

Remerciements

Je remercie d'abord Monsieur Delvigne pour m'avoir guidé ce quadrimestre durant le développement de ce travail.

Je remercie ensuite l'EPHEC pour m'avoir donné les connaissances techniques dont j'avais besoin pour débiter ma carrière.

Je remercie enfin James Anderson, mon patron, pour la chance qu'il m'a donnée de pouvoir me développer dans sa société.

Table des matières

CHAPITRE 1	4
INTRODUCTION	4
1.1 PRESENTATION DU TRAVAIL	4
1.2 OBJECTIFS	4
1.3 MARCHE SUIVIE	5
CHAPITRE 2	6
ANALYSE DE PRE-DEVELOPPEMENT	6
2.1 FONCTIONNALITES	6
2.1.1 CONNEXION ET ACCUEIL	6
2.1.2 PAGE PERSONNELLE	7
2.1.3 CREER UN NOUVEL AUDIT	7
2.1.4 FAIRE UN AUDIT INTERNE	8
2.1.5 CREER UN NOUVEL AUDIT EXTERNE	10
2.1.6 REVUE DES AUDITS INTERNES CREEES	11
2.1.7 ADMINISTRATION DE L'APPLICATION	12
2.2 GESTION DES CLIENTS	14
2.3 SEGMENTATION DES ACCES	15
2.3.1 DIVISION DES ROLES	15
2.3.2 GESTION DES PERMISSIONS DES AUDITS	16
CHAPITRE 3	17
MISE EN PRATIQUE	17
3.1 MARCHE SUIVIE	17
3.1.1 COTE SERVEUR	17
3.1.2 COTE CLIENT	17
3.2 SYMFONY 4	18
3.2.1 OUTILS EMPLOYES	18
3.2.2 STRUCTURE D'UN PROJET SYMFONY	20
3.3 INTEGRATION DE LA BASE DE DONNEES DANS LES FONCTIONNALITES	20
3.3.1 SCHEMA ENTITE-ASSOCIATION	21
3.3.2 SCHEMA RELATIONNEL	22
.....	22
3.3.3 GESTION DES EVENEMENTS	23
3.3.4 SUPPRESSION LOGIQUE	23
3.3.5 GESTION DES OPERATIONS SUR LES AUDITS	23
3.3.6 GESTION DES RESULTATS D'UN AUDIT	24
3.4 IMPLEMENTATION DE LA SECURITE	24
3.4.1 INJECTIONS SQL ET XSS	25
3.4.2 SPAM DE FORMULAIRE DE CONNEXION ET FORCE BRUTE	25
3.4.3 AUTHENTIFICATION ET GESTION DES ACCES GRACE AU PARE-FEU SYMFONY	25
CHAPITRE 4	29
CONCLUSION	29
4.1 RESUME	29
4.2 ÉTAT D'AVANCEMENT ET REMARQUES PERSONNELLES	30
4.3 DEMARCHE CRITIQUE	30
CHAPITRE 5	31

WEBOGRAPHIE	31
• REFSNES DATA, <i>W3SCHOOLS</i> . HTTPS://WWW.W3SCHOOLS.COM/	31
CHAPITRE 6	32
ANNEXES	32
ANNEXE A : EXEMPLE PROFILER, REQUETES GET ET POST	32
ANNEXE B : FONCTIONNEMENT DE DOCTRINE	32
ANNEXE C : PRINCIPES ET DROITS RGPD APPLIQUES*	33

Chapitre 1

Introduction

1.1 Présentation du travail

Le noyau de mon travail est le développement de l'application web **Securicheck**. L'idée m'est venue par mon travail **dans la société de consultance informatique Anderson**. Celle-ci s'occupe, entre autres, de l'infogérance de parcs informatiques de petites entreprises. Cette activité correspond majoritairement à assurer quotidiennement le bon fonctionnement des systèmes informatiques de nos clients.

L'application peut jouer un rôle déterminant pour la société. Premièrement, elle **améliore notre service auprès de nos clients** en offrant instantanément une vue sur leur niveau de sécurité. De plus, grâce au résultat d'un audit, nous pouvons voir rapidement les faiblesses dans une infrastructure informatique afin de prendre des mesures efficacement. Deuxièmement, elle a **un pouvoir marketing** qui a le potentiel d'élargir notre clientèle existante. En effet, l'application pourrait également avoir comme vocation de servir comme **point d'accroche** chez des clients potentiels. Un audit pourrait être un service gratuit que l'on fournirait à toute entreprise externe en ayant fait la demande.

D'un côté, nous aurions les audits internes destinés à notre propre clientèle. Ils seraient réalisés par nous avec une approche technique. D'un autre côté, les audits externes réalisés par l'entreprises même. De ce fait, l'analyse n'est pas le même en fonction du type d'audit.

Securicheck se doit d'être **un outil complet de gestion d'audits informatiques**. Le tout de manière professionnelle et sécurisée avec des fonctionnalités innovantes ainsi que dans le respect des données des entreprises auditées.

À noter que dans le cadre de ce travail, je n'inclue pas la rédaction du contenu des audits mais l'implémentation des outils nécessaires à leur bonne gestion. Ainsi, il ne restera plus qu'à les écrire en interne et à les injecter dans la base de données.

1.2 Objectifs

Le premier objectif est de **répondre au besoin de mon client** qui est de disposer d'un système personnel d'audit pour PME. Le second objectif est de prouver mon aptitude à **analyser une problématique de façon professionnelle et d'y répondre avec des solutions adaptées**. Le dernier objectif est un développement personnel par l'apprentissage de nouveaux outils et méthodes de travail.

1.3 Marche suivie

La première partie du travail a été d'en faire **son analyse** afin de comprendre quels sont les tenants et aboutissants de l'application. D'abord, j'ai écrit les fonctionnalités adaptées aux besoins de mon client. En suivant les bonnes pratiques de gestion de projet, je les ai décrites par un schéma des **cas d'utilisation** afin de les rendre plus visuelles. Celui-ci est par nature très schématique et ne rentre pas dans les détails.

Elles sont également accompagnées des **récits d'utilisateur** afin de décrire au mieux le fonctionnement de l'application. Puis, j'ai fait l'étude de l'implication du **Règlement Général sur la Protection des Données** dans la gestion de nos clients. J'ai conclu l'analyse par construire le découpage des accès.

J'ai entamé la partie pratique par mettre en commun ces différentes parties que j'ai traduites en **schémas de base données**. De cette manière, j'ai pu la construire pour qu'elle soit durable, adaptée à la problématique et flexible.

Je me suis plongé sérieusement dans le développement de l'application seulement après avoir terminé chaque étape ci-dessus. Je suis d'abord passé par une phase d'apprentissage des différents outils qui m'ont servis durant le développement.

Chapitre 2

Analyse de pré-développement

Le client pour lequel je développe l'application est une petite société dans laquelle je travaille. Celle-ci pratique la consultance informatique de différentes manières. Premièrement, elle offre aux petites entreprises **un service à la demande** afin de les dépanner quand cela est nécessaire. Deuxièmement, elle propose aux petites et moyennes entreprises **une infogérance complète** de leur parc informatique ainsi qu'un suivi continu. Troisièmement, elle **déploie des projets de réseau et d'administration système** avec des solutions adaptées.

La partie suivante décrit l'analyse du travail. Elle permet de répondre à plusieurs questions importantes comme :

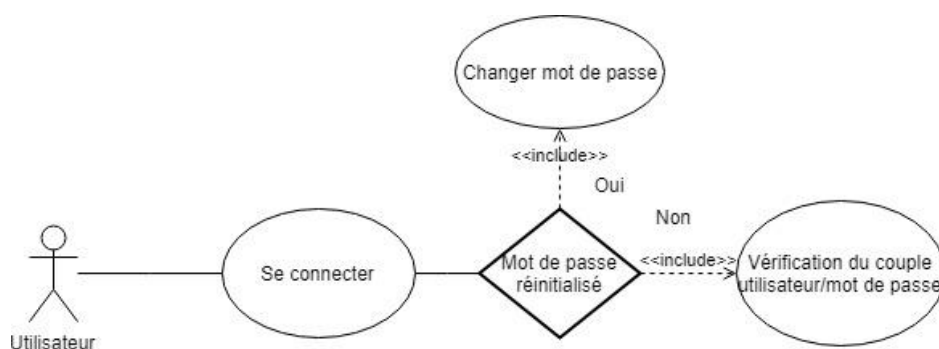
- Qui peut accéder à quelles fonctionnalités ?
- Quelles données à caractère personnel doivent être demandées aux entreprises ?
- Quelles données doivent être archivées dans la base de données et quelles données doivent être définitivement supprimées dans le cadre de la législation ?
- Comment les accès aux audits créés contenant les résultats sont-ils maintenus ?

2.1 Fonctionnalités

2.1.1 Connexion et accueil

La première chose à faire pour accéder aux fonctionnalités de l'application va être de **s'authentifier** via un portail de connexion en ligne par un couple adresse mail/mot de passe. Si l'utilisateur a perdu son mot de passe, il peut faire une demande de réinitialisation. Celle-ci sera prise en charge par un administrateur compétent.

Après une réinitialisation, il sera demandé à l'utilisateur de modifier son mot de passe à la première connexion. Une fois authentifié, celui-ci est redirigé vers un **tableau de bord** qui contient diverses données intéressantes de la base de données comme des statistiques. La plus importante est, pour les administrateurs globaux, le **journal des événements**.



