SIMPLON
.CO

Stratégie de Sécurisation
de l'application pire2pire.com

Réalisé par :

- Yousra Chbib

Encadré par :

- Cyril MARCQ
- Hachemi HARIZI
- Claire VAN WYNENDAELE

Sommaire

I. Introduction

In today's digital age, security is more important than ever. With the increasing number of cyber threats, data breaches, and hacking attempts, ensuring the safety of user information and the integrity of online platforms is a crucial challenge for any digital project. As technology evolves, so do the methods used by cybercriminals, making it essential for organizations to adopt strong security measures.

A single vulnerability in an application can lead to significant damage, not only to the platform itself but also to its users. That's why having a well-thought-out and robust security strategy is essential for protecting both the application and its users.

The goal of this report is to present a security strategy for the "pire2pire.com" learning application. The main idea is to ensure a secure system that protects user data and prevents unauthorized access.
The application is an online platform where different types of users interact: learners, professionals, and administrators. It includes features like course registration, catalog browsing, and training sessions.

To develop this strategy, I followed a structured approach. There was a lot of information available about security, so I had to filter and organize the most relevant points. My approach is based on defense-in-depth and component-based security, applying protection at different levels. A simple architecture diagram will also be included to explain the system better.

II. Coté Client

1. Page de Connexion

1.1 Sécurisation du nom d'utilisateur

La sécurisation du champ « nom d'utilisateur » est une étape clé dans la protection de vos systèmes. Elle doit garantir que les données saisies sont conformes à un format attendu et ne peuvent pas être utilisées pour mener des attaques telles que l'injection de code ou l'énumération de comptes existants. Voici les axes essentiels à mettre en place :

1.1.1 Validation des entrées avec des expressions régulières (Regex):

Les expressions régulières sont des outils puissants permettant de définir des motifs précis pour contrôler la validité des données saisies par l'utilisateur. Elles servent à :

- **Définir un format précis** : Par exemple, vérifier qu'un nom d'utilisateur ne contient que des lettres, chiffres et certains caractères spéciaux autorisés.
- **Valider des formats spécifiques** :
  - Pour le adresse email : `^[a-zA-Z0-9._%+-]+@gmail\.com$`
  - Pour le numéro de téléphone : `^\+?[1-9]\d{1,14}$`

Cela permet de s'assurer que l'email et/ou le numéro de téléphone suivent un schéma standard (bien que dans un cas spécifique, vous puissiez restreindre par exemple aux adresses Gmail).

1.1.2 Filtrage et nettoyage des entrées (Sanitization) :

Même après validation, il est essentiel de nettoyer la saisie afin d'éviter les injections ou d'autres attaques malveillantes :

➢ **Suppression des espaces inutiles** : Pour enlever les espaces en début et fin de chaîne pour éviter des erreurs ou des incohérences dans le stockage.

➢ **Échappement ou refus des caractères dangereux :** I interdire ou échapper des caractères comme « `<, >, ;, --`», ou d'autres symboles susceptibles d'être exploités dans une injection **SQL** ou **XSS**.

1.1.3 Définition des Contraintes de longueur minimale et maximale :

La définition des contraintes de longueur vise à garantir un équilibre entre accessibilité et sécurité. Une exigence de longueur minimale de 6 caractères permet d'éviter les identifiants trop courts, souvent vulnérables aux attaques par dictionnaire. À l'inverse, fixer une longueur maximale,de 50 caractères, prévient les abus et réduit les risques liés aux dépassements de mémoire. Ces contraintes doivent être cohérentes avec la politique de sécurité globale de l'application et s'aligner avec les autres critères de validation et exigences métier.

1.2 La Sécurité des Mots de Passe

La gestion des mots de passe est un élément essentiel de la sécurité informatique. Cette section présente les mesures de sécurité pour assurer une protection efficace contre les attaques courantes, en mettant l'accent sur la longueur, la complexité et la gestion des mots de passe.

