BREVET DE TECHNICIEN SUPÉRIEUR
SERVICES INFORMATIQUES AUX ORGANISATIONS
Option : Solutions logicielles et applications métiers

U6 – CYBERSÉCURITÉ DES SERVICES INFORMATIQUES

SESSION 2023

_____

Durée : 4 heures
Coefficient : 4

_____

Matériel autorisé :
Aucun matériel ni document est autorisé.

Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le sujet comporte 21 pages, numérotées de 1/21 à 21/21.

Cas Lama Zoo

Barème

DOSSIER A	Refonte du système d'habilitation des applications métier	30 points
DOSSIER B	Sécurisation du recueil des avis des participants aux ateliers	20 points
DOSSIER C	Sécurisation des activités liées au parrainage d'animaux	30 points
	TOTAL	80 points

Présentation du contexte

Le **parc Lama Zoo** est un parc zoologique français qui présente environ 35 000 animaux sur 44 hectares. Créé en 1980, il fut un des premiers parcs zoologiques à présenter certains animaux en France, ce qui a contribué à sa notoriété et à son développement. La forme juridique du parc est une société par actions simplifiée (SAS) au capital social de plus de 20 millions d'euros.

Membre permanent de l'association européenne des zoos et aquariums ainsi que de l'association mondiale des zoos et aquariums, il s'engage également dans la conservation en participant à des programmes européens pour les espèces menacées. Il soutient également des associations de conservation œuvrant sur le terrain à travers son association dédiée, Retour à la Nature, et a déjà réintroduit plusieurs animaux en Afrique.

Pour soutenir ces actions, le parc récolte des fonds grâce à des dons et à des parrainages d'animaux.

Le parc zoologique propose une offre hôtelière conséquente avec trois structures d'hébergement : une résidence hôtelière et deux hôtels trois étoiles. L'ensemble de ces structures emploie 40 salariés permanents et près de 180 saisonniers. Le parc compte par ailleurs 20 points de restauration.

Au sein du zoo, le parc organise des ateliers pédagogiques d'observation des animaux ouverts à tout type de public, uniquement sur inscription.

Parallèlement à l'extension des activités du parc, le système d'information (SI) et l'infrastructure sous-jacente se sont étendus ces dernières années.

Parmi les événements marquants, le parc a été victime d'une cyberattaque par rançongiciel (*ransomware*) en 2019 qui a paralysé le SI de l'entreprise pendant deux jours. Le service informatique a réagi en modernisant l'infrastructure du SI et en prenant en compte la cybersécurité plus en amont dans le développement des applications métier. Ce service a en charge le développement et l'exploitation de certaines applications du SI comme la boutique en ligne, le site *Web* du parc et le système de réservation des ateliers. D'autres applications, notamment la billetterie, sont en revanche réalisées par différents prestataires.

L'entreprise souhaite mener davantage de contrôles sur les différents éléments de son système informatique, hébergé autant que possible en interne sur des serveurs virtualisés. Les chantiers de modernisation actuels concernent la refonte du système d'habilitation des applications métier et la sécurisation des processus métier Ateliers et Parrainage.

Vous participez aux chantiers en cours au sein de la direction des systèmes d'information (DSI) sous la responsabilité de M. Brasson, le responsable de la sécurité du système d'information et Mme Delperouse, votre cheffe de projet.

Vous vous appuierez sur le dossier documentaire mis à votre disposition.

Dossier A : Refonte du système d'habilitation des applications métier

Le service des réservations du parc zoologique gère les ateliers pédagogiques proposés aux visiteurs. Les collaborateurs de ce service disposent pour cela d'une application de gestion des ateliers accessible uniquement depuis les postes du service, postes sur lesquels ils s'authentifient avec un identifiant (mél du collaborateur) et un mot de passe.

Le compte d'un collaborateur (M. Breto) a récemment subi une attaque par force brute qui a heureusement été détectée immédiatement par le système d'analyse des traces (*logs*). Le technicien en charge a été alerté et le compte a pu être bloqué dans les 5 minutes après la détection de l'incident, qui n'a pas produit de dégâts. M. Brasson, nouveau responsable de la sécurité du système d'information (RSSI) souhaite éviter que ce type d'incident se reproduise et vous a demandé une analyse des conditions de cette attaque.

Un examen du poste de travail concerné dans les locaux du service des réservations a mis en évidence des pratiques des collaborateurs du service contrevenant à la charte informatique signée par les employés :

- un mot de passe «***1234Travail*** » noté au dos du clavier de M. Breto ;
- un autre poste du service resté allumé avec une session ouverte sans personne devant.

À la suite de ces constatations, le RSSI envisage d'organiser, pour les employés, une sensibilisation à la sécurité afin de leur faire comprendre les enjeux associés aux mots de passe sécurisés et aux fermetures de session.

Question A.1 :

a) Expliquer en quoi consiste une attaque par force brute et pourquoi elle a été efficace pour compromettre le mot de passe de M. Breto.
b) Citer au moins deux moyens de sensibiliser les employés aux enjeux de sécurité.

À la suite de cet incident, le RSSI a fait le point sur l'état actuel des connexions aux postes de travail ayant accès à l'application de gestion des ateliers.
Il apparait qu'un utilisateur, qui se connecte sur n'importe lequel des postes du service, peut ensuite accéder librement à l'application de gestion des ateliers installée en local. Le RSSI considère que ce choix de gestion entraîne des risques de sécurité liés à la traçabilité et à la confidentialité.

Question A.2 :

a) Indiquer en quoi ne pas avoir d'authentification sur l'application constitue un risque de non-traçabilité.
b) Indiquer en quoi ne pas avoir d'authentification sur l'application constitue un risque de perte de confidentialité.

À la demande du RSSI et pour répondre plus globalement à la problématique de l'authentification sur les différentes applications que les collaborateurs du parc zoologique sont amenés à utiliser, le service informatique a commencé le développement d'un système d'authentification et de gestion des habilitations centralisé.