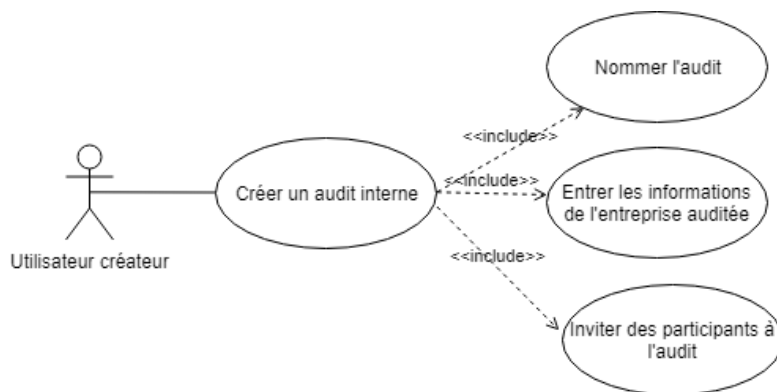
En tant que	Je voudrais	Afin de	Critères d'acceptation
Utilisateur	Me connecter à l'application	M'authentifier et bénéficier des rôles et des permissions propres à mon compte	<ul style="list-style-type: none"> - Valider le Captcha - Rentrer un couple login/mot de passe correct - Dans le cas d'une réinitialisation de mot de passe, rentrer le mot de passe initial et confirmer doublement le nouveau mot de passe avec une exigence de 8 caractères alphanumériques minimum

2.1.2 Page personnelle

Une fois cette première étape complétée, l'utilisateur peut accéder à ses informations personnelles et les modifier. Celles-ci comprennent le **prénom**, **nom**, **fonction** dans la société et **une photo de profile** optionnelle.

2.1.3 Créer un nouvel audit

Avant toute chose, créer un audit requiert de lui donner **un nom**, d'entrer **les informations à caractère personnel** de l'entreprise auditée. Le créateur peut également inviter des participants si l'audit se fait à plusieurs. Ceux-ci prendront également le **rôle de créateur** sur cet audit.



En tant que	Je voudrais	Afin de	Critères d'acceptation
Utilisateur Créateur	Un bouton	Créer un nouvel audit	- Un audit pour cette entreprise n'a pas déjà été créé auparavant
Utilisateur Créateur	Nommer l'audit	L'identifier facilement dans le futur s'il existe plusieurs versions de cet audit	
Utilisateur Créateur	Des champs	Encoder le nom de l'entreprise et la personne responsable	

Utilisateur	Inviter des collègues à	Partager le rôle de créateur
Créateur	participer à l'audit	sur cet audit

2.1.4 Faire un audit interne

Chaque audit doit être **dynamique** en fonction de l'infrastructure informatique auditée. Pour les PME, on peut facilement partir **d'un jeu de questions de base** auquel viendront s'ajouter des tests suivant certains critères. Un audit commence donc par un petit questionnaire pré-audit qui identifie l'utilisation de l'informatique du client.

Chacune d'entre elles se caractérise par un **type** suivant la nature de sa réponse :

- Une **'checkbox'**
- Une **sélection à choix multiple**
- Une **zone de texte pour les réponses informatives**

L'idée est de pouvoir générer un template de questionnaire pré-audit rapidement en modifiant simplement quelques champs dans la base de données.

Elle peut être liée à un ou plusieurs tests de l'audit. Par exemple, toutes les entreprises n'ont pas le besoin d'un réseau Wi-Fi. Par conséquent, il ne conviendrait pas d'auditer la sécurité d'un réseau qui n'existe pas. On pourra ainsi poser la question « Disposez-vous d'un réseau Wi-Fi ? ». Si la réponse est positive, alors les tests qui s'y rapportent doivent s'ajouter à l'audit avant de le débiter.

Cependant, une question pourrait également **retirer** un test ajouté précédemment. Le lien avec le celui-ci doit être complété par une action : **l'ajouter** ou **le retirer** du modèle de base. De cette manière, chaque nouvel audit est potentiellement différent des autres. Tout comme l'infrastructure d'un client n'est pas l'autre.

Un audit Securicheck est découpé en plusieurs couches. Au-dessus, nous avons les grandes **sections** comme 'Sécurité', 'Backup', 'Maintenance',... composées de **sous-sections** comme 'Serveurs', 'Réseau',... Grâce à cette segmentation, nous pouvons auditer de manière structurée en vérifiant différents points distinctivement. En dernière couche se trouve les tests sous forme d'une question.

Un test est défini par :

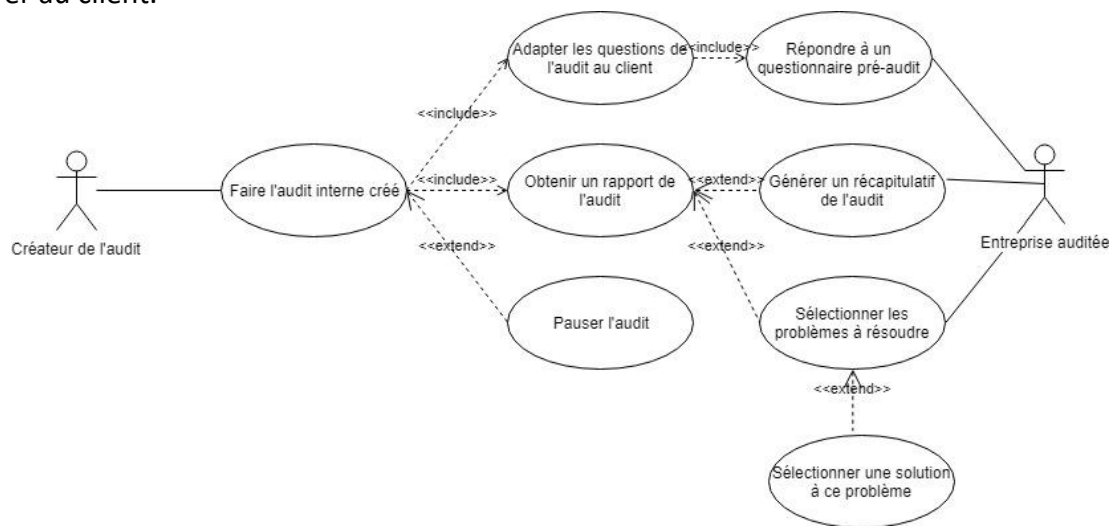
- Une **priorité** : Haute, Moyenne, Faible
- Un **type** comme une question de pré-audit
- La **sous-section** à laquelle il se réfère
- Un **état** : échec, passe avec erreurs, passe
- Un **parent** pour les tests qui dépendent de la réponse à un autre test. Effectivement, vérifier les règles d'un pare-feu n'aurait pas de sens si l'entreprise n'en a pas mis en place. Pour répondre à ce problème, j'ai pensé à un **système de famille**. Tous les tests dépendant d'un autre sont considérés comme des enfants de celui-ci et sont par défaut négatifs et inchangeables tant que le test parent n'a pas été validé.
- Un **commentaire** optionnel comme une erreur rencontrée

Il est possible de **mettre un audit en pause** afin de l'achever ultérieurement au dernier test répondu.

Une fois ce dernier achevé, un rapport est généré qui comprend plusieurs choses :

1. Un **pourcentage** en fonction de la priorité et de l'état des tests passés par section. Un test de priorité Haute vaut trois points, de priorité Moyenne vaut deux points et de priorité Faible vaut un point. Les points des tests avec l'état 'passe avec erreurs' sont divisés par deux. On remarque l'importance des priorités dans le résultat. Sans elles, les résultats ne seraient pas réalistes.
2. Un score général qui est la moyenne des pourcentages ci-dessus
3. Une liste de tous les tests effectués avec leur état

Dans le cadre d'un audit interne, il est possible de sélectionner manuellement avec le client **une solution** aux tests échoués. Lors cette sélection, un rappel des priorités est en place afin d'aider à faire les choix adaptés. On peut conclure l'audit en **exportant le rapport au format PDF** afin de l'envoyer au client.



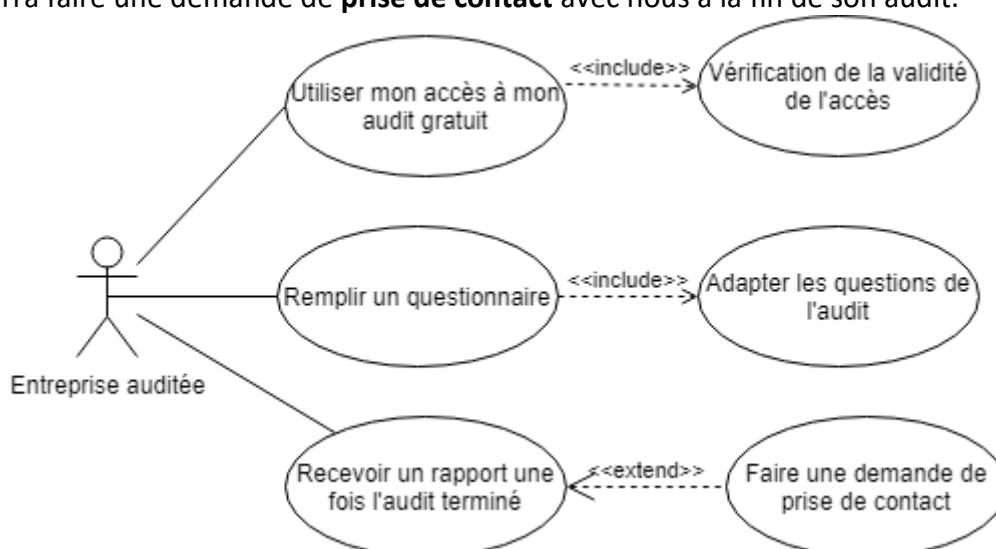
En tant que	Je voudrais	Afin de	Critères d'acceptation
Créateur de l'audit	Un bouton	Démarrer un audit dont je suis créateur	- L'audit n'est pas entrain d'être effectué par un autre créateur
Créateur de l'audit	Remplir un questionnaire pré-audit	Adapter les questions de l'audit à l'entreprise auditée en fonction des réponses	
Créateur de l'audit	Pauser l'audit	Reprendre l'audit plus tard si je n'ai pas le temps de le finir	

Créateur de l'audit	Des pourcentages dans le rapport en fonction de la priorité des tests	Évaluer la sécurité informatique de l'entreprise auditée par ordre d'importance
Créateur de l'audit	Une liste organisée des tests effectués	Sélectionner une solution
Créateur de l'audit	Un récapitulatif de l'audit	Créer un rapport à envoyer au client

2.1.5 Créer un nouvel audit externe

Les audits externes seront envoyés à la demande après approbation. Le client recevra un lien qui l'amènera vers un portail qui lui donnera accès à son audit sur base d'un code unique fourni. Il ne sera valable que 30 jours. Passé ce délai, il devra refaire une demande d'audit.