1.2.1 Longueur Minimale :

Un mot de passe sécurisé doit comporter au moins **12 caractères** pour résister aux attaques par **force brute**. Les mots de passe courts sont rapidement cassés, notamment par des attaques par dictionnaire et des méthodes de **calcul parallélisées**.

1.2.2 Complexité :

Pour renforcer la sécurité des comptes, il est essentiel que les mots de passe respectent des critères de complexité spécifiques. Lorsqu'un utilisateur tente de définir un mot de passe ne répondant pas à ces exigences, des messages d'avertissement clairs doivent être affichés pour guider l'utilisateur vers la création d'un mot de passe conforme.

Exemple :
L'utilisateur saisit un mot de passe composé d'un seul chiffre.
Messages d'avertissement affichés :

- **Longueur insuffisante**
- **Absence de majuscules**
- **Absence de minuscules**
- **Absence de caractères spéciaux**

Ces messages guident l'utilisateur pour qu'il crée un mot de passe conforme aux critères de sécurité recommandés, renforçant ainsi la protection de son compte.

1.2.3 Interdiction des Informations Personnelles :

Pour renforcer la sécurité des mots de passe, il est essentiel d'éviter l'utilisation d'éléments facilement devinables tels que les noms, prénoms, dates de naissance, noms d'utilisateur ou des mots courants comme "password123". Dans ce cadre, des **vérifications** seront mises en place côté serveur pour comparer les mots de passe saisis avec une liste de mots interdits. Cette approche garantit que les utilisateurs ne peuvent pas définir de mots de passe contenant des informations personnelles ou des termes trop simples, renforçant ainsi la robustesse globale des mots de passe utilisés.

1.2.4 Vérification des Fuites de Données :

Pour renforcer la sécurité des mots de passe, il est essentiel de vérifier si ceux-ci ont été compromis lors de fuites de données précédentes. On a mis en place des vérifications côté serveur qui comparent les mots de passe saisis avec des bases de données de mots compromis, telles que **Have I Been Pwned**.

Exemple d'intégration en Python :

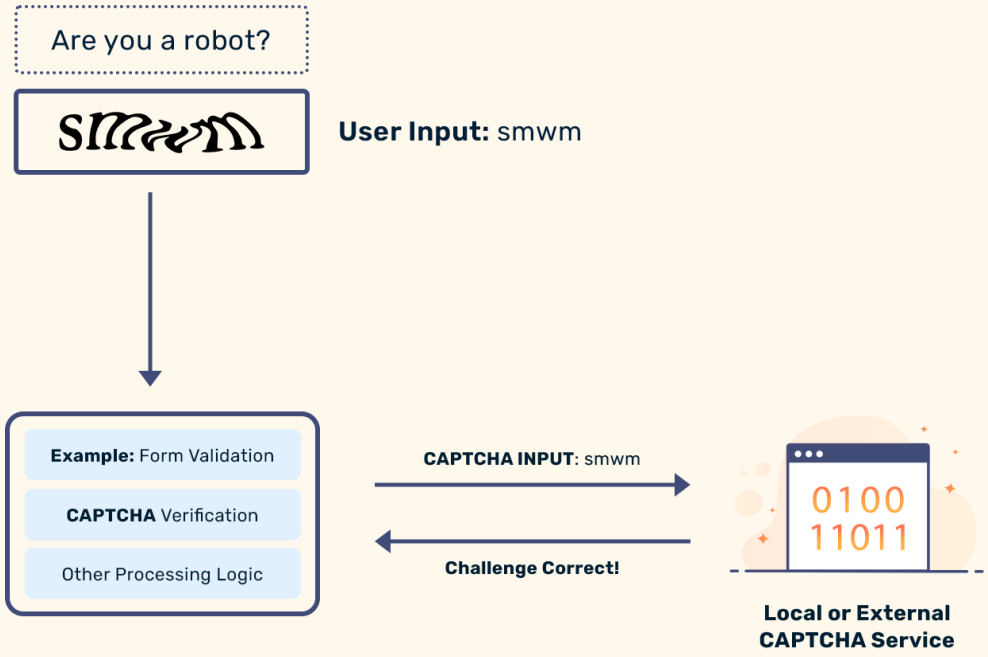
```python
import requests
import hashlib

def check_password(password):
    sha1_hash = hashlib.sha1(password.encode()).hexdigest().upper()
    prefix, suffix = sha1_hash[:5], sha1_hash[5:]
    response = requests.get(f"https://api.pwnedpasswords.com/range/{prefix}")

    return suffix in response.text

password = "monMotDePasseSuperSecret"
if check_password(password):
    print("Ce mot de passe a déjà été compromis !")
else:
    print("Ce mot de passe est sûr.")
```