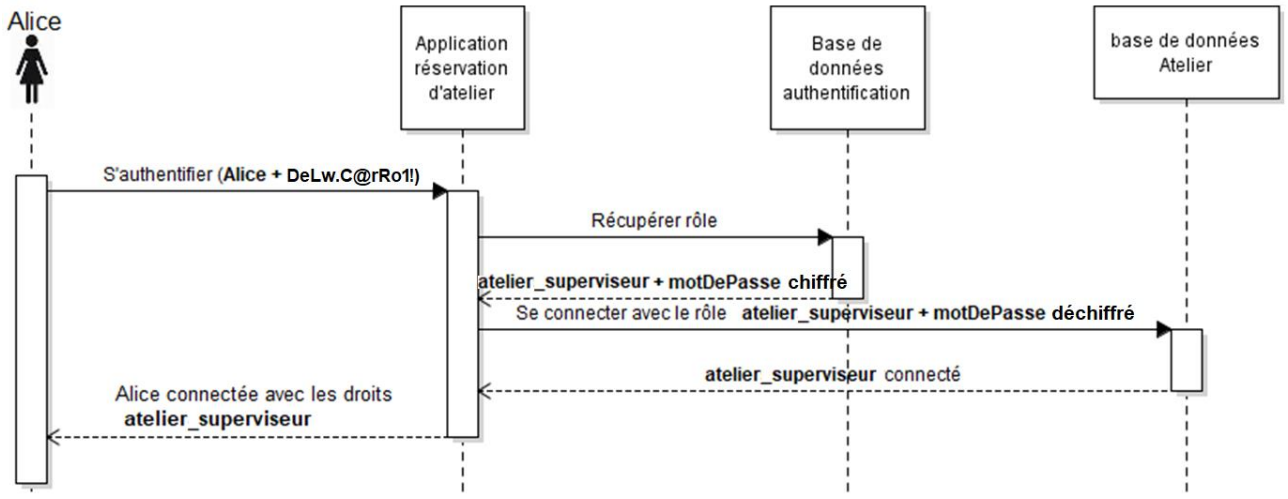
Ce système prendra appui sur plusieurs bases de données :

- une base de données *BdAuthentification* qui permet la gestion des utilisateurs de l'ensemble des applications du SI ;
- des bases de données métier contenant les données propres à chaque application ; par exemple, *BdAtelier* pour la gestion des ateliers.

La base de données *BdAuthentification* permet d'attribuer à chaque membre du personnel, un rôle pour chacune des applications qu'il est amené à utiliser. Ces rôles applicatifs vont correspondre à des comptes utilisateur du système de gestion de bases de données (SGBD), comptes ayant des droits particuliers sur les différentes tables et vues du SGBD.

Lors de la connexion d'un personnel à une des applications du SI, celle-ci va vérifier dans la base *BdAuthentification* si cet utilisateur existe et possède un rôle sur cette application.
Si c'est le cas, l'application va alors stocker en variable de session, les informations de ce rôle applicatif. Ces informations seront ensuite utilisées dans le code de l'application :

- l'identifiant du rôle permettra de gérer au niveau des contrôleurs l'habilitation de l'utilisateur pour accéder aux fonctionnalités appelées ;
- le couple de données identifiant du rôle et mot de passe du rôle seront utilisés par le pilote (*driver*) de connexion à la base de données métier de l'application.



Question A.3 :

Expliquer en quoi l'utilisation de différents rôles applicatifs au niveau du pilote de connexion à la base de données dans l'application permet une meilleure sécurité des données et des traitements dans l'application.

La gestion des habilitations se fera via une application de gestion des utilisateurs qui permettra de leur affecter des rôles applicatifs et de pouvoir les modifier et les supprimer.

Lors de l'écriture des spécifications fonctionnelles du système de gestion des habilitations, il a été identifié que chaque employé possède un statut (superviseur, coordinateur, etc.) et travaille dans un ou plusieurs services du parc zoologique. Les employés sont amenés à changer de service et de statut au fil de leur carrière.

En fonction de ses responsabilités, chaque employé disposera d'un rôle unique sur chaque application qu'il utilise. En cas de changement de responsabilité, l'application de gestion des habilitations permettra d'attribuer à l'employé un nouveau rôle et de modifier voire de supprimer l'ancien rôle.

Par exemple, Mme Lucie Pinaud dont le poste a évolué le 1er septembre 2022 : elle a arrêté de travailler dans le service des ateliers, a changé de rôle dans le service de l'hôtellerie et a intégré le service des animaux. Voici l'évolution de ses droits :

Applications	Rôles jusqu'au 31/08/2022	Rôles au 01/09/2022
Gestion des ateliers	atelier_coordinateur	*Rôle supprimé*
Gestion des hébergements	hotellerie_coordinateur	hotellerie_superviseur
Gestion du parc animalier	/	animaux_coordinateur

Par ailleurs, le parc zoologique emploie régulièrement des stagiaires et des saisonniers qui nécessitent d'avoir des accès temporaires avec une expiration automatique.

Par exemple, Mme Angèle MOREAU a effectué un stage l'été dernier du 2 mai au 30 juin 2022 dans le service de l'hôtellerie et le service des animaux. Voici l'évolution de ses droits :

Applications	Rôles du 02/05/2022 au 30/06/2022	Rôles au 01/07/2022
Gestion du parc animalier	animaux_stagiaire	*Rôle supprimé*
Gestion des hébergements	hotellerie_stagiaire	*Rôle supprimé*

Une première version de la base de données *BdAuthentification* a été mise en place et est en cours de test avec un jeu de données contenant les situations de Mmes Pinaud et Moreau. Un collègue attire votre attention sur le fait que la base de données actuellement testée ne respecte pas les spécifications car :

- les habilitations n'ont pas de durée de vie ;
- un même personnel peut avoir plusieurs rôles pour la même application.

Question A.4 :

a) Indiquer pourquoi ces manques sont des sources de vulnérabilité en matière de sécurité.
b) Proposer les modifications de la base de données corrigeant ces problèmes.

Pour des raisons de traçabilité, il est nécessaire de garder l'historique des opérations de suppression et de modification des habilitations. Pour implémenter ce besoin, une table HistoHabilitation a été créée :

HistoHabilitation (dateheure, numMatriculePerso, idAppli, action)
Clé primaire : dateheure, numMatriculePerso, idAppli

Le champ action de cette table contient un message différent selon l'opération historisée :

- En cas de suppression, il contient : « Suppression du rôle [id-role] ».
- En cas de modification : « Modification du rôle [id-ancien-role] à [id-nouveau-role] ».