Le déroulement d'un audit externe sera relativement similaire à celui d'un audit interne. Les différences principales résident dans le fait qu'un autre jeu de question sera utilisé et que les rôles seront inversés. C'est le demandeur qui fait son audit lui-même sans intervention directe de notre part. Il pourra faire une demande de **prise de contact** avec nous à la fin de son audit.



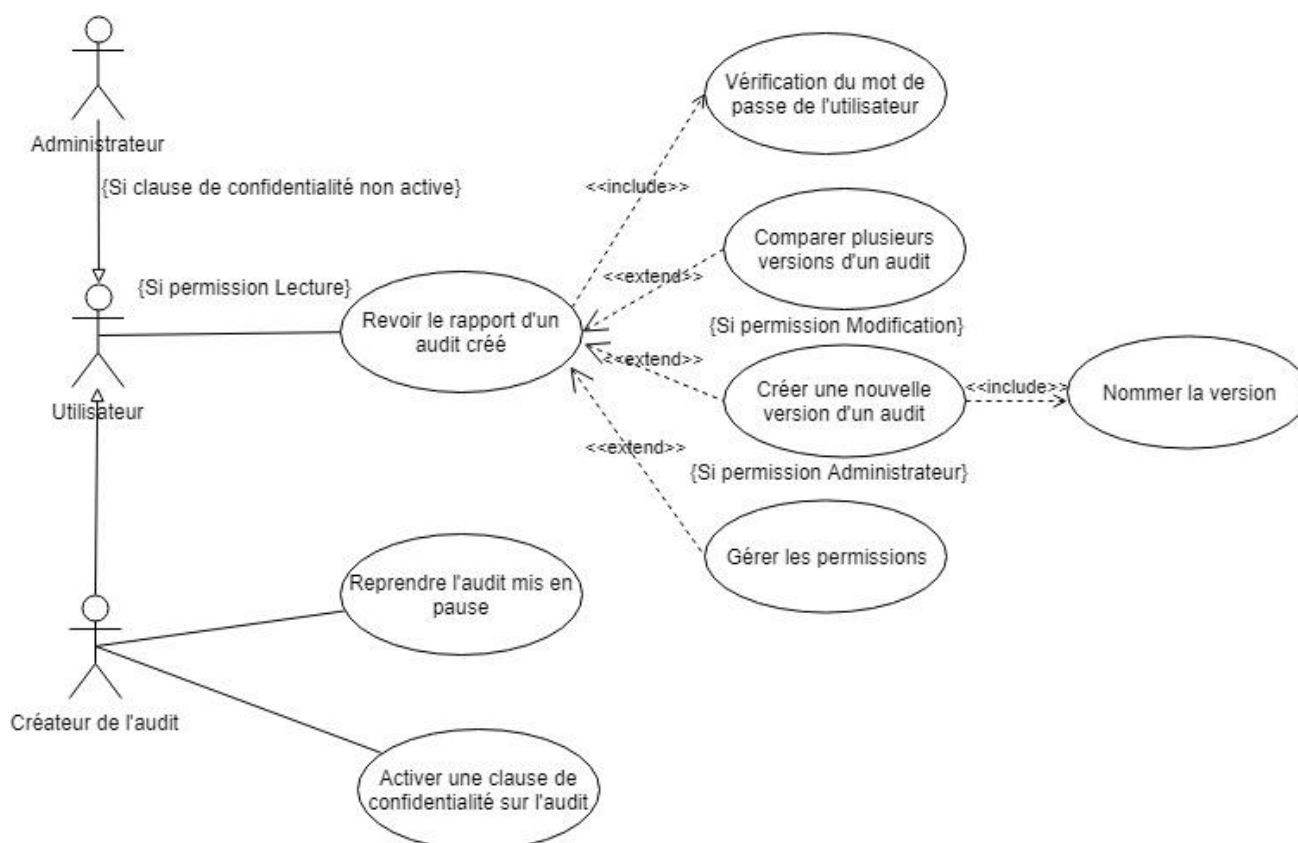
En tant que	Je voudrais	Afin de	Critères d'acceptation
Entreprise Auditée	Recevoir un lien et un code	Accéder à mon audit gratuit personnel	<ul style="list-style-type: none"> - Avoir fait une demande d'audit gratuit - Avoir été approuvé
Entreprise Auditée	Remplir un questionnaire	Adapter les questions de l'audit à ma situation	<ul style="list-style-type: none"> - Faire son audit endéans les 30 jours qui suivent la réception du lien
Entreprise Auditée	Un rapport de l'audit	Connaître les risques qui pèsent sur mon parc informatique	

Entreprise Auditée	Faire une demande de prise de contact avec nous après l'audit	Donner suite à l'audit gratuit
---------------------------	---	--------------------------------

2.1.6 Revue des audits internes créés

Par mesure de sécurité, l'utilisateur est amené à valider son mot de passe avant de pouvoir rentrer dans tout audit terminé. Pour chaque audit interne créé est, suivant l'assignation des permissions, possible de :

- **Revoir le rapport**
- **Créer une deuxième version** pour contre-auditer un client avec les mêmes questions que le premier audit créé pour celui-ci
- **Comparer** deux versions d'un audit
- **Gérer les permissions** des autres utilisateurs sur cet audit
- **Activer une clause de confidentialité** qui annule l'implication du rôle pour les accès à cet audit. Normalement, les administrateurs désignés peuvent visualiser les audits créés sans avoir les permissions nécessaires sur l'audit.



En tant que	Je voudrais	Afin de	Critères d'acceptation
Utilisateur	Un bouton	Accéder au rapport d'un audit terminé	<ul style="list-style-type: none"> - Avoir des permissions en lecture sur l'audit - S'être réauthentié

Utilisateur	Créer une deuxième version de l'audit	Effectuer un contre-audit	- Avoir des permissions de modification
Utilisateur	Comparer les versions d'un audit	Visualiser les changements après un contre-audit	- Existence d'une deuxième version de l'audit
Créateur de l'audit	Reprendre un audit mis en pause	Afin de le finir	- Un autre créateur ne travaille actuellement pas dessus
Créateur de l'audit	Activer une clause de confidentialité	Limiter l'accès à l'audit aux permissions	

2.1.7 Administration de l'application

La partie administration est le noyau de l'application. C'est de celle-ci que va dépendre son fonctionnement car elle permet de gérer ses utilisateurs, le contenu de l'audit et les clients. Afin d'accéder à ses fonctionnalités, il faut disposer **des droits administrateurs adéquats** ainsi que **revalider son mot de passe** afin d'éviter qu'un individu ayant accès à la session d'un administrateur puisse s'y connecter. **Le journal des événements** est disponible respectivement dans chaque partie.

L'**administration des utilisateurs** est utilisée afin de pouvoir ajouter un nouvel utilisateur, modifier ses droits, réinitialiser son mot de passe ou désactiver/réactiver son compte. À noter **que désactiver un compte supprime ses permissions**.

En tant que	Je voudrais	Afin de	Critères d'acceptation
Administrateur Utilisateur	Un lien	Accéder à l'administration des utilisateurs	- Valider son mot de passe
Administrateur Utilisateur	Un bouton	Ajouter un nouvel utilisateur	- L'adresse mail de l'utilisateur n'existe pas déjà
Administrateur Utilisateur	Modifier les rôles des utilisateurs	Gérer leurs droits	- Ne pas peut pas modifier les administrateurs globaux et le super administrateur
Administrateur Utilisateur	Un bouton	Désactiver ou réactiver un utilisateur	- Idem que ci-dessus
Administrateur Utilisateur	Avoir un registre des activités récentes	Visualiser qui a fait quoi et quand	

L'**administration du contenu de l'audit** est responsable de la maintenance du contenu des questionnaires d'audit. À tout moment, uniquement un jeu est considéré comme actif. Ce dernier est susceptible d'être altéré afin d'être mis à jour par les actions suivantes directement depuis l'application :

- **Ajouter/Modifier/Supprimer/Modifier l'ordre des sections**
 - **Ajouter/Modifier/Supprimer/Modifier l'ordre des sous-sections** au sein des sections
 - **Ajouter/Modifier/Supprimer/Modifier l'ordre des tests** au sein des sous-sections

Cependant, **un principe clé** de l'application est que pour chaque audit, il faut être capable de retrouver exactement son contenu. Or, toute modification d'une couche (sections, sous-sections et tests) modifie le jeu de question actif. Les actions décrites ci-dessus impliquent donc **un archivage couche par couche**.

Explication :

- Mettre à jour une couche existante équivaut à en créer une nouvelle et à archiver l'ancienne. Vu qu'on ne peut la supprimer de la base de données intégralement sous peine de devoir supprimer tous les liens avec celle-ci. Mais cela ne suffit pas. Il faut également **recopier intégralement le contenu de ses couches inférieures dans la nouvelle couche et l'archiver**. Sinon, elles ne seraient plus prises en compte car elles pointent vers l'ancienne couche supérieure et ce lien doit également persister.
- Pour les mêmes raisons de consistance, la suppression d'une couche consiste à **l'archiver logiquement**.

