permet de récupérer l’ensemble du listing. Pour l’instant, la prise en compte des filtres pour récupérer uniquement les données sélectionnés n’a pas été mis en place.

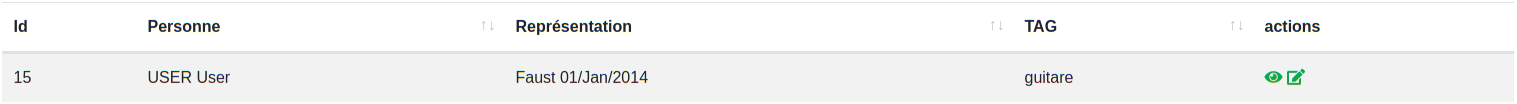
La technologie Spout a été choisi pour sa performance et par le fait qu’elle permet de gérer et d’exporter des fichiers en xlsx, csv et ods. Dans le cas du projet, l’exportation se fait uniquement en format xlsx.

L’utilisateur peut donc exporter en format xlsx la liste entière des participants présent dans la base de données.

Affectation de tags :

L’affectation de tags sur l’interface web se fait sur la page de récapitulatif d’une personne (/manager/person/{id} ⇒ manager/entity\_people\_show). Les tags décrivent le rôle de la personne dans une représentation (performances). Une représentation est définie par un nom de spectacle et une date.

Ex :



La personne “USER User est guitariste lors de la représentation du spectacle “FAUST” le 2 janvier 2014.

Cette affectation de tags est faite dynamiquement, elle a besoin d’un tag et d’une représentation. Si l’un des deux champs ou les deux ne sont pas crées, un bouton permet de les ajouter sans avoir à recharger la page.

Améliorations :

* Grâce à une API, la récupération des données pourrait se faire beaucoup plus facilement. Actuellement, la liste des tags dans le champ “select” est récupérée via une route (/manager/tags/list) sous la forme d’une liste d’option.
* Le contenu des modals est généré grâce à une route spécifique (/manager/tags/modal/new pour ajouter des tags) afin de ne pas avoir tout le contenu de la page (menus…) dans ceux-cis mais d’avoir quand même le formulaire d’ajout de Symfony). Il faudrait donc plutôt passer par une API pour ajouter les tags pour avoir quelque chose de plus propre.

Filtres :

La gestion des filtres a été réalisée avec DataTables (<https://datatables.net/>), une librairie JS très complète pour trier et filtrer des tableaux. Un thème Bootstrap est fourni avec (CSS + JS). Tout est défini dans le fichier package.json” dans “app”.

Pour chaque tableau présent sur le site, une fonction de recherche, un dropdown pour sélectionner le nombre d'éléments à afficher ainsi qu’une gestion de la pagination, sont implémentés.

Pour afficher les filtres comme sur la page “Personnes” ou “Sauvegardes / Restaurations” en bas du tableau (voir ci-dessous), il suffit de rajouter un élément “tfoot” à celui-ci et le JS (script.js) devrait le prendre en compte.



Améliorations :

* Grâce à une API, filtrer plus précisément ce que l’utilisateur souhaite (prendre exemple sur la gestion des filtres d’OpenProject sur la page des “lots de travaux”).

CRUD sur les données avec des appels Ajax sur l’API, cela évitera de charger toutes les données et les trier mais plutôt de les récupérer triées et les afficher.

# Tâches restantes :

Bundle 1 :

* Création de l’API REST avec API Platform ou FOS REST Bundle
* Gérer les champs dynamiques des formulaires en fonction de modèles sous MongoDB

Bundle 2 :

* Ajouter une procédure d’ajout de compte utilisateur sur la page de connexion…

Bundle 3 :

* Gérer les différents types de documents (CSV et ODS)
* Exporter / Importer les données en fonction des modèles définis plus haut

Bundle 4 :

* Chiffrement des données avec Halite
* Chiffrement du site après déploiement et passage en stade de production

Déployer l’application sur le site lfovdl.com (voir avec M. Rozsavolgyi pour les identifiants ou avec M. Pannetier).