L'alimentation de cette table se fera automatiquement avec des déclencheurs (*triggers*) qui surveilleront les suppressions et les modifications d'habilitations dans la table EstHabilite. Le déclencheur *after_delete_habilitation* a déjà été implémenté et il reste à écrire le déclencheur *after_update_habilitation*.

Question A.5 :

Écrire le code du déclencheur *after_update_habilitation* permettant l'historisation des modifications d'habilitations dans la table EstHabilite.

L'application de gestion des habilitations en cours de développement doit permettre de créer et de supprimer les rôles applicatifs. Chaque rôle correspondant à un compte utilisateur du SGBD, il faut donc réaliser deux opérations indissociables en base de données :

- une opération concernant le compte utilisateur du SGBD ;
- une opération concernant l'enregistrement dans la table RoleApplicatif de la base *BdAuthentification*.

La fonction *createRole()* qui permet d'effectuer ces opérations lors de la création d'un nouveau rôle a déjà été codée. Il reste à coder la fonction *deleteRole()* qui permet d'effectuer ces opérations lors de la suppression d'un rôle existant n'ayant plus d'habilitations dépendantes (vérification effectuée en amont de l'appel de la fonction).
Ces deux fonctions utilisent un mécanisme de transaction pour exécuter les requêtes en toute sécurité.

Question A.6 :

a) Expliquer l'intérêt d'utiliser des transactions pour réaliser ces opérations.
b) Compléter le code de la fonction *deleteRole().*

Dossier B : Sécurisation du recueil des avis des participants aux ateliers

Dans le cadre des ateliers, on souhaite pouvoir recueillir les avis des participants pour les rendre visibles sur le site *Web*. Mme Delperouse, votre cheffe de projet, vous demande de participer au développement de la version bêta du recueil des avis sur le site. Cette application, développée avec le cadre applicatif (*framework*) *CodeIgniter*, est expérimentale et son adoption finale sera discutée ultérieurement.

Sur la page présentant une séance passée, on peut lire les avis postés par les participants. Lors des tests orientés sécurité menés à la fin de la dernière itération de code, un testeur attire votre attention sur le risque d'attaque de type XSS (*cross-site scripting*) sur les formulaires de saisie des avis, plus précisément, dans le champ de saisie du texte de l'avis.

Pour démontrer la présence de la faille, le testeur poste un avis contenant du code *JavaScript*.	Le testeur vous indique ensuite que le code *JavaScript* est exécuté à chaque affichage de son « faux commentaire ».
	
http://lamazoo.com/ateliers/commentform/5	http://lamazoo.com/ateliers/comments/5

Le fichier de configuration des routes de *CodeIgniter* permet de recenser toutes les routes de l'application et leur traitement. La classe Filters permet de définir les filtres actifs dans l'application. Un filtre actif permet d'exécuter des actions avant ou après l'exécution des contrôleurs.

Question B.1 :

a) Expliquer le rôle de la méthode *before* de la classe AuthGuard.
b) Expliquer si un attaquant a besoin d'être préalablement authentifié pour insérer un commentaire dans le formulaire de saisie des avis.

On distingue parmi les différentes sortes d'attaques XSS celles dont la charge utile (code malveillant envoyé) est stockée en base de données (attaque XSS stockée) et celles dont la charge utile est transmise via une adresse réticulaire (*URL*) et exécutée par le navigateur (attaque XSS reflétée).

Question B.2 :

a) Expliquer pourquoi il s'agit ici d'une attaque XSS stockée.
b) Expliquer pourquoi une attaque de type XSS stocké est susceptible de toucher un plus grand nombre de visiteurs du site *Web* qu'une attaque XSS reflétée.

Le scénario le plus redouté serait un vol de témoins de connexion (*cookies*) entrainant l'usurpation de comptes. Il est donc indispensable de corriger le code afin d'empêcher tout type d'attaque XSS.
Afin de sécuriser complètement l'application, la bonne pratique est d'échapper les données à la fois lors de leur affichage sur la page des commentaires et lors de leur enregistrement dans la base de données.

Question B.3 :

a) Donner au moins un argument lié à la sécurité expliquant pourquoi il est conseillé de filtrer les données issues de la base de données avant leur affichage sur le site.
b) Donner au moins un argument lié à la sécurité expliquant pourquoi il est conseillé de filtrer les données issues du formulaire avant leur écriture en base de données.

Afin de prévenir les attaques XSS, *CodeIgniter* fournit une méthode *esc()* qui permet d'échapper les données lors des traitements.
Mme Delperouse vous demande de sécuriser le processus de saisie des commentaires en respectant les bonnes pratiques et en utilisant les outils fournis par *CodeIgniter*.

Question B.4 :

a) Modifier la méthode *commentStore* de la classe AteliersController pour neutraliser le code *JavaScript* lors de l'enregistrement dans la base de données. (Ne recopier que les lignes modifiées en précisant leur numéro.)
b) Modifier le code de la vue *coms.php* afin d'empêcher l'exécution du code *JavaScript* lors de l'affichage des données. (Ne recopier que les lignes modifiées en précisant leur numéro.)

Dossier C : Sécurisation des activités liées au parrainage d'animaux

Note : Ce dossier utilise les documents C1 à C6 et les documents A4, B3 et B4.

Le site du parc zoologique dispose d'une fonctionnalité permettant de parrainer un animal. Le visiteur du site peut choisir l'animal qu'il souhaite parrainer et pour quel montant. Il obtient ensuite régulièrement, par différents canaux, des nouvelles de l'animal qu'il parraine.
Cette fonctionnalité, en raison de sa popularité, constitue une source de revenus non négligeable pour soutenir les activités du parc. Une adresse de courriel **communication@lamazoo.com** a donc été créée pour communiquer régulièrement des informations aux parrains. Par ailleurs, le serveur **communication.lamazoo.com** permet d'héberger les pages d'information destinées aux parrains.

Récemment, des parrains se sont plaints de courriels suspects. Il semble qu'une campagne d'hameçonnage (*phishing*) soit à l'œuvre. Un parrain, M. Edwin Hardi qui semble avoir perdu l'accès à son compte en cliquant sur un lien dans un de ces messages, a transmis une copie du courriel frauduleux par la page contact du site. Mme Delperouze vous a remis ce courriel pour l'étudier et confirmer sa nature d'hameçonnage.

Question C.1 :

Observer le courriel reçu par M. Hardi et citer cinq types d'indices qui laissent penser qu'il s'agit bien d'un hameçonnage.