1.3 Sécurisation du Formulaire d'Authentification

Sécuriser le formulaire d'authentification avant qu'il n'atteigne le serveur est essentiel pour protéger les informations sensibles des utilisateurs. Cette approche vise à prévenir diverses menaces, notamment les attaques par injection et la falsification de requêtes intersites (**CSRF**), qui ciblent souvent les formulaires comme points d'entrée vulnérables.

1.3.1 Implémentation de Systèmes CAPTCHA :

Pour protéger le formulaires contre les soumissions automatisées par des bots malveillants, nous intégrons des **systèmes CAPTCHA**. Ces tests permettent de différencier les utilisateurs humains des programmes automatisés, réduisant ainsi les risques d'abus et d'attaques par force brute.

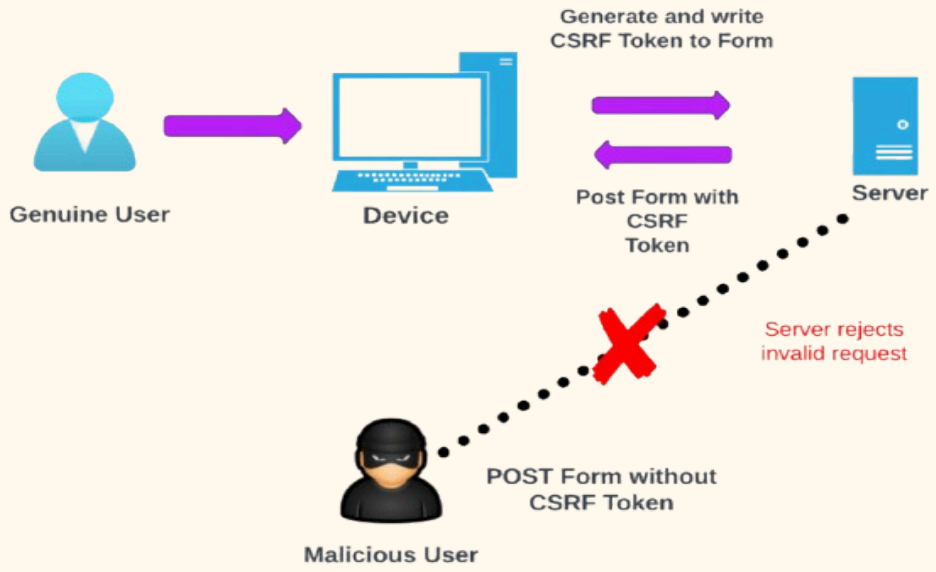


Fonctionnement d'un Test CAPTCHA pour la Protection des Formulaires

1.3.2 Utilisation de Jetons Anti-CSRF :

Une attaque CSRF est une menace sérieuse, qui se produit lorsqu'un utilisateur authentifié d'un site web est amené, à son insu, à exécuter une action non désirée sur ce site. Par exemple, un pirate pourrait envoyer une requête à votre insu, utilisant vos informations d'authentification pour exécuter une action que vous n'avez pas autorisée, comme modifier votre mot de passe ou effectuer un transfert bancaire.

C’est pour cela qu’on va utiliser la méthode de **tokens CSRF**. Un token CSRF est un jeton unique et aléatoire qui est généré par le serveur et inclus dans les formulaires HTML. Lors de la soumission du formulaire, ce token est vérifié par le serveur pour s'assurer qu'il correspond à celui stocké dans la session de l'utilisateur. Si le token ne correspond pas, la requête est rejetée.