Ce principe d'archivage/recopie me permet d'implémenter une fonctionnalité de '**snapshot**' du jeu de question actif. En d'autres termes, d'en créer une sauvegarde afin de pouvoir y revenir à tout moment dans le futur. Cette fonctionnalité copie les tests actifs dans une table. Lorsque nous voudrions rétablir une certaine version, ces liens me permettront de remonter chaque couche et de les réactiver. Questions de pré-audit incluses.

En tant que	Je voudrais	Afin de	Critères d'acceptation
Administrateur Contenu	Un champ	Créer ou modifier une section	- Ne peut pas être vide ou uniquement composé d'espaces
Administrateur Contenu	Un champ	Créer ou modifier une sous-section	- Ne peut pas être vide ou uniquement composé d'espaces
Administrateur Contenu	Un champ	Créer ou modifier un test	- Ne peut pas être vide ou uniquement composé d'espaces - Par défaut, les tests prennent la priorité 1 - Le type du test est inchangeable après sa création
Administrateur Contenu	Un champ	Créer ou modifier les différents choix des tests type Sélection	- Ne peut pas être vide ou uniquement composé d'espaces - Chaque choix doit avoir un état modifiable, par défaut 'échec'

Administrateur Contenu	Un champ	Créer ou modifier les enfants d'un test parent	- Ne peut pas être vide ou uniquement composé d'espaces
Administrateur Contenu	Un menu déroulant par test	Modifier la priorité de chaque test	
Administrateur Contenu	Modifier l'ordre de chaque couche	Réorganiser la structure du jeu de question actif	
Administrateur Contenu	Un bouton	Sauvegarder la version actuelle du jeu de question	
Administrateur Contenu	Un menu déroulant	Restaurer une version antérieure du jeu de question	
Administrateur contenu	Une liste	Visualiser l'historique des activités de la partie contenu	

2.2 Gestion des clients

Suite à l'entrée en vigueur du **Règlement Général sur la Protection des Données**, il y'a des règles concernant l'utilisation des données collectées. De plus, son propriétaire conserve **des droits** à respecter sur celles-ci. Je me suis référé aux principes prônés par le RGPD pour conformiser mon application et utiliser les données de manière licite.

Par le principe de **minimisation des données**, les informations à caractère personnel de l'entreprise collectées sont réduites au strict nécessaire. À savoir :

- Le **nom de l'entreprise**
- La **personne responsable** dans le cadre de l'audit au sein de l'entreprise auditée
- Le **contrat** signé par le prestataire(nous) et le client pour audit interne
- La **date de réalisation** de l'audit

Le client a le droit de s'opposer au traitement de ses données à caractère personnel pour des raisons sérieuses et légitimes. Il ne peut toutefois pas s'opposer au traitement des données nous étant nécessaires à l'exécution d'une obligation légale que constituent les données mentionnées ci-dessus. Nous devons les conserver car celles-ci attestent de la réalisation de l'audit.

J'ai inclus la possibilité de supprimer définitivement toutes les données en réponse à l'audit pour une entreprise :

- Toutes les réponses **au questionnaire de pré-audit**
- Toutes les réponses de **toutes les versions d'un audit ainsi que les données qui en découlent comme les potentielles solutions sélectionnées lors du résultat**

Par le principe de **limitation des finalités**, les réponses aux audits externes envoyés à des clients potentiels ne seront pas conservées en base de données. Ce type d'audit est un questionnaire différent et ne pourra pas poser des questions trop confidentielles sous peine de faire fuir. Il a pour seule vocation de faire prendre conscience l'audité d'éventuelles lacunes afin de le pousser à venir vers nous. Seulement ce dernier recevra le rapport final qu'il pourra utiliser comme bon lui semble car il ne nous concerne pas dans un premier temps.

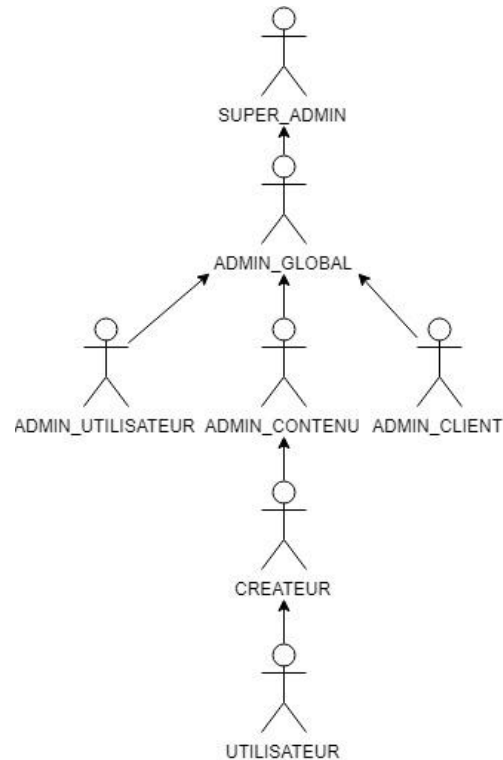
Avant chaque audit, l'audité devra prendre connaissance d'un document à adapter suivant le type d'audit. Celui-ci fait office de contrat et confirme que l'audité est informé sur les conditions d'utilisation de ses données.

2.3 Segmentation des accès

La segmentation des accès est divisée en deux parties. Chaque utilisateur dispose obligatoirement d'un ou plusieurs **rôles**. Ils correspondent à sa fonction dans l'application. La visibilité sur les audits créés est gérée par des **permissions**. Ces deux types d'accès sont complémentaires comme nous allons le voir dans la suite.

2.3.1 Division des rôles

Par défaut, un utilisateur non authentifié est considéré par l'application comme étant **Anonyme**. Un Anonyme ne peut accéder uniquement qu'au portail de connexion principal, quel que soit la page qu'il tente de joindre. Le contrôle des accès aux fonctionnalités de l'application est géré de manière pyramidale par la hiérarchie suivante :



- **Explication :**

Chaque niveau hérite des propriétés du précédent. Tout en haut de l'échelle se trouve le rôle **SUPER_ADMIN**. Il ne peut y en avoir qu'un seul. Celui-ci a les pleins pouvoirs et accès aux audits créés. C'est le directeur de l'application. Il peut choisir de déléguer son rôle à un pool de **ADMIN_GLOBAL** que seul lui peut gérer. Leur accès à un audit terminé peut être coupé par **l'activation de la clause de confidentialité** contrairement au **SUPER_ADMIN**.

Ce rôle est ensuite subdivisé en plusieurs **sous-administrateurs** avec une fonction bien précise :

- Gestionnaire des **utilisateurs**
- Gestionnaire du **contenu de l'audit**
- Gestionnaire des **clients**

Ce sont les managers de l'application. Ils n'ont par défaut pas de vue sur les audits terminés sans les permissions adéquates. Un utilisateur peut avoir **plusieurs rôles sous-admin**.

Le **ROLE_UTILISATEUR** est le minimum de droits attribués à une personne authentifiée. Il limite l'accès au tableau de bord, à la page de profile et aux audits en fonction des permissions qui lui sont accordées. Ce rôle convient par exemple pour des stagiaires et autres personnes externes car il n'autorise même pas la création d'un nouvel audit. Cette fonctionnalité est ajoutée par le **ROLE_CREATEUR** qui ajoute une notion supplémentaire : **créateur d'un audit**.

Chaque audit initialisé a un créateur principal et optionnellement des créateurs secondaires (participants). Ils peuvent démarrer l'audit ou le reprendre s'il est en pause. Actuellement, un créateur à la fois peut réaliser un audit. Celui-ci sera bloqué aux autres créateurs tant que l'audit est occupé. Une piste d'amélioration intéressante serait de pouvoir synchroniser plusieurs utilisateurs sur un audit comme pour l'édition de certains documents en ligne type Google Docs.

2.3.2 Gestion des permissions des audits

Les permissions sur un audit sont gérées suivant le modèle de base **Lecture, Modification et Administrateur** (à ne pas confondre avec les **rôles d'administration**). La permission de lecture permet d'ouvrir le rapport de l'audit principal d'un client ainsi que ses différentes révisions s'il y'en a. La possibilité de créer une révision est implémentée par la permission de **modification**.

La permission **d'administrateur** hérite des fonctionnalités des précédentes et offre en plus la gestion des permissions des utilisateurs sur l'audit. Les **créateurs** de l'audit sont un cran au-dessus et peuvent activer la clause de confidentialité en plus du **SUPER_ADMIN**.

Chapitre 3

Mise en pratique

La partie suivante détaille la mise en pratique de l'analyse ci-dessus. Premièrement, je vais **justifier et expliquer les différents outils** auxquels j'ai fait recours durant le développement. Deuxièmement, je vais parler **de l'implémentation de la base de données** et de ses conséquences. Troisièmement, je vais montrer comment j'ai mis en place **l'authentification et le contrôle des accès**.

3.1 Marche suivie

3.1.1 Côté serveur

J'étais désireux d'employer un framework pour mon application car la plupart des développeurs professionnels en utilisent. Mon choix s'est tourné vers **Symfony 4**. Symfony est un framework **PHP MVC libre**. Il fournit des fonctionnalités modulables qui permettent de faciliter et d'accélérer le développement d'un site web telles que :

- Existence de nombreux plugins
- Des performances optimisées par un système de cache
- Une gestion des URL par un module de routage
- Intégration de l'ORM Doctrine
- Gestion complète des accès par un pare-feu

Symfony a attiré mon attention car lors de mes recherches d'emploi, j'ai remarqué que les développeurs Symfony étaient très demandés. J'ai pensé que c'était l'occasion rêvée pour l'apprendre et l'ajouter à mon CV. De plus, Symfony est très répandu. Il est aujourd'hui bien rodé et offre une documentation exhaustive. J'ai commencé par l'étudier afin d'avoir une vision générale des capacités de l'outil.

3.1.2 Côté client

La programmation cliente de mon application utilise le **HTML** et le **CSS** sur base du framework **Bootstrap 4**. Je l'ai utilisé pour les mêmes raisons que Symfony. Il est facile d'utilisation grâce à un système de classes prédéfinies qui répond aux besoins les plus courants lors de la conception du style d'un site web.