M. Edwin et la société Lama Zoo souhaitent porter plainte pour mettre un terme à la campagne d'hameçonnage.

Question C.2 :

Citer deux infractions qui peuvent être retenues contre la personne malveillante.

L'équipe informatique du parc zoologique souhaite répondre à cette menace.
Mme Delperouze vous adresse le message suivant :

« Il semble que nous soyons victimes d'une attaque de type CSRF *(cross site request forgery), selon le scénario suivant : un parrain reçoit un message contenant de fausses nouvelles de ses parrainages et clique sur un lien contenu dans celui-ci alors qu'une session est ouverte dans son navigateur sur notre site. Le clic sur ce lien entraîne la suppression de son compte.*
Notre site a été développé à partir du framework PHP CodeIgniter et je sais que celui-ci intègre directement des mécanismes de protection contre les attaques CSRF*. Je vous transfère la documentation technique en pièce jointe. Merci de nous proposer une solution simple à mettre en œuvre, qui nous assure de ne plus être vulnérables face à ce type d'attaque à l'avenir. »*

Question C.3 :

a) Indiquer pour chacune des deux solutions de protection contre les attaques CSRF proposées par l'environnement de développement *CodeIgniter* les éléments de sécurisation stockés côté client et ceux stockés côté serveur.
b) Indiquer les modifications à apporter aux classes Security et Filters afin d'activer la protection CSRF basée sur la session (*Synchronizer token*) et préciser ce que les développeurs des vues devront utiliser dans les formulaires qu'ils veulent protéger.

Mme Nadia Erben a développé une interface de programmation d'application (*API*) permettant d'accéder en lecture aux bases de données des applications du parc.

Depuis que vous avez mis en place, avec succès, la protection contre les attaques CSRF, les requêtes de l'interface *API* ne renvoient plus la réponse attendue. Mme Erben indique que les données qu'elle met à disposition sont publiques et en lecture seule. Elle indique également qu'il est impossible de faire un scénario d'attaque CSRF qui ait un sens en utilisant les routes qui concernent les traitements de l'interface *API* (routes qui commencent par '/ws/…').
Elle souhaite que la protection contre les attaques CSRF ne s'applique pas à ces routes. La documentation de l'environnement de développement *CodeIgniter* suggère l'utilisation d'une liste blanche (*whitelist*) pour permettre à certaines routes d'échapper au filtre CSRF.

Question C.4 :

a) Expliquer pourquoi les routes de l'interface *API* ne répondent plus depuis la mise en place de la protection contre les attaques CSRF.
b) Indiquer les modifications à apporter à la classe Filters afin de mettre en place une liste blanche en vous aidant du fichier de configuration des routes.

Le site *Web* permet actuellement d'afficher pour les parrains des fiches d'informations générales sur les animaux parrainés (photo, nom, espèce, etc.).
Actuellement, le site *Web* se connecte à la base *BdAnimaux* grâce au compte utilisateur *site_public* avec des droits d'accès sur la base *BdAnimaux*. Cette base de données contient cependant certaines informations qui doivent rester protégées afin de ne pas faciliter le vol éventuel d'animal ou la dégradation de leur environnement de vie par exemple. Afin d'éviter tout risque de fuite de données ou de destruction de données, il a été décidé de ne plus accéder aux informations des animaux en interrogeant directement les tables mais par le biais d'une vue SQL.
Mme Delperouze vous demande de retirer les droits actuels attribués au compte utilisateur *site_public*, de créer la vue SQL *vueAnimal* en vous appuyant sur l'exemple de fiche d'un animal fournie dans la documentation, et d'attribuer au compte utilisateur *site_public* un droit en lecture seule sur cette vue.

Question C.5 :

Écrire toutes les requêtes à effectuer sur la base de données afin de prendre en compte la demande de Mme Delperouze.

Dans la prochaine version de son site, le parc zoologique souhaite mettre à l'honneur les dix principaux parrains, en affichant un tableau classé.
Lors des tests d'acceptation, plusieurs testeurs mettent en garde contre la divulgation de données personnelles. Il serait souhaitable de modifier le tableau affiché afin de ne plus divulguer de données personnelles mais que chaque parrain puisse quand même se reconnaître (pseudonymisation).

Question C.6 :

Proposer une nouvelle version du tableau d'honneur.

Dossier documentaire

Document A1 : Extraits de la base de données BdAuthentification

Extrait du schéma relationnel :

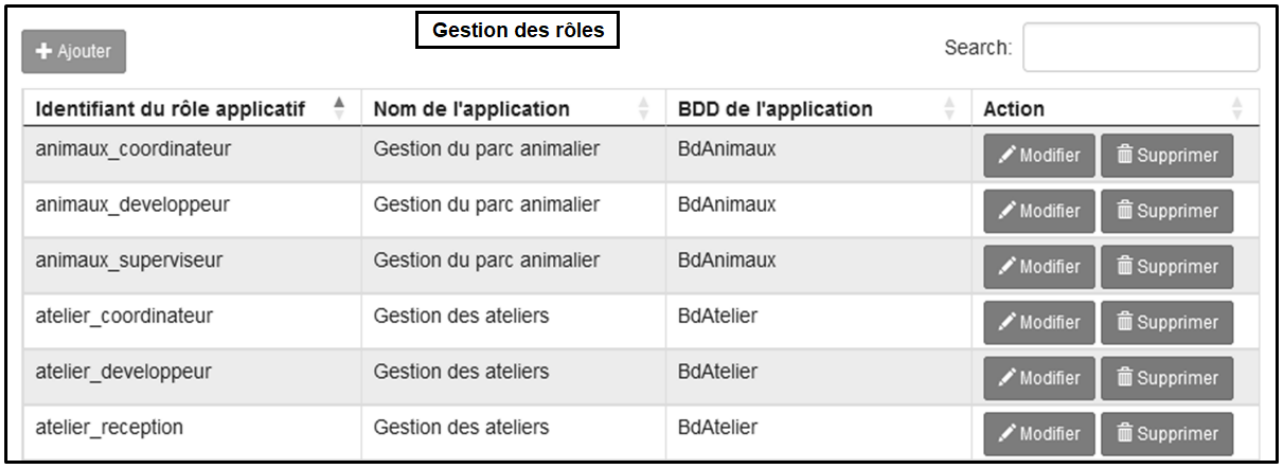
Les tables **Service** et **Application** ne sont pas présentées dans cet extrait.