1.4 Sécurisation d'Authentification

L’authentification est le processus qui suit la soumission du formulaire de connexion. C’est là où le serveur valide l’identité de l’utilisateur et lui accorde (ou non) l’accès à

l'application. A ce stade, plusieurs menaces peuvent compromettre la sécurité de la plateforme, notamment les attaques par **force brute**, le vol de sessions ou la divulgation d'informations sensibles.

1.4.1 Protection contre les attaques par force brute :

Les attaques par force brute consistent à tester une multitude de combinaisons de mots de passe jusqu'à en trouver un valide. Pour limiter ces tentatives :

- ➔ **Limiter le nombre d'essais** : Ajouter un délai entre les tentatives (ou bloquer un compte après un certain nombre d'échecs)
- ➔ **CAPTCHA après plusieurs échecs :** Empêche les bots d'essayer des milliers de combinaisons automatiquement**.**

1.4.2 Gestion des jetons d'authentification :

Les jetons d'authentification permettent aux utilisateurs de s'authentifier sans stocker leur session côté serveur.

- ➔ **Choix du type de jeton :**

  WT (JSON Web Token) est un jeton d'authentification compact et sécurisé. Les informations nécessaires à l'authentification de l'utilisateur sont contenues directement dans le token lui-même. Ainsi, contrairement aux sessions classiques qui requièrent un stockage côté serveur pour maintenir l'état de l'utilisateur, le JWT dispense en grande partie de ce mécanisme de stockage.

1.4.3 Gestion des sessions :

La gestion des sessions est essentielle pour sécuriser l'accès aux ressources après l'authentification. Une session mal gérée peut exposer les utilisateurs à des attaques comme le vol de session, l'usurpation d'identité ou les attaques de fixation de session.

Mécanismes misent en place pour sécuriser les sessions :

- ➢ Définition d'un **timeout** après une période d'inactivité (15-30 min).
- ➢ Fermer la session après une longue durée, même en activité (24h max).

- ➢ Génération d'un nouvel identifiant de session lorsque l'utilisateur de connecte pour éviter la fixation de session.
- ➢ Restreindre l'utilisation d'une même session sur plusieurs appareils simultanément.
- ➢ Utilisation des attributs sécurisés pour les cookies de session, notamment **HTTPOnly**, **secure** et **samesite**.

HTTPOnly permet d'éviter que l'identifiant de session soit accessible aux scripts Javascript, ce qui permet de contrer certaines méthodes de vol de session.

Secure informe les navigateurs que l'identifiant de session ne doit transiter que via des canaux HTTPS, ce qui permet de contrer certaines attaques basées sur l'écoute du réseau.

Samesite qui permet de se protéger contre des attaques de type CSRF.

2. Tableau de Bord

La sécurisation d'un tableau de bord nécessite une approche rigoureuse afin de s'assurer que l'accès aux informations sensibles et les actions possibles sont contrôlés et bien protégés. Voici un ensemble de mesures à appliquer :

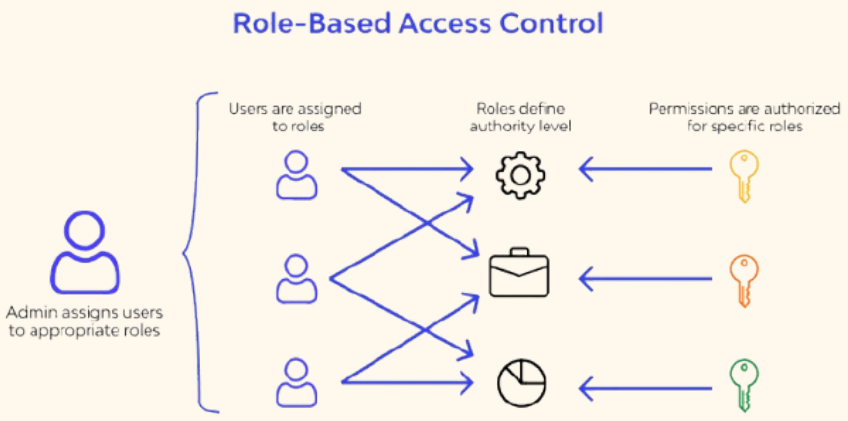
2.1 Contrôle d'accès basé sur les rôles (RBAC)