J'ai également vastement utilisé le **JQuery** en collaboration avec mon PHP. J'en ai particulièrement eu besoin pour la programmation de l'administration du contenu de l'audit. La raison est qu'elle nécessite beaucoup de modifications de données. Pouvoir modifier chaque couche séparément en **temps réel** sans devoir soumettre tous les changements en une seule fois est un vrai plus.

La contrainte était d'arriver à identifier la nature de l'élément (section, sous-section,...) et son identifiant. J'y suis parvenu avec un système de **classes CSS**. Chaque couche créée est écrite à l'écran dans un `` avec la classe 'editable' et une classe propre à sa nature. Un **évènement** est attaché à la classe 'editable' pour qu'au clic, le `` soit remplacé par un **champ texte**. Ensuite, j'identifie la nature de l'élément, par exemple une section, grâce à la fonction JQuery `element.hasClass('section')` dans une condition if. Je vérifie également que le champ n'est pas vide, composé uniquement d'espaces ou à la même valeur que l'élément d'origine. Si oui, je ne fais pas de traitement par le serveur et remet le `` à la place du champ.

Une fois identifié, je récupère l'id de l'élément que j'ai préalablement mis dans une div parente suivant une dénomination. Enfin, il ne me reste plus qu'à faire une requête Ajax avec la nature, l'identifiant et la valeur du champ vers le contrôleur adéquat. Il se charge de créer le nouvel élément, d'archiver l'ancien et de recopier tout son contenu dans le nouveau.

Un système similaire est en place pour gérer les réponses à un audit en cours afin de faciliter la synchronisation entre créateurs.

3.2 Symfony 4

3.2.1 Outils employés

- **Le Profiler**

Le Profiler est un **debugger** complet avec une interface graphique. Il facilite la vie et accélère le développement. Grâce à celui-ci, le développeur peut voir les données engendrées par n'importe quelle action (HTTP, POST et GET, SQL...).

- **Le Router**

La gestion des URLs se fait dans Symfony par un **système de routage**. Le router contient une table qui fait correspondre pour chaque route de l'application un contrôleur et une fonction de celui-ci à exécuter. Ensuite, cette fonction retourne la vue demandée. L'accès aux pages du site se fait par un nom de route et pas le nom de la page même. Les URLs sont plus jolies et professionnelles. Les routes jouent également un rôle important dans la sécurité.

Exemple : La racine / utilisée comme page d'accueil est configurée comme route et appelle la fonction qui retourne la page accueil.html. Pour accéder à l'accueil, l'adresse utilisée sera `www.nom.domaine/` au lieu de `www.nom.domaine/accueil.html`. Elles se définissent grâce à un **système d'annotations**. Il suffit, dans notre exemple, de la définir au-dessus de la méthode qui gère la page d'accueil comme ceci :

```
/**
 * @Route("/", name="homepage", methods="GET")
 */
```

- **Générateur de templates Twig**

Les templates sont nécessaires afin qu'un contrôleur puisse mettre à jour la vue avec les données du modèle. PHP lui-même peut être utilisé malgré que ce ne soit pas sa fonction première. Cependant, les deux ne se mélangent pas bien et rendent le code difficile à interpréter. **Twig** est une réponse efficace à ce problème. Il est optimisé pour cette fonctionnalité et sa syntaxe allège le code.

En repartant de notre exemple précédent, le contrôleur va chercher dans le bon modèle les données nécessaires à l'affichage de la page d'accueil et Il finit la requête par demander à Twig de générer la page avec la méthode prévue à cet effet :

```
return $this->render('user/homepage.html.twig', $array);
```

Le paramètre **\$array** est un tableau qui contient les données du modèle que Twig va interpréter grâce à son propre langage. Celui-ci permet de faire des boucles **For**, conditions **If** et **afficher les données en provenance du contrôleur** avec un syntaxe particulière.

Une autre fonctionnalité utile de Twig est de pouvoir créer un **template de base** sur lequel d'autres templates reposent. En effet, certaines parties comme la navigation, le pied de page ou l'en-tête sont souvent communes à toutes les pages. J'ai créé un fichier de base contenant ce code que je peux appeler les autres pages. Une modification du fichier de base se répercute dans toutes les pages qui en héritent.

- **L'ORM Doctrine**

Doctrine est un **ORM** (Object Relational Mapping) basé sur l'API **PDO**. Il permet de créer des entités (objets) et de les convertir en base de données relationnelle (voir **annexe B** pour schéma explicatif). Créer ou modifier une entité se fait dans un terminal par la commande PHP :

```
$ php bin/console make:entity
```

La commande demande ensuite le **nom de l'entité** et quels sont ses **champs**. Ils ont un type (entier, chaîne de caractère, booléen,...) ou sont liés à d'autres entités par une relation **ManyToOne**, **OneToMany** ou **ManyToMany**.

Une fois l'entité terminée, Doctrine crée un fichier de migration avec la commande :

```
$ php bin/console make:migration
```

Qu'il utilise pour mettre à jour la base de données grâce à la commande :

```
$ php bin/console doctrine:migrations:migrate
```

Le résultat est que la classe et la table correspondante dans la base de données sont créées ou mise à jour de manière homogène.

Doctrine est prévu pour faire **des conversions bidirectionnelles**. Il est également capable d'aller chercher des données de la base et de les traduire en objets grâce à ses méthodes :

- **findAll()** qui converti en objets toutes les entrées d'une table
- **findOneBy()** qui converti en objet une entrée d'une table sur base d'un ou plusieurs paramètres
- **findBy()** qui converti en objets toutes les entrées d'une table sur base d'un ou plusieurs paramètres

Ces trois méthodes permettront de gérer la plupart des requêtes SQL.

3.2.2 Structure d'un projet Symfony

Nativement, un nouveau projet Symfony est structuré en plusieurs dossiers avec chacun un but précis.

- Dossiers Public, Src et Template :

Ils représentent les trois dossiers de développement principaux. Le dossier **Public** est la partie visible par les utilisateurs. Pour plus de visibilité, j'ai découpé le mien en **js** et **css** subdivisés suivant les différentes parties de mon application (administration, audit,...), **fonts**, **images** (photos de profile des utilisateur,...) et **vendors** pour les fichiers de plugins.

Le dossier **Src** est la partie **back-end** de l'application. Il réunit mes **contrôleurs** :

- **Utilisateur**
- **Audit**
- **Client**
- **Administration**
- **Sécurité** (qui gère certains accès)

Et mes **entités** qui correspondent aux **modèles**.

Le dossier **Template** est réservé aux fichiers Twig qui abritent la **vue** de l'application.

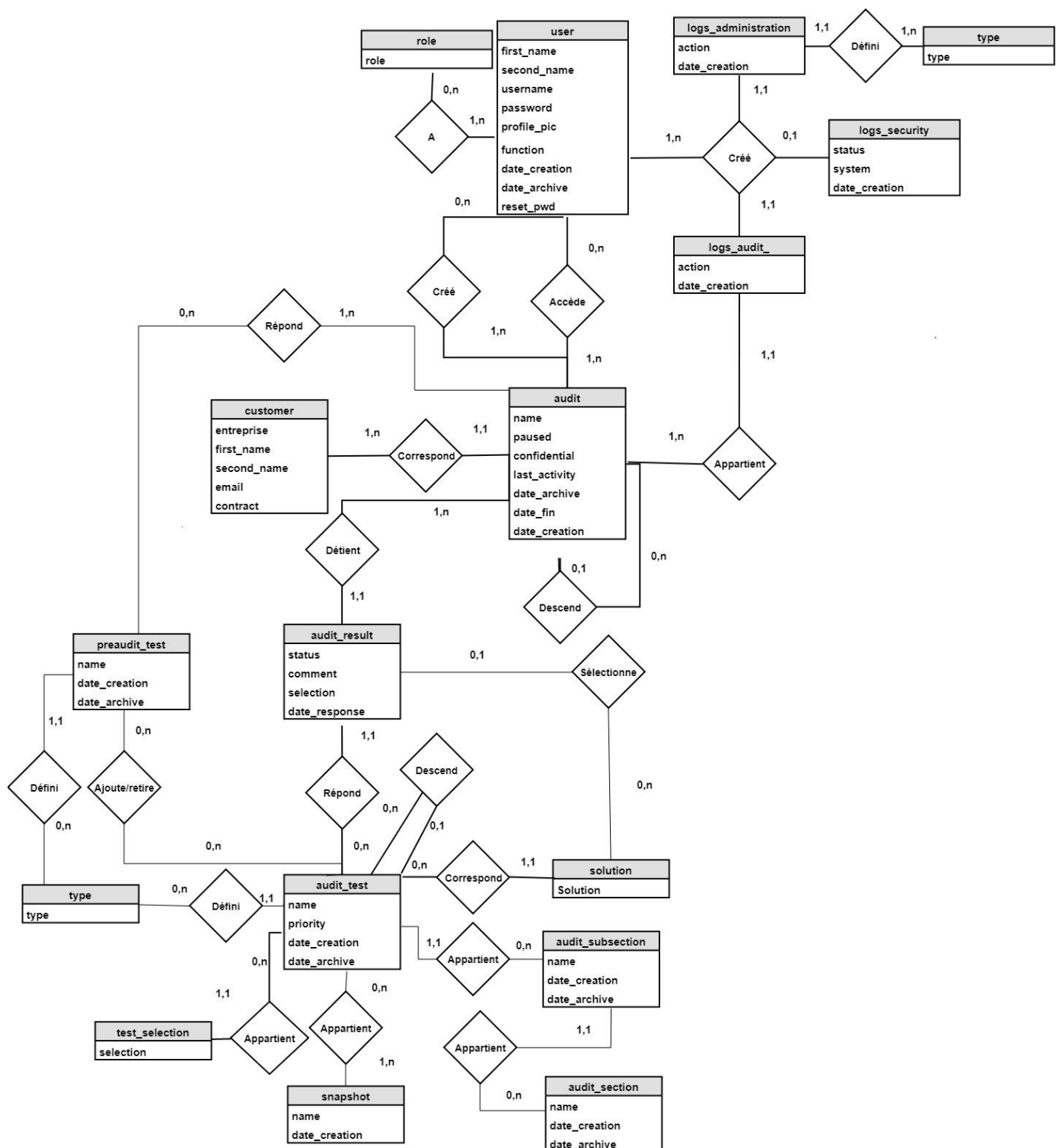
• Dossiers config et var :

On retrouve dans le premier certains fichiers de configuration sous un format YAML comme la configuration du pare-feu. Le dossier **var** contient les fichiers de logs et le cache de l'application.