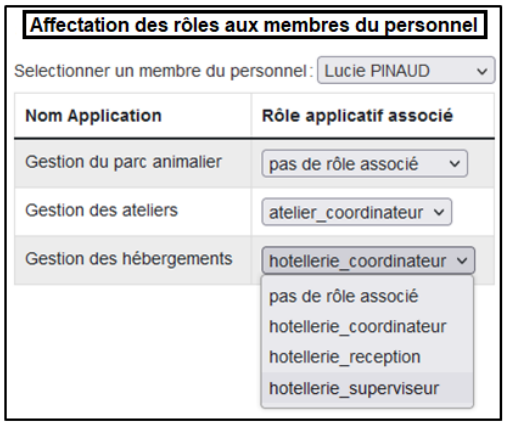
Personnel (numMatriculePerso, melPerso, mdpPerso, nomPerso, prenomPerso, dateNaissancePerso, adressePerso, telPerso, numService)
- Clé primaire : numMatriculePerso
- Clé étrangère : numService en référence à numService de Service

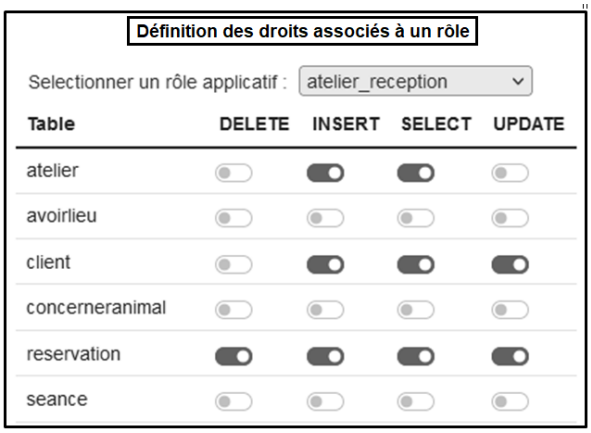
RoleApplicatif (idAppli, idRoleAppli, mdpRoleAppli)
- Clé primaire : idAppli, idRoleAppli
- Clé étrangère : idAppli en référence à idAppli de Application

EstHabilite (numMatriculePerso, idAppli, idRoleAppli)
- Clé primaire : numMatriculePerso, idAppli, idRoleAppli
- Clé étrangère : numMatriculePerso en référence à numMatriculePerso de Personnel
- Clé étrangère : idAppli, idRoleAppli en référence à idAppli, idRoleAppli de RoleApplicatif

Document A2 : Maquettes de l'application de gestion des habilitations







Document A3 : Code du déclencheur after_delete_habilitation

```
CREATE TRIGGER `after_delete_habilitation`
AFTER DELETE ON `EstHabilite`
FOR EACH ROW
BEGIN
   INSERT INTO HistoHabilitation
      VALUES (NOW(), old.numMatriculePerso, old.idAppli, CONCAT("suppression du rôle ", old.idRoleAppli))
END
```

Dans un déclencheur MySql, l'élément **old** contient les valeurs des colonnes de la ligne traitée avant modification par l'évènement déclencheur, l'élément **new** contient les valeurs après cet évènement.

Document A4 : Extraits de la documentation MySQL

NOW() : renvoie la date et l'heure du système

CONCAT(colonne1, colonne2) : renvoie la concaténer les valeurs de plusieurs colonnes ou expressions pour ne former qu'une seule chaîne de caractères.

YEAR(date) : renvoie l'année de la date passée en paramètre.

```
SELECT YEAR("2017-06-15 09:34:21"); -- renvoie 2017
```

CREATE USER : permet la création d'un nouveau compte utilisateur MySQL.

```
CREATE USER 'MorineauJ' IDENTIFIED BY 'unMotDePasse';  -- création d'un utilisateur 'MorineauJ' qui se connecte avec le mot de passe 'unMotDePasse'.
```

DROP USER : permet la suppression d'un compte utilisateur MySQL.

```
DROP USER 'MorineauJ';  -- supprime l'utilisateur 'MorineauJ'.
```

GRANT : permet l'attribution de droit à un compte utilisateur.

```
GRANT priv_type ON [object_type] priv_level TO user
```

REVOKE : permet la suppression de droit sur un compte utilisateur.

```
REVOKE priv_type ON [object_type] priv_level FROM user
```

CREATE VIEW : permet la création de vue.

```
CREATE VIEW view_name [(column_list)] AS select_statement [WITH CHECK OPTION]
```

```
CREATE VIEW test.uneVue AS SELECT * FROM uneTable; -- crée la vue uneVue affichant les informations de la table uneTable
```

Dans les instructions GRANT et REVOKE :

- *priv_type* contient les privilèges attribués. Les valeurs autorisées sont : ALTER, CREATE VIEW, CREATE, DELETE, DROP, GRANT OPTION, INDEX, INSERT, SELECT, SHOW VIEW, TRIGGER et UPDATE et ALL
- *object_type* contient le nom de la base de données
- *priv_level* contient le nom de l'objet (table, vue, colonne, …)
- *user* contient le nom du compte.

