# Solutions techniques :

## Gérer les participants :

Dans sa globalité, le système sera gérée comme une bibliothèque d’entités personnalisables.

Le fonctionnement sera principalement orienté “objet” et tournera autour de l’entité “People” représentant les personnes en interaction avec l’association et nécessitant d’être entrées dans l’annuaire.

Les deux autres entités principales sont Institutions et TAGs :

* Les institutions sont des entités représentant les origines des participants.
* Les TAGs sont des mots clés permettant d’identifier facilement un groupe de personnes.

Système de fiches identifiées par nom/prénom (+ date de naissance pour les doublons)

-> Modèle document + modèle relationnel ?

-> Modèle document simple ? -> problème pour l’identification / problème de sécurité des données

Volume de données assez important -> nécessité d’utiliser un SGBD

-> potentiellement 2 différents au vu de la pluralité des types des gestion (relationnel + document)

Affichage des éléments et de leur formulaire de modification grâce à un itérateur de parcours de l’objet.

## Représentation des données à l’utilisateur

Export dans un fichier Excel / Page d’affichage

ça dépend

svp lire openproject

## Permettre la saisie des données par les personnes autorisées

L’accès à la saisie nécessitera une authentification préalable.

Le formulaire d’ajout de participants devra contenir des champs non obligatoires, et des champs modulables pour permettre l’ajout des données nécessaires aux représentants.

La base de données devra contenir un référencement des droits ou/et des rôles qui sera attribué aux utilisateurs :

On gérera les droits et/ou les rôles sur une page de saisie de données administrateur

La possibilité de modifier des informations sera régie par les rôles et/ou les droits

Des rôles seront déjà prédéfinis : Accès lecteur (Possibilité de tri et d’export des données), Administrateur (Accès total), Contributeur (Modification des fiches attribués et accès aux modèles d’export qui lui sont assignés)

Les permission seront en premier lieu une gestion de la lecture et de l’écriture de la BD.

Un participant sera automatiquement rattaché à un ou plusieurs responsables par le biais de son institution.

Il faudra définir des “templates” de formulaires pour correspondre à tous les participants.

Les contributeurs pourront modifier les “templates” qui leur auront été assigné pour correspondre aux informations à renseigner.

L’accès aux données sera géré par les rôles défini par l’administrateur.

Les actions réalisées sur les participants (ajout, modification, etc ...) seront enregistrées dans un registre pour avoir un suivi.

formatage des entrées du registre : “Date;Heure;Element;Type\_action;Commentaire;ID\_utilisateur”

## Permettre la personnalisation des champs au cas par cas

**Flexibilité des champs à renseigner**

Le SGBD doit mettre en relation une entité avec sa fiche d’informations.

Le modèle document se prête bien à la fiche puisqu’il permet une liberté presque totale dans le choix des champs de données de chaque entité.

Un utilisateur admin pourra définir un modèle par défaut afin de le proposer comme document d’origine et définir des champs obligatoires pour le formulaire d’ajout d’entité.

Un utilisateur externe pourra utiliser un éditeur de modèle pour créer son propre modèle et ainsi personnaliser ses besoins en terme de renseignement d’informations.

Un utilisateur admin pourra modifier n’importe quel modèle ou entité dans la base de données.

Un utilisateur externe ne pourra modifier que les entités auquel il est rattaché, sur lequel il a les droits (exemple : un professeur d’école ne pourra que modifier ou ajouter des élèves dont il est le responsable)

Le système de modèles de formulaire pourra être un élément central dans la distribution des formulaires de renseignement vers les utilisateurs externes désignés comme responsables d’inscription.

## Permettre l’import/export depuis et vers l’ancien système

L’utilisateur administrateur doit pouvoir choisir entre l’ancien système et le nouveau. L’ancien système correspondrait à une manière alternative de sortir les données de la base.  
Nécessité d’adapter un processus de conversion de formatage des données entre le nouveau et l’ancien format.

Un bouton d’export sera présent sur sur la fiche personnel et sur le listing pour récupérer les données ajoutées.

Conversion entre l’ancien et le nouveau système d’import/export qui permet de garder le même format de fichier que celui déjà en place pour ne pas déstabiliser les utilisateurs.

L’utilisateur pourra exporter, soit la liste entière des participants (avec ou sans filtres selon son rôle et son choix), ou alors une fiche particulière si besoin d’informations sur un seul participant. L’exportation pourra se faire en fonctions des tags.

Un fichier sera mis à disposition d’un fichier modèle pour l’importation afin de donner un exemple des informations à renseigner sous les deux formats (ancien et nouveau).

L’importation/exportation dépendra des droits prédéfinis au préalable aux utilisateurs