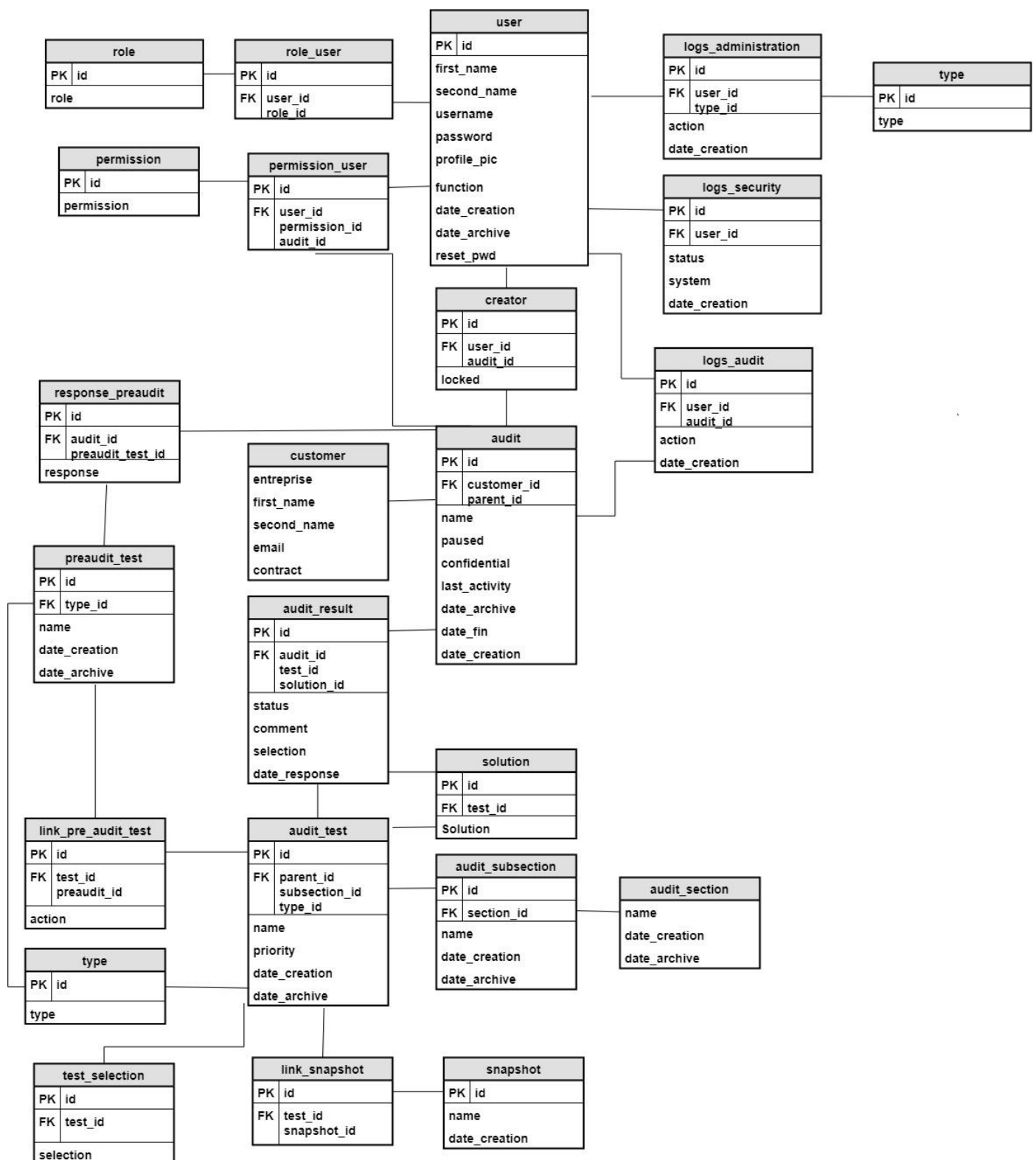
3.3 Intégration de la base de données dans les fonctionnalités

Nous avons vu dans l'analyse quels sont les besoins fonctionnels et de gestion des accès. Intéressons-nous maintenant à l'implémentation de la base afin d'y répondre.

3.3.1 Schéma entité-association



3.3.2 Schéma relationnel



3.3.3 Gestion des évènements

Par mesure de sécurité, **il ne peut pas il y'avoir d'activités importantes anonymes**. Pour toute action un minimum critique, il faut savoir quel utilisateur est à l'origine de celle-ci ainsi que la date d'exécution.

J'ai défini plusieurs types d'évènement : **administratif, audit et sécurité**. Les évènements administratifs sont relatifs aux action entreprises dans l'administration de l'application. Ils ont chacun un type qui fait référence à une section de l'administration : **utilisateur, contenu de l'audit et client**. **Chaque action de la partie administration est enregistrée et catégorisée suivant sa partie.**

Les actions du module d'audit suivantes sont enregistrées lorsque :

- Quelqu'un **créé un audit**
- Quelqu'un **fini un audit**
- Quelqu'un **créé un contre-audit**
- Quelqu'un **assigne des permissions à un utilisateur sur un audit**

Les **évènements de sécurité** répertorient les **tentatives de connexion fructueuses et infructueuses** sur de portail de connexion générale et sur celui d'administration.

Cette manière me permet d'optimiser la segmentation des accès. En effet, le super administrateur et les administrateurs globaux ont accès à l'entièreté des journaux tandis que les sous-administrateurs auront accès uniquement aux parties pour lesquelles ils ont été désignés. Il en va de même pour les utilisateurs gestionnaires d'audits.

3.3.4 Suppression logique

Comme expliqué précédemment, la suppression des données **est logique et non définitive**. Avec l'exception de la conformisation au RGPD. Sinon, je serais forcé de supprimer en plus toutes les données liées à l'entité supprimée.

Pour connaître si une donnée n'est plus en activité, je vérifie son champ `date_archive`. S'il est nul, elle considérée comme active. Toute donnée archivable, est complétée par **une date de création**.

3.3.5 Gestion des opérations sur les audits

Une combinaison de **trois champs** est employée pour gérer la synchronisation d'un audit entre ses différents créateurs. Le champ **paused** sert à savoir si un audit est en pause. Le champ **date_fin** permet de connaître s'il a déjà été terminé. Il s'initialise lorsqu'un créateur a validé le rapport. Sinon, l'audit reste accessible en modification jusqu'à changement de cet état. Par contre, une fois achevé, les réponses de l'audit d'origine sont changeables qu'en opérant un contre-audit.

Par déduction, si l'audit n'est pas fini et qu'il n'est pas en pause, c'est qu'un créateur le réalise. Cela implique pour les autres créateurs **qu'il soit verrouillé afin d'éviter les conflits**. Cependant, une autre question se pose : comment empêcher qu'un utilisateur bloque indéfiniment un audit ?

Grâce à la programmation orientée client par le JQuery, chaque réponse est traitée immédiatement. Je sais tracer la date de la dernière action depuis que l'audit a été débuté ou repris. Le troisième champ dont j'ai fait allusion qui est **last_activity** contient cette donnée. Si elle dépasse une heure, je considère la session comme expirée et réouvre l'audit aux autres créateurs.

3.3.6 Gestion des résultats d'un audit

Les résultats d'un audit sont contenus dans la table **audit_result**. Elle a un lien très fort avec la table **audit_test**. En effet, on remarque sur le schéma que cette dernière est au centre du module d'audit. Je peux faire le lien avec toutes les autres tables qui contiennent le contenu de l'audit par celle-ci. Le champ **'status'** enregistre l'état du test référencé, **'comment'** le commentaire optionnel et **'selection'** est la réponse à un choix multiple si le test est de type Sélection.

Le lien avec la table **solution** enregistre les solutions qui ont été sélectionnées lors du résultat de l'audit.

Les réponses au questionnaire de pré-audit sont dans le lien entre la table **preaudit_test** et **audit**.

Après la soumission de celui-ci, la table de résultat est directement mise à jour avec des états par défaut. Si je ne le fais pas, des problèmes vont se poser si le questionnaire pré-audit est mis à jour avant que l'audit ne soit terminé. Si une des questions est archivée entre temps, je risque de perdre un lien qui ajoute ou retire des tests.

3.4 Implémentation de la sécurité

Cette partie décrit la sécurité de l'application. Je vais d'abord expliquer comment je l'ai protégée des failles web connues en me référant au cours d'Openclassrooms sur les failles de sécurité web. Ensuite, je vais montrer mon utilisation du pare-feu de Symfony pour l'authentification et le contrôle des accès.

3.4.1 Injections SQL et XSS

Ces failles web connues peuvent être extrêmement dangereuses. Elles permettent à un attaquant d'injecter une requête SQL ou un script Javascript afin d'exécuter du code illicite. Elles exploitent les portes d'entrée vers la base de données comme les champs textes et paramètres GET.

Toutes les méthodes '**find**' de Doctrine sont parées nativement contre les injections SQL. Je n'ai pas fait l'usage de requêtes personnalisées. Si cela venait à venir, je prendrais soin de la **préparer avec PDO**.