L'implémentation du contrôle d'accès basé sur les rôles (**RBAC**) est essentielle pour définir et restreindre les actions que chaque utilisateur peut effectuer en fonction de son rôle dans l'application.

Pourquoi ?
Le tableau de bord peut contenir des informations sensibles, telles que des données personnelles des utilisateurs ou des données de performance critiques. En fonction de leur rôle (administrateur, apprenants standard, professionnels...), certains utilisateurs peuvent avoir accès à des informations et fonctionnalités spécifiques tandis que d'autres

n'ont pas ce privilège. Le RBAC garantit que seuls les utilisateurs autorisés peuvent accéder à ces informations et effectuer des actions sensibles.

Exemple : Un administrateur pourrait avoir accès à la gestion des apprenants et à la configuration de l'application, tandis qu'un apprenant standard n'aurait accès qu'à son propre profil et à ses propres données.



2.2 Limitation des actions utilisateurs

Restreindre les actions possibles en fonction du rôle de l'utilisateur, notamment l'édition, la suppression ou l'accès aux données sensibles.

Si un utilisateur peut effectuer des actions non autorisées, cela peut entraîner des fuites de données ou des erreurs. Limiter les actions garantit que les utilisateurs n'effectuent que des actions correspondant à leur niveau de privilège.

2.3 Suivi des actions des utilisateurs

Implémentation d'un système d'audit pour suivre les actions des utilisateurs dans le tableau de bord, y compris les connexions, les modifications de données, et les changements de paramètres de l'application.

Pourquoi : Les journaux d'audit permettent de détecter toute activité suspecte et de suivre les actions des utilisateurs pour des raisons de sécurité et de conformité. Cela est crucial pour identifier toute tentative d'accès non autorisé.

2.2.1 Journalisation des connexions :

Ce qui doit être suivi :
Les connexions des utilisateurs (heure de connexion, adresse IP, type de dispositif, et adresse de connexion).

Cela permet de détecter toute tentative de connexion suspecte, comme un accès depuis une adresse IP inconnue ou un changement d'appareil. Cela peut aussi aider à retracer des actions en cas d'incident.

2.2.2 Utilisation d'outils de gestion des logs (ELK Stack) :

Parmi les outils de gestion des logs, **ELK Stack (Elasticsearch, Logstash,)** est une solution puissante et open-source adaptée à un tableau de bord. Il permet :

- ➔ **Centralisation des logs** : Logstash collecte et transforme les logs provenant de différentes sources.
- ➔ **Stockage et recherche rapide** : Elasticsearch indexe et stocke les logs, permettant une recherche efficace.
- ➔ **Visualisation et analyse** : Kibana permet de créer des tableaux de bord interactifs pour analyser les logs et détecter les anomalies.
- ➔ **Alertes en temps réel** : Possibilité de configurer des alertes pour signaler des événements critiques.

Exemple :

Un utilisateur se connecte au tableau de bord :

- ➢ **Logstash** récupère les logs de connexion.
- ➢ **Elasticsearch** indexe ces logs pour permettre une recherche rapide.
- ➢ **Kibana** affiche un graphique montrant les connexions récentes, avec des filtres pour identifier les connexions suspectes.

3. Catalogue de Formations

Le catalogue de formation est un composant essentiel de la plateforme. Il permet aux utilisateurs d'accéder aux formations proposées et de consulter leur contenu. Pour garantir la sécurité de cette section, nous mettons en place des mesures de protection à différents niveaux :

3.1 Cacher les boutons/actions interdites

Un utilisateur ne doit voir que les actions qui lui sont permises en fonction de son rôle et de son inscription. Cela empêche une mauvaise expérience utilisateur et limite les tentatives d'accès non autorisées.