Document A5 : Fonctions PHP de création et de suppression d'un rôle applicatif

Lors de la création d'un rôle applicatif, le mot de passe enregistré dans la table RoleApplicatif est chiffré de façon réversible par l'application car le pilote (*driver*) de connexion à la base métier nécessite le mot de passe en clair. Il sera déchiffré par l'application, uniquement lors de sa récupération au moment d'établir la connexion à l'application métier concernée.

```
1.  public function createRole(string $id, string $mdp, string $idAppli) : string
2.  {
3.     $message = "";
4.     $encryptMdp = openssl_encrypt($mdp, $config['encryption_algo'] ,
    $config['encryption_key']) ; // Chiffre le mot de passe avec l'algorithme et la clé
    fournis dans le tableau de variables globales $config
5.
6.     try {
7.        this->monPdo->beginTransaction(); // début de la transaction
8.           $req = $this->monPdo->prepare('CREATE USER :idUser IDENTIFIED BY :mdpUser ;');
9.           $req->bindParam(':idUser', $id, PDO::PARAM_STR);
10.          $req->bindParam(':mdpUser', $mdp, PDO::PARAM_STR);
11.          $resultat = $req->execute();
12.
13.          $req = $this->monPdo->prepare('INSERT INTO RoleApplicatif (idAppli, idRoleAppli,
    mdpRoleAppli) VALUES (:idAppli, :idRole, :mdpRole) ;');
14.          $req->bindParam(':idAppli', $idAppli, PDO::PARAM_STR);
15.          $req->bindParam(':idRole', $id, PDO::PARAM_STR);
16.          $req->bindParam(':mdpRole', $encryptMdp, PDO::PARAM_STR);
17.          $resultat = $resultat + $req->execute();
18.
19.       $this->monPdo->commit(); // fin de la transaction par un commit
20.
21.       if ($resultat) { $message = "ok"; }
22.    }
23.    catch (PDOException $e) {
24.       $this->monPdo->rollback(); // annulation de la transaction
25.       $message = "Erreur !: " .$e->getMessage();
26.    }
27.    return $message;
28. }
```

```
1.  public function deleteRole(string $id, string $idAppli) : string
2.  {
3.     $message = "";
4.     try {
5.        $this->monPdo->beginTransaction(); // début de la transaction
6.
7.        /*     Code à compléter sur votre copie     */
8.
9.        $this->monPdo->commit(); // fin de la transaction par un commit
10.
11.       if ($resultat) { $message = "ok"; }
12.    }
13.    catch (PDOException $e) {
14.       $this->monPdo->rollback(); // annulation de la transaction
15.       $message = "Erreur !: " .$e->getMessage();
16.    }
17.    return $message;
18. }
```

Document B1 : Code du contrôleur AteliersController.php

```
1.     class AteliersController extends BaseController
2.     {
3.         public function index()
4.         {}
5.         public function comments(int $numAtelier)
6.         {}
7.         public function commentForm(int $numAtelier)
8.         {}
9.
10.        public function commentStore(){
11.            $session = session();
12.            helper(['form']); /**chargement de la bibliothèque form_helper**/
13.
14.            if ($session->get('isLoggedIn')){
15.                $numCli=$session->get('id');
16.                $numSeance=$this->request->getVar('numseance');
17.                $rules = [
18.                   'comment'   => 'required|min_length[2]',
19.                   'numseance' => 'required',
20.                   'note'      => 'required'];
21.                $evalExiste=$evaluerModel->evaluerExiste($numSeance,$numCli);
22.                if($this->validate($rules) && !$evalExiste ) {
23.                     /** on crée le commentaire à partir des valeurs saisies**/
24.                    $data = [
25.                        'numCli' => $numCli,
26.                        'numSeance' => $numSeance,
27.                        'commentaireEvaluer' => $this->request->getVar('comment'),
28.                        'noteEvaluer' => $this->request->getVar('note')
29.                    ];
30.                    $evaluerModel=new EvaluerModel();
31.
32.                    /** Inscription du commentaire en base de données**/
33.                    $evaluerModel->save($data);
34.                    return redirect()->to('/ateliers');
35.                }else{
36.                   /** Affichage du formulaire de saisie de commentaires avec les erreurs**/
37.                    $atelierModel=new AtelierModel();
38.                    $data['title']="Ajout d'un commentaire";
39.                    // on récupère la séance concernée par le commentaire
40.                    $seanceModel=new SeanceModel();
41.                    $seance=$seanceModel->getSeanceParNum($numSeance);
42.                    //on récupère le num d'atelier
43.                    $numAtelier=$seance->numAtelier;
44.                    $data['validation']=$this->validator;
45.                    /** tous les numéros des séances concernant cet atelier et
46.                     cet utilisateur**/
47.                    $data['numseances']=$atelierModel->getNumSeances($numCli,$numAtelier);
48.                    /** indiquer si refusé parce qu'un commentaire existe déjà**/
49.                    $data['evalExiste']=$evalExiste;
50.                    echo view('atelier/header.php', $data);
51.                    echo view('atelier/comform.php', $data);
52.                    echo view('atelier/footer.php', $data);
53.                }
54.            }
55.        }
56.    }
```

Document B2 : Code de la vue coms.php

```
1.     <?php
2.     echo "<br/>";
3.     foreach ($coms as $com){
4.            echo $com."<br/>";
5.     }
6.     /** $vecu contient true si l'utilisateur connecté a participé à cet atelier **/
7.     if($vecu){
8.         echo('<a href="'.base_url().'/ateliers/commentform/'.$numAtelier.'">Ajouter un
commentaire</a>');
9.     }
10.    ?>
```

Document B3 : CodeIgniter - Fichier de configuration des routes

Les routes nécessaires à la réalisation du sujet sont indiquées en caractères gras.

```
1.  //*
2.  Route Definitions
3.  */
4.  // We get a performance increase by specifying the default
5.  // route since we don't have to scan directories.
6.  $routes->get('/ateliers', 'AteliersController::index');
7.  $routes->get('/ateliers/comments/(:num)', 'AteliersController::comments/$1');
8.  $routes->get('/ateliers/commentform/(:num)', 'AteliersController::commentForm/$1',['filter'
    => 'authGuard']);
9.  $routes->post('/ateliers/commentstore', 'AteliersController::commentStore',['filter' =>
    'authGuard']);
10. $routes->get('/', 'AccueilController::index');
11. $routes->get('/inscription', 'InscriptionController::index');
12. $routes->match(['get', 'post'], 'inscrire', 'InscriptionController::store');
13. $routes->match(['get', 'post'], 'connecter', 'ConnexionController::loginAuth');
14. $routes->get('/connexion', 'ConnexionController::index');
15. $routes->get('/profil', 'ProfilController::index',['filter' => 'authGuard']);
16. $routes->post('/profil/supprcmptconfirm', 'ProfilController::supprcmpt',['filter' =>
    'authGuard']);
17. $routes->get('/parrainage', 'ParrainageController::index');
18. $routes->get('/parrainage/parrainer/(:num)',
    'ParrainageController::parrainerForm/$1',['filter' => 'authGuard']);
19. $routes->post('/parrainage/parrainagestore',
    'ParrainageController::parrainageStore',['filter' => 'authGuard']);
20. $routes->get('/parrainage/tableau', 'ParrainageController::tableau');
21. $routes->get('/disconnect', 'DisconnectController::index',['filter' => 'authGuard']);
22. $routes->match(['get', 'post'],'/ws/animaux', 'WSAnimauxController::getAnimaux');
23. $routes->match(['get', 'post'],'/ws/hebergements',
    'WSHebergementsController::getHebergement');
24. $routes->match(['get', 'post'],'/ws/ateliers', 'WSAtelierController::getAteliers');
```