Je me suis également protégé des failles XSS en échappant les caractères spéciaux avec **htmlspecialchars()** pour que les balises <scripts></scripts> soient considérées comme du texte et non des balises HTML.

3.4.2 Spam de formulaire de connexion et force brute

La soumission du formulaire de connexion est contrôlée par l'**API reCAPTCHA** de google. Elle est considérée comme fiable et utilisée par nombreux sites. Elle protège l'application en empêchant l'envoi automatisé du formulaire de connexion.

3.4.3 Authentification et gestion des accès grâce au pare-feu Symfony

Le pare-feu de Symfony fait partie du bundle Security de SensioLabs. C'est un outil très puissant qui gère l'authentification et les accès. Je vais expliquer comment je l'ai configuré pour répondre aux besoins de mon application sur base du fichier de configuration :

```
security:
    encoders:
        App\Entity\AppUser:
            algorithm: bcrypt
    providers:
        app_db:
            entity:
                class: App\Entity\AppUser
                property: username
    firewalls:

        administration_login:
            pattern: ^/administration
            form_login:
                login_path: login_admin
                check_path: login_admin_check
            provider: app_db
            anonymous: false

        main_login:
            pattern: ^/
            anonymous: ~
            form_login:
                login_path: login
                check_path: login_check
```

```
csrf_token_generator: security.csrf.token_manager
provider: app_db
logout:
    path:    logout
    target: /
```

Encoders

‘L’**encoder**’ est la **méthode de hachage** du mot de passe que le pare-feu utilise lors de l’authentification. Il en existe plusieurs mais **bcrypt** est actuellement considéré comme un des meilleurs. Il inclut un salage du mot de passe pour contrer les attaques type ‘rainbow table’.

Providers

Le ‘**provider**’ est la banque de données qui fournit les utilisateurs au pare-feu. Il peut être de différents types comme une base de données ou un annuaire LDAP. Dans mon cas, les utilisateurs sont gérés dans une base de données SQL. Les informations de connexion à la base de données sont configurées dans un fichier à part. Les clés **class** et **property** sont la classe et le champ à utiliser pour l’authentification.

Firewalls

C’est dans cette partie que l’on va configurer les différentes zones sécurisées de l’application.

- **Le pare-feu main_login :**

Le pare-feu **main_login** est chargé de contrôler l’authentification au portail de connexion principal. La clé **pattern** définit sur quelle URL le pare-feu est en activité. Le symbole ^ indique de commencer l’URL à partir de la racine du serveur. Comme il est en charge du portail de connexion général, il doit être ouvert à tous les utilisateurs. La clé **anonymous** autorise les anonymes à le contacter.

Ensuite, il faut définir par quel moyen l’utilisateur va s’authentifier. Il y’a plusieurs manières de créer un identificateur. La solution utilisée dans ce cas est un formulaire de connexion. La clé **login_path** est le nom de la route à utiliser afin de générer le template de connexion lorsqu’un utilisateur tente de s’authentifier. J’ai regroupé toutes les routes responsables de la gestion de l’authentification dans un contrôleur à part. Lorsque l’utilisateur a soumis le formulaire avec ses identifiants, la clé **check_path** appelle la route qui exécute le code de vérification sur base de deux champs qu’il faut nommer **_username** et **_password**. Tout le processus a l’air magique car il se fait relativement automatiquement. Il n’y a qu’à adapter à ses besoins. Le pare-feu gère lui-même la déconnexion lorsque l’utilisateur se rend sur la route définie dans la clé **logout**. Voici la méthode dans mon contrôleur qui est responsable de la vérification pour ce pare-feu :

```

public function loginCheck(AuthenticationUtils $authenticationUtils)
{
    $securityContext = $this->container->get('security.authorization_checker');
    if (!$securityContext->isGranted('IS_AUTHENTICATED_REMEMBERED')) {
=> Cette ligne vérifie si, sur base des informations rentrées par l'utilisateur, il dispose au moins du statut authentifié.
        $error = $authenticationUtils->getLastAuthenticationError();

        $lastUsername = $authenticationUtils->getLastUsername();
=> Si il ne l'a pas, il est redirigé vers le formulaire de connexion.
        return $this->render('user/login.html.twig', array(
            'last_username' => $lastUsername,
            'error'         => $error,
        ));
    }
    else {
        return $this->redirectToRoute('homepage');
    }
}

```

- Le pare-feu administration_login :

Il définit la zone sécurisée de l'administration de l'application. Il ne prend le dessus sur le pare-feu principal que pour les URLs commençant par /administration. Il est plus sécurisé et **bloque entièrement l'accès aux anonymes**. Il force la réauthentification d'un utilisateur déjà authentifié par le pare-feu principal pour profiter de ses droits administrateurs. La différence dans les méthodes de vérification avec ce pare-feu est qu'il vérifie que l'utilisateur authentifié a bien le rôle administrateur adéquat. Le premier s'assure juste que l'utilisateur est un utilisateur reconnu.

Gestion des accès

Symfony laisse libre choix au développeur de nommer ses rôles. Il est juste préconisé de les commencer par le préfixe ROLE_. J'ai appliqué l'analyse pour configurer ma hiérarchie de rôles dans le pare-feu :

```

role_hierarchy:
    ROLE_CREATOR: ROLE_USER
    ROLE_ADMIN_USER: ROLE_CREATOR
    ROLE_ADMIN_CONTENT: ROLE_CREATOR
    ROLE_ADMIN_CUSTOMER: ROLE_CREATOR
    ROLE_GLOBAL_ADMIN: [ROLE_ADMIN_USER, ROLE_ADMIN_CONTENT,
ROLE_ADMIN_CUSTOMER]
    ROLE_SUPER_ADMIN: ROLE_GLOBAL_ADMIN

```

Chaque rôle hérite du précédent. Le SUPER_ADMIN aura donc également les rôles GLOBAL_ADMIN, ADMIN_USER, ADMIN_CONTENT, ADMIN_CUSTOMER, CREATOR et USER.

Le pare-feu Symfony gère les accès de plusieurs manières possibles sur base de ces rôles :

1. Par une **Access List** pour protéger une liste d'URL. Symfony commence tout en haut de la liste et accorde l'accès à la première règle validée. J'ai implémenté mon routage de manière à exploiter cette méthode stratégiquement. Mes routes sont découpées suivant la fonctionnalité. Par exemple :
 - Toute ma partie administration commence par /administration/ pour ensuite se diviser /utilisateurs/..., administration/contenu-audit/... etc.

Mon access list :

- { **path:** ^/connexion, **roles:** IS_AUTHENTICATED_ANONYMOUSLY }
=> URL utilisée derrière le nom de route 'login' employé par l'authentification par le pare-feu principal. Elle est uniquement accessible aux anonymes. Celui-ci est redirigé automatiquement vers cette URL tant qu'il n'est pas authentifié.
- { **path:** ^/\$, **roles:** ROLE_USER } => Restreint la page d'accueil aux utilisateurs.
- { **path:** ^/utilisateur/profile, **roles:** ROLE_USER } => Restreint la page de profile aux utilisateurs.
- { **path:** ^/audits-crées, **roles:** ROLE_USER }
- { **path:** ^/audit/créer, **roles:** ROLE_CREATOR } => Autorise la création d'un nouvel audit aux créateurs.
- { **path:** ^/administration/utilisateurs, **roles:** ROLE_ADMIN_USER } => Limite l'accès à l'administration des utilisateurs.
- { **path:** ^/administration/contenu-audits, **roles:** ROLE_ADMIN_CONTENT } => Limite l'accès à l'administration du contenu de l'audit.
- { **path:** ^/administration/clients, **roles:** ROLE_ADMIN_CUSTOMER } => Limite l'accès à l'administration des clients.
- { **path:** ^/administration/clients/supprimer, **roles:** ROLE_GLOBAL_ADMIN } => Limite la possibilité de supprimer définitivement les données d'un client aux administrateurs globaux.

2. Depuis un contrôleur.
3. Depuis un template Twig pour adapter la vue avec la fonction intégrée `is_granted('role')`. J'ai utilisé cette méthode pour :
 - **Supprimer le bouton de création d'un audit** pour le ROLE_USER
 - **Supprimer le journal des événements principal** du tableau de bord pour les rôles en dessous de ADMIN_GLOBAL
 - Empêcher les ADMIN_USER et GLOBAL_ADMIN d'assigner le rôle GLOBAL_ADMIN ou SUPER_ADMIN à un utilisateur
 - Empêcher les ADMIN_USER et GLOBAL_ADMIN de changer le rôle ou supprimer un GLOBAL_ADMIN
 - Empêcher les ADMIN_USER et GLOBAL_ADMIN de changer le rôle ou supprimer le SUPER_ADMINISTRATEUR

Chapitre 4

Conclusion

4.1 Résumé

Securichек est un outil de gestion d'audits informatiques que j'ai développé pour la société Anderson dans laquelle je travaille. Il se base sur une structure en couches qui répond aux besoins de mon client. Chaque audit est sur mesure grâce à un questionnaire pré-audit qui va ajuster les tests à employer en partant d'un modèle de base. Il propose des fonctionnalités sur la création d'un audit comme sa mise en pause ou encore la génération de rapports personnalisés. De plus, il offre une continuité sur la durée grâce à des fonctionnalités de révision.

C'est avant tout un gain de temps pour notre activité. Il s'aligne parfaitement avec celle-ci. Grâce à cette application, nous pourrions avoir instantanément une vue générale sur l'utilisation de l'informatique de nos clients et prendre des mesures adéquates si nécessaire.

D'abord, j'ai mis un point d'honneur à ce qu'elle soit facilement maintenable et durable dans le temps. J'ai réfléchi à la nature de chaque donnée et à l'implication de leur modification. Le contenu de l'audit est entièrement personnalisable. Pour pouvoir retrouver le contenu de n'importe quel audit, j'ai procédé à un système d'archivage qui repose sur des dates de création et d'archivage. De cette manière, je peux connaître le jeu de questions actif sans devoir altérer le résultat des audits effectués avec un jeu de question antérieur.