Rôle / Action	Voir le catalogue	Accéder à une formation	Télécharger les ressources	Modifier une formation
Utilisateur non inscrit	Oui	Non	Non	Non
Apprenant inscrit	Oui	Oui (uniquement les formations inscrites)	Oui	Non
Formateur	Oui	Oui (ses propres formations)	Oui	Oui (ses propres formations)
Administrateur	Oui	Oui	Oui	Oui

- ➔ Conditionner l'affichage des boutons en fonction des permissions
- ➔ Désactiver les liens interdits (Si une action ne doit pas être possible)

3.2 Gérer les erreurs d'accès

Même si les boutons sont cachés, un utilisateur malintentionné peut essayer d'accéder directement à une formation en entrant l'URL manuellement. Il faut donc gérer les erreurs côté frontend et **backend**.

➔ Redirection automatique si l'utilisateur n'a pas l'accès, et affichage d'un message d'erreur clair

4. Mesures de sécurité globales

4.1 Protection contre le Cross-Site-Scripting (XSS)

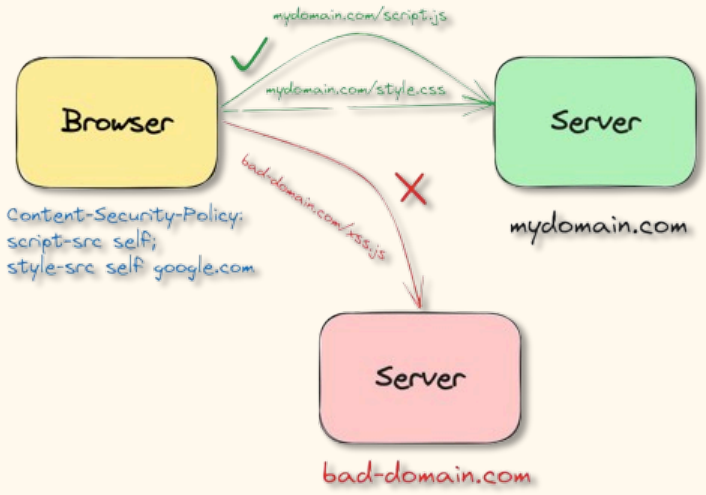
4.1.1 Qu'est-ce que le XSS ?

Le **Cross-Site Scripting (XSS)** est une faille de sécurité qui permet à un attaquant d'injecter du code malveillant dans une page web vue par d'autres utilisateurs. Cela peut entraîner le vol de sessions, la redirection vers des sites malveillants, l'affichage de faux formulaires pour du phishing, ou même l'exécution d'actions non autorisées sur le site.

4.1.2 Mesures de protection contre le XSS :

➔ **Échapper toutes les entrées utilisateur avant affichage :** Quand une donnée entrée par l'utilisateur doit être affichée sur le site, elle doit être transformée pour empêcher l'exécution de scripts.

➔ **Appliquer le Content Security Policy (CSP) :** C'est une règle de sécurité définie dans l'en-tête HTTP qui empêche le navigateur d'exécuter du JavaScript injecté.



4.1.3 Valider et filtrer les entrées utilisateur :

Avant de stocker ou afficher une donnée, elle doit être validée pour s'assurer qu'elle respecte les règles attendues, pour empêcher les injections de code malveillant dès l'entrée des données, et réduire la surface d'attaque en ne traitant que des entrées valides.

Exemple : Un champ "Nom" ne doit contenir que des lettres (Regex : `^[a-zA-Z\s]+$`).

4.2 Protection contre le Cross-Site Request Forgery (CSRF)

4.2.1 Qu'est-ce que le CSRF ?

Le CSRF est une attaque où un utilisateur authentifié exécute une action non désirée sur un site légitime à son insu. Un attaquant peut exploiter la session active d'un utilisateur pour effectuer des requêtes malveillantes (ex : modifier un mot de passe, effectuer un virement).