## 

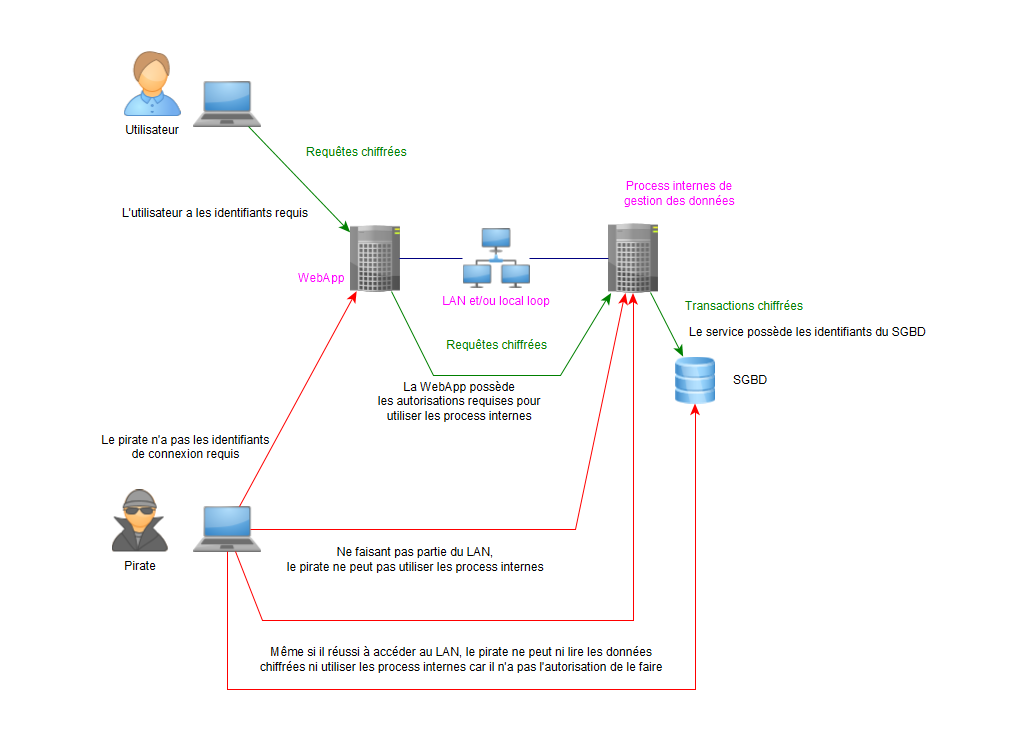
## Être sécurisé

Chiffrer les données. Cela pourrait passer par un système intégré dans le SGBD ou par une API externe ou le développement d’une API propre.

Authentification des utilisateurs à simple facteur.

Vérification de l’authenticité de l’acteur (personne ou logiciel) lors de la modification des données (token ?).

Process de gestion des données uniquement utilisés en local.



Système de sauvegarde des données. Au cas où les données soient corrompues malgrés les précautions précédentes. (Donnera lieu au PRA/PCA).

## Doit être agréable à l’utilisation, ergonomique et intuitif

Éviter les changements de page inutiles. Cela risque de perdre l’utilisateur dans son cheminement.

-> Utilisation de modaux, de fenêtre, champs dynamiques, validateurs …

-> Utilisation de technologies de mises à jour dynamiques.

# Solutions technologiques :

## Sécurisation de l'infrastructure :

Afin de sécuriser l’infrastructure, nous avons décidé de choisir entre 2 bibliothèques, nous n’allions pas utiliser *crypt()* pour chiffrer des données personnelles car cette fonction hash juste le texte, nous avons besoin de le déchiffrer derrière et non de le comparer. Pour choisir entre les deux bibliothèques, nous les avons comparer. Nous avons fais notre choix entre Halite et Libsodium. Cette dernière étant la plus sécurisé des librairies de cryptographie actuellement et Halite étant une extension de Libsodium, elle embarque plus de choses.

Par exemple Halite permet de chiffrer des fichiers entiers, des cookies et d’autres encore.

De plus, Halite est plus intuitive, possède une interface plus simple et est plus rapide que Libsodium. Face à tous ces avantages, notre choix c’est porté sur Halite qui semble plus facile d’utilisation sans pour autant être moins sécurisé que Libsodium.

Les données que nous aurons à chiffrer sont des plus sensibles car ce sont des données personnelles, elles seront chiffrer en entrée et déchiffrer en sortie de la base de données. Les utilisateurs qui voudront s’authentifier passeront par le *SecurityBundle* de Symfony, ils auront ensuite accès aux données personnelles déchiffrées dès l’authentification réussie.

Pour identifier ces données, nous allons devoir les marquer en créant deux table comme-ci dessous. On y trouvera tous les champs de la base de données organisés selon un système permettant d’identifier leur table et leur politique de chiffrement

## Front-End :

Gestion des paquets :

Pour la gestion des paquets, on utilisera yarn pour sa capacité de verrouillage des versions dans le but de faciliter la gestion des paquets entre toutes les machines de développement.

Gestion des assets :

Pour la gestion des assets (css/js), on utilisera gulp via node.js de manière à rendre la gestion plus efficace en automatisant via des tâches de routine.

On créera un dossier “assets” à la racine du projet de manière à centraliser les ressources du gulpfile.js. Ainsi, la gestion est plus efficace et permet une meilleure gestion dans le versionnage Git.

Les résultats des compilations iront trouver leur place dans les dossiers de chaque app/bundle.

## Import/Export:

Afin de faciliter l’import et l’export de données, nous allons définir des modèles de tableau pour l’import et l’export. Par exemple, un modèle de fiche personnelle, de liste d’un orchestre, de liste d’élève… L’utilisateur pourra ajouter et modifier des modèles, afin de donner plus de détails ou de grouper les participants sous une autre catégorie.

En cherchant des solutions technologiques, nous avons pus trouver deux librairies qui peuvent effectuer l’export : mk-j PHP\_XLSXWriter et Spout. En les comparant nous pouvons voir que en terme de performance, ces deux librairies sont au même niveau. Par contre Spout est avantagé par le fait qu’elle puisse exporter en fichier csv, et ods. Pour effectuer l’export nous utiliserons [Spout](https://github.com/box/spout).

Comme notre projet est un projet qui date de l’année dernière, il y a déjà des fonctionnalités implémentées, notamment pour l’import depuis un fichier csv vers notre base de donnée. Il nous faudra juste détaillé cette fonctionnalité.

Quand l’utilisateur choisira d’exporter une liste ou une fiche personnelle, le fichier créé portera l’extension .xlsx

Conversion excel to JSON : <https://www.npmjs.com/package/convert-excel-to-json>