Document B4 : CodeIgniter - Extraits des classes Security et Filters

```
1.   class Security extends BaseConfig
2.   {
3.       public $csrfProtection = 'cookie';
4.       public $tokenRandomize = false;
5.       public $tokenName = 'csrf_test_name';
6.       public $headerName = 'X-CSRF-TOKEN';
7.   ...
```

```
1.    class Filters extends BaseConfig
2.    {
3.        /** Configure des alias pour les filtres */
4.        public $aliases = [
5.            'csrf'          => CSRF::class,
6.            'toolbar'       => DebugToolbar::class,
7.            'honeypot'      => Honeypot::class,
8.            'invalidchars'  => InvalidChars::class,
9.            'secureheaders' => SecureHeaders::class,
10.           'authGuard' => \App\Filters\AuthGuard::class,
11.       ];
12.
13.       /** Liste d'alias de filtres exécutés avant et après chaque requête */
14.       public $globals = [
15.           'before' => [ 'honeypot', 'invalidchars', ],
16.           'after' => [ 'toolbar', ],
17.       ];
18.   ...
```

Document B5 : CodeIgniter - Classe de filtres AuthGuard

```
1.  class AuthGuard implements FilterInterface
2.  {
3.    public function before(RequestInterface $request, $arguments = null)
4.    {
5.           if (!session()->get('isLoggedIn'))
6.           {
7.                  return redirect()->to('/connexion');
8.           }
9.    }
10. ...
```

Document B6 : CodeIgniter - Documentation sur la méthode esc()

esc(*$data*[, *$context = 'html'*[, *$encoding*]])

Paramètres :

- **$data** (*string* | *array*) – les données à échapper.
- **$context** (*string*) – le contexte d'échappement (valeur 'html' par défaut).
- **$encoding** (*string*) – l'encodage utilisé lors de l'échappement.

Retour : Les données du paramètre $data échappées selon le contexte fourni.
Par exemple, **esc('
','html')** *retournera* "**
**"

Échappe les données pour les inclure dans les pages *Web*, afin d'aider à prévenir les attaques XSS. Utilise la bibliothèque *Laminas Escaper* pour gérer le filtrage réel des données. Si $data est une chaîne, elle est échappée simplement et renvoyée. Si $data est un tableau, il est parcouru, et les valeurs des paires clé valeur sont échappées. Les valeurs valides pour $context sont : html, js, css, url, attr, raw.

Document C1 : Courriel d'hameçonnage (phishing) reçu par Edwin Hardi

Sujet :	[LAMAZOO] Offre Spéciale
Date :	22/08/2022
De :	zomagic.yahoo-micro@businessmail.com
Pour :	e.hardi24@orange.fr <e.hardi24@orange.fr>

Bonjour cher Parrain,

Pour vous remercier de votre fidèle soutien envers nos animaux, nous vous offront 4 entrées gratuite accessibles

en cliquant sur le lien suivant cestla fete enfamille

Vous souhaitant bonne réception

Léquipe zoo

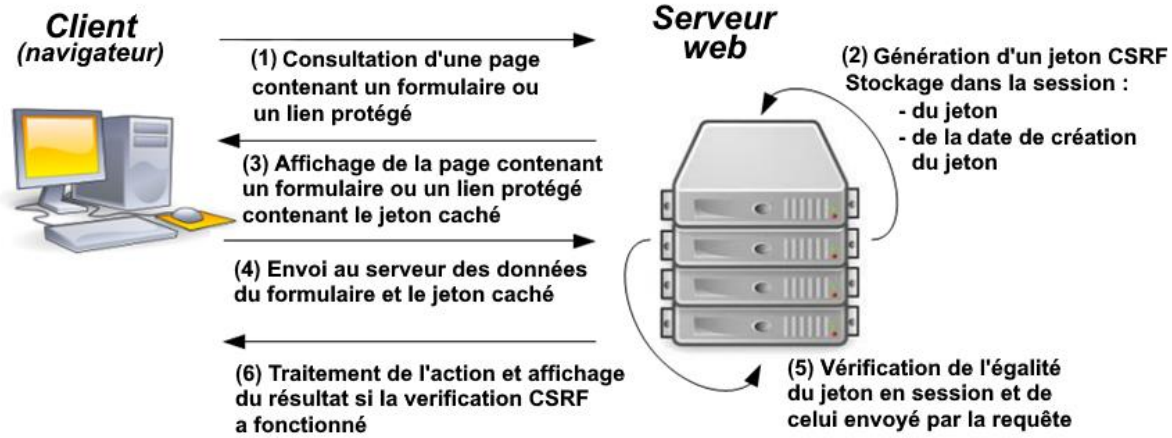
Ouvrir l'hyperlien :
https://lamazoo.com/profil/supprcmptconfirm

Document C2 : Comment se protéger des attaques CSRF

Le projet OWASP (*Open Web Application Security Project*) propose différentes approches pour se prémunir des attaques CSRF qui diffèrent selon que des informations sont ou non stockées côté serveur.