4.2.1 Mesures de Protection :

- Implémentation des jetons CSRF pour prévenir les attaques

  Les jetons CSRF sont des valeurs uniques générées aléatoirement côté serveur et envoyées au client. Ces valeurs étant uniques pour chaque requête, elles permettent de renforcer la sécurité d'une application en rendant difficile (voire impossible) pour un attaquant de les deviner et donc d'exploiter une vulnérabilité CSRF.

- **Restriction des requêtes CORS**

  Autorisation uniquement les **origines de confiance** (`Access-Control-Allow-Origin`), et bloquer les requêtes provenant de **sites tiers non autorisés**.

4.3 Protection contre le Clickjacking

Le **Clickjacking** est une attaque où un utilisateur pense cliquer sur un bouton légitime d'un site, alors qu'en réalité, il interagit avec un site malveillant qui le piège. Cela se fait en chargeant le site légitime dans un `<iframe>` invisible ou partiellement transparent sur une autre page. (achat,, suppression de données...).

4.3.1 Mesures de Protection :

- **Restriction du chargement en iframe**

  L'application doit empêcher son intégration dans un `<iframe>` non autorisé afin d'éviter que des sites malveillants ne l'affichent en superposition invisible. Cela protège l'utilisateur contre des actions involontaires et empêche des attaques exploitant l'interface du site.

- **Sécurisation des actions critiques**

  Les actions sensibles, telles que les modifications de compte ou transactions, doivent être confirmées manuellement par l'utilisateur. L'ajout d'une validation explicite, comme une confirmation ou un CAPTCHA, empêche l'exécution automatique d'actions via une interface détournée.

Même si l'attaque réussit à afficher le site dans un `<iframe>`, il faut empêcher l'exécution automatique d'actions sensibles :

- **Ajout d'une confirmation manuelle** (une boîte de dialogue "Êtes-vous sûr ?").
- **CAPTCHA ou authentification** pour valider certaines actions importantes.

III. Coté Serveur

1. API

Une grande partie de l'Internet moderne repose sur les API pour fonctionner. La sécurité des API est le processus de protection des API contre les attaques et les violations de données.

Les **API** de notre plateforme assurent la communication entre le front-end et le back-end, permettant la gestion des utilisateurs, des formations et des interactions pédagogiques. Cependant, elles constituent une surface d'attaque privilégiée et nécessitent une sécurisation rigoureuse.

Une API mal protégée peut exposer des **données sensibles** (informations personnelles, progression des utilisateurs, contenu des formations) et être la cible d'attaques telles que l'usurpation d'identité, l'injection de code ou encore le déni de service.

1.1 Réception des Requêtes API

Lorsqu'une requête est envoyée à l'API, elle représente un point d'entrée potentiel pour des attaques. Il est donc essentiel de mettre en place des **mécanismes de validation et de contrôle** avant tout traitement.

1.1.1 Validation et Filtrage des Données Entrantes :

Chaque requête doit être vérifiée contre un schéma précis pour s'assurer qu'elle ne contient que des données attendues et conformes. Tout paramètre inconnu ou mal formaté est rejeté. Cela pour empêcher l'injection de données malveillantes (SQL Injection, XSS..).

1.1.2 Authentification et Autorisation :

La sécurisation des accès doit reposer sur trois mécanismes essentiels :

a. **Authentification avec JWT :** Utilisation de JSON Web Token pour identifier les utilisateurs de manière sécurisée et éviter le stockage de sessions côté serveur. JWT permet une transmission sécurisée des identités.

b. **Autorisation avec OAuth 2.0** : Mise en place d'OAuth 2.0 pour gérer l'accès aux services tiers sans exposer les identifiants des utilisateurs. Ce protocole garantit un contrôle granulaire sur les permissions accordées aux applications externes.

c. **Gestion des accès par rôles :** Attribution de rôles et permissions spécifiques pour limiter les actions autorisées en fonction du niveau d'accès de chaque utilisateur. Cela empêche des abus et renforce la sécurité des données sensibles.