Ensuite, j'ai accordé une importance particulière à la sécurité et aux interactions avec les utilisateurs. Ceux-ci sont tous dotés d'un ou plusieurs rôles qui définissent ce qu'ils peuvent ou ne peuvent pas faire. Ils sont segmentés hiérarchiquement et correspondent à une fonction bien précise afin d'envisager différents cas de figure. Des pare-feu Symfony sont les entités responsables de s'assurer du respect des accès.

Mais il fallait un système à part un peu plus élaboré pour régir les accès aux audits créés. J'ai mis en place un système de permission divisé en trois droits : lecture, modification et administrateur. Il est indépendant du rôle et chaque audit dispose de sa propre gestion des utilisateurs.

Afin qu'un utilisateur ne puisse pas faire une action critique sans être identifié, j'ai mis en place une gestion des événements selon trois types :

- Administratif pour les événements propres à l'administration de l'application
- Sécurité qui répertorie les tentatives de connexion aux différents portails
- Audit pour les événements importants sur les audits

Enfin, j'ai pris le temps de conformiser mon application au Règlement Général sur la Protection par l'analyse de ses principes.

4.2 État d'avancement et remarques personnelles

Ce travail m'a beaucoup appris. J'ai pu développer mes connaissances techniques par l'apprentissage des frameworks Symfony et Bootstrap. Symfony facilite tous les aspects de la programmation d'un site web par l'intégration de ses outils et sa flexibilité. Par exemple, son débbugger accélère grandement la résolution de problèmes. Symfony m'a également permis d'être beaucoup plus familiarisé à l'architecture MVC.

D'un point de vue personnel, Il m'a poussé à réfléchir plus loin pour que mon application tienne la route et soit professionnelle. Il y'a beaucoup de tenants et aboutissants à la problématique de ce travail qui n'apparaissent pas en surface. Ce fut une expérience intéressante de les découvrir au fur et à mesure.

Actuellement, j'ai programmé tout le noyau de l'application sur base de ce rapport. Il me manque encore certaines fonctionnalités mais l'essentiel est mis en place et réfléchi pour être intégré facilement. La grande partie manquante est le développement du module d'audits externes évoqué dans l'analyse. Il partage certaines fonctionnalités avec le module interne mais dispose de ses propres spécificités. Je pense qu'elle constitue une perspective d'avenir pour cette application une fois qu'elle sera rodée.

4.3 Démarche critique

Mon grand ennemi dans la réalisation du projet a été le calendrier. En effet, travaillant à plein temps, j'ai eu énormément de mal à rester dans le timing. J'aurais aimé aller plus loin et, par exemple, mettre en place un système d'historique des évènements plus poussé avec plus de lien vers d'autres tables pour mieux cibler.

Chapitre 5

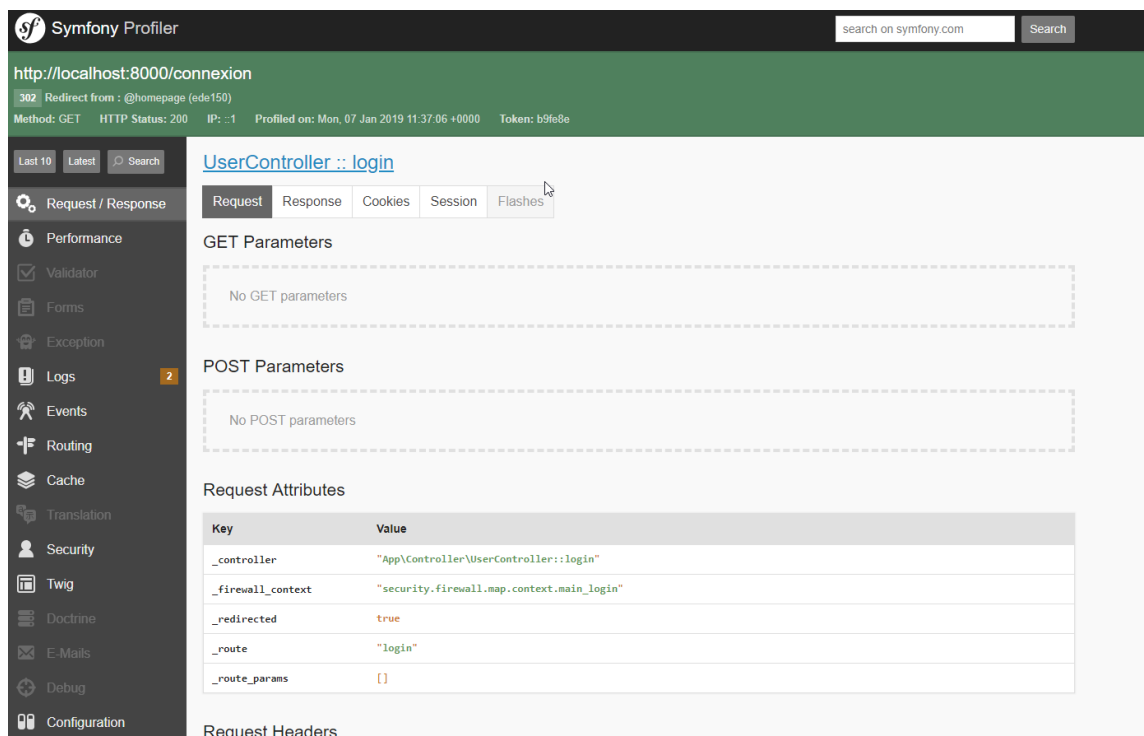
Webographie

- **SensioLabs**, *documentation Symfony*. <https://symfony.com/doc/current/index.html>
- **Refsnes Data**, *W3Schools*. <https://www.w3schools.com/>
- **Bootstrap**, *documentation Bootstrap*. <https://getbootstrap.com/docs/4.2/getting-started/introduction/>
- **Chaîne YouTube Lucidchart**, *UML Use Case Diagram Tutorial*. <https://www.youtube.com/watch?v=zid-MVo7M-E>
- **JGraph**, *application de création de diagrammes Draw.io*. <https://www.draw.io/>
- **OpenClassrooms**, *Protégez-vous efficacement contre les failles web*. <https://openclassrooms.com/fr/courses/2091901-protegez-vous-efficacement-contre-les-failles-web>
- **Commission Européenne**, *Protection des données*. https://ec.europa.eu/info/law/law-topic/data-protection_fr

Chapitre 6

Annexes

Annexe A : Exemple Profiler, requêtes GET et POST



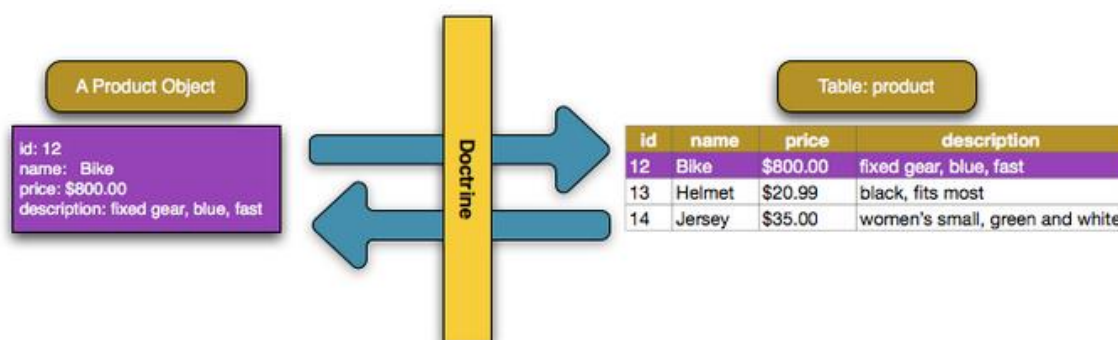
The screenshot shows the Symfony Profiler interface for a request to `http://localhost:8000/connexion`. The request method is GET, and the status is 200. The profiler shows the following details:

- Request / Response:** Request, Response, Cookies, Session, Flashes.
- GET Parameters:** No GET parameters.
- POST Parameters:** No POST parameters.
- Request Attributes:**

Key	Value
<code>_controller</code>	<code>"App\Controller\UserController::login"</code>
<code>_firewall_context</code>	<code>"security.firewall.map.context.main_login"</code>
<code>_redirected</code>	<code>true</code>
<code>_route</code>	<code>"login"</code>
<code>_route_params</code>	<code>[]</code>

Request Headers

Annexe B : Fonctionnement de Doctrine



Source : documentation Symfony

Annexe C : Principes et droits RGPD appliqués*

- Principe de « loyauté de transparence » :

*‘Les données à caractère personnel doivent être traitées de **manière licite et transparente**, en garantissant la loyauté envers les personnes dont les données à caractère personnel sont traitées’*

- Principe de « limitation des finalités » :

*‘Il doit y avoir des **finalités spécifiques** pour traiter les données et l'entreprise/organisation doit indiquer ces finalités aux personnes concernées lorsqu'elle collecte leurs données à caractère personnel; une entreprise/organisation ne peut pas simplement collecter des données à caractère personnel pour des finalités non déterminées’*

- Principe de « minimisation des données » :

*‘L'entreprise/organisation ne peut collecter et traiter **que les données à caractère personnel qui sont nécessaires pour atteindre ces finalités**’*

- Principe « d'exactitude » :

‘L'entreprise/organisation doit s'assurer que les données à caractère personnel sont exactes et tenues à jour au regard des finalités pour lesquelles elles sont traitées, et les corriger le cas échéant’

- Droit de suppression :

*‘Une personne a le droit de demander que les **données à caractère personnel soient effacées** lorsqu'elles ne sont plus nécessaires ou si leur traitement est illicite’*

(*) Source : Commission Européenne