Protection CSRF avec le patron de conception *Synchronizer Token Pattern*

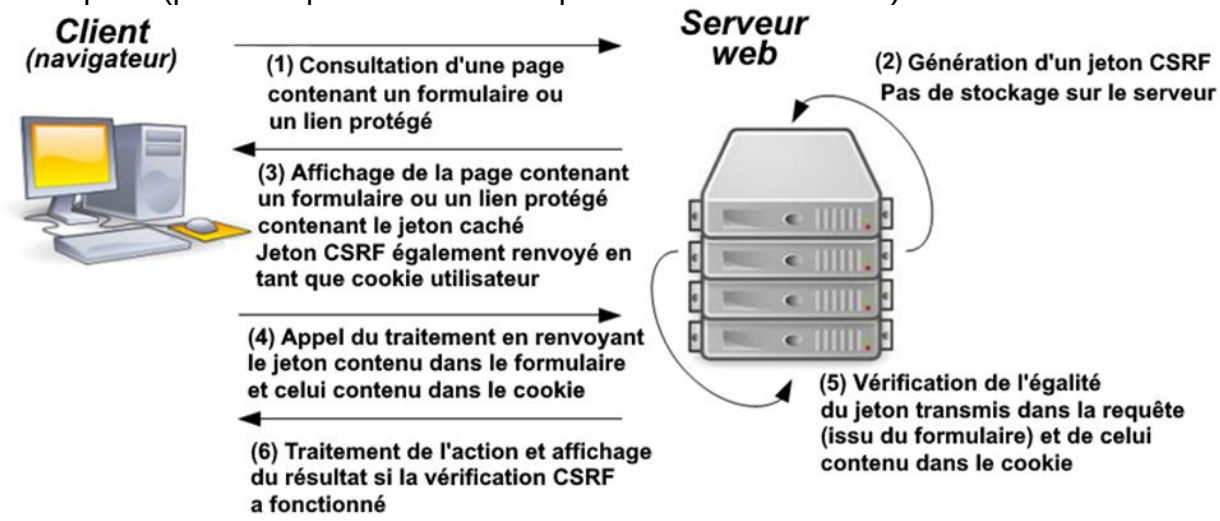
Il s'agit lors d'une session de générer sur le serveur un identifiant d'accès unique, nommé jeton (*token*) CSRF, non prédictible et avec une durée de validité limitée dans le temps. Ce jeton est stocké en variable de session et ajouté dans un champ caché d'un formulaire ou dans un paramètre de l'identifiant *URI* (*uniform resource identifier*) pour chaque action sensible à sécuriser (déconnexion, modification/suppression de données en BDD, etc.).



Lorsque le serveur reçoit une requête, il vérifie que le jeton stocké en variable de session et celui transmis correspondent bien. Si ce n'est pas le cas, la requête est rejetée comme étant une tentative d'attaque CSRF.

Protection CSRF avec le patron de conception *Double Submit Pattern* :

Il s'agit également de faire générer un jeton CSRF par le serveur mais celui-ci n'est pas stocké en variable de session sur le serveur mais envoyé au client lors de la génération de la page HTML sous deux formes distinctes : un témoin de connexion (*cookie)* envoyé de manière habituelle dans les entêtes de la requête HTTP et un jeton envoyé dans le corps de la requête (par exemple dans un champ caché d'un formulaire).



Lorsque le serveur reçoit une requête, il vérifie que les deux jetons transmis correspondent bien. Si les deux informations ne sont pas présentes ou ne correspondent pas, la requête est rejetée comme étant une tentative d'attaque CSRF.

Document C3 : CodeIgniter - Extraits de documentation sur la protection contre les attaques CSRF

<u>Activer la protection contre les attaques CSRF</u>

Pour activer la protection CSRF, il faut activer le filtre '*csrf*' avant l'exécution des requêtes dans le tableau $globals de la classe de configuration Filters (document B4).

Dans l'environnement de développement *CodeIgniter*, par défaut, la protection contre les attaques CSRF basée sur les témoins de connexion est utilisée (patron de conception *double submit cookie*).
Pour activer la protection contre les attaques CSRF basée sur la session (patron de conception *sychronizer token*) il faut attribuer la valeur 'session' à la propriété publique $csrfProtection dans la classe de configuration **Security**.

Certaines routes peuvent être ajoutées à une liste blanche afin qu'elles ne soient pas protégées par la protection contre les attaques CSRF du logiciel *CodeIgniter.* Pour cela, vous pouvez ajouter ces routes en tant qu'exceptions dans le filtre '*csrf*' exécuté avant chaque requête. Il est possible d'utiliser des expressions régulières (*regex*) pour écrire ces exceptions.

Dans l'exemple suivant, la route 'api/record/save' et toutes les routes de la forme 'test/1', 'test/2', etc. et les routes qui commencent par 'dev' échappent au filtre '*csrf*' :

```
'csrf' => ['except' => ['api/record/save', 'test/[0-9]+', 'dev*']]
```

Paramétrer les formulaires HTML avec les fonctions *csrf_token(), csrf_hash() et csrf_field()*

Si vous utilisez la bibliothèque *form_helper*, l'appel à la fonction *csrf_field()* générera dans votre formulaire un champ caché contenant un jeton (*token*) CSRF :

```
      <?= csrf_field() ?>
// Génère:  <input type="hidden" name="{csrf_token}" value="{csrf_hash}" />
```

Si vous n'utilisez pas la bibliothèque *form_helper,* vous pouvez utiliser les fonctions *csrf_token()* et *csrf_hash()* dans un champ caché créé manuellement dans votre formulaire :

```
<input type="hidden" name="<?= csrf_token() ?>" value="<?= csrf_hash() ?>" />
```

Document C4 : Tableau d'honneur des parrains

Tableau d'honneur des parrains

#	Prénom	Nom	Rue	Code Postal	Ville	Téléphone	Total
1	Sylvie	Schmidt	6 rue Chaptal	34000	Montpellier	0123456769	1050.0000
2	Linda	Menin	3 rue des bois	12340	Millau	0123456789	20.0000

@ 2023 Parc Lama Zoo

Document C5 : Description des tables Animal et Espece dans la base de données BdAnimaux et script des droits d'accès pour le site Web

```
Animal(numAnimal, nomAnimal, poidsAnimal, paysOriAnimal, pretAnimal,
dateNaissAnimal, dateDecesAnimal, commentaire, numPereAnimal, numMereAnimal,
numEspece, numZoo, numZone, photoAnimal)
      Clé primaire : numAnimal
      Clés étrangères : numEspece en reference à numEspece de Espece
                        ...
Espece(numEspece, nomCommunEspece, nomScientifiqueEspece, codeType)
      Clé primaire : numEspece
```

Script des droits d'accès à la base *BDAnimaux* pour le site *Web* :

```
CREATE USER 'site_public' WITH PASSWORD 'fdg$?54kH&T*';
GRANT ALL ON BdAnimaux.animal TO 'site_public';
```

Document C6 : Exemple de fiche d'un animal affichée sur le site Web

Nom : Serge
Espèce : Lama
Année de naissance : 2012
Poids : 142 kg

Serge arrive d'un grand zoo du sud de l'Italie où il était le seul de son espèce. Il est d'une nature calme et posée. C'est également un grand gourmand qui croque tout ce qui passe à sa portée.