1.1.2 Limitation du Trafic et Protection DDoS :

Les attaques par déni de service (DDoS) visent à saturer un serveur avec un nombre excessif de requêtes, rendant le service indisponible ou inaccessible.

Mesure de protection :

➔ **Rate limiting :** Limite le nombre de requêtes autorisées par utilisateur/IP sur une période définie, pour protéger la plateforme contre une surcharge volontaire qui pourrait impacter la disponibilité des services.

➔ **Utiliser des CAPTCHA** : empêche les attaques automatisées sur les actions sensibles (connexion, inscription).

➔ **Firewall d'application web (WAF) :** Filtre et bloque les requêtes malveillantes avant qu'elles n'atteignent le serveur.

1.2 Traitement des requêtes (traitement interne)

Une fois une requête reçue et authentifiée, son traitement doit être sécurisé pour éviter toute manipulation malveillante et garantir l'intégrité des données.

1.2.1 Validation et nettoyage des données :

Même après une première validation lors de la réception, les données doivent être réévaluées avant d'être utilisées. Cela permet d'empêcher toute tentative d'injection SQL, XSS ou autre manipulation dangereuse.

1.2.2 Contrôle des permissions :

À chaque étape du traitement, il est essentiel de vérifier que l'utilisateur dispose des droits nécessaires pour l'action demandée. Cette vérification empêche les abus et protège les données sensibles.

1.3 Gestion des Sessions et des Tokens

La gestion des sessions et des tokens côté serveur est essentielle pour garantir une authentification sécurisée et éviter les abus.
Les API doivent vérifier l'identité des utilisateurs sans exposer leurs identifiants à chaque requête. Deux approches existent :

- **Sessions côté serveur** : L'API stocke l'état de connexion et associe un identifiant de session à l'utilisateur via un cookie.
- **Tokens JWT** : L'API génère un **token signé**, que le client envoie à chaque requête.

1.3.1 Stockage sécurisé des tokens :

➢ Éviter `localStorage` (risque de vol via XSS).
➢ Préférer les cookies `HTTP-only` pour empêcher l'accès au token par JavaScript.

1.3.2 Protection contre le Vol de Session :

Les attaques visant les tokens sont fréquentes, notamment le vol de JWT via le stockage local ou les attaques XSS. Pour sécuriser les sessions côté serveur :

Utilisation de cookies sécurisés : Plutôt que de stocker les tokens en "localStorage", les stocker dans des cookies HTTP-only pour les protéger contre le vol.

Mise en place de restrictions IP ou device : Un token ne doit être valide que pour l'adresse IP ou le device où il a été initialement émis.

2. Gestion des Permissions

La gestion des permissions est essentielle pour contrôler qui peut accéder à quelles ressources (modules, quiz, forums...) et quelle action chaque utilisateur peut réaliser (visualiser, créer, modifier, supprimer). Cette gestion est essentielle pour garantir la sécurité des données et des actions des utilisateurs.

2.1 Rôles et Permissions

Les utilisateurs de la plateforme sont assignés à des rôles spécifiques, chacun ayant un ensemble de permissions. Ces rôles définissent le niveau d'accès à la plateforme et les actions possibles.

2.1.1 Rôle :

Définit le type d'utilisateur et son niveau d'accès. Exemples :

- Apprenant : Peut visualiser les cours et participer.
- Formateur : Peut créer, modifier, et supprimer des cours.
- Admin : A accès à toutes les ressources de la plateforme, peut gérer les utilisateurs, consulter les statistiques et configurer les paramètres du site.

2.1.2 Permissions :

Détaille les actions autorisées pour chaque rôle. Exemples :

- Lire un cours, télécharger un matériel pédagogique.
- Créer ou modifier des leçons ou des examens.
- Gérer les utilisateurs ou les affectations de cours.

2.1.3 Mise en œuvre de la gestion des permissions :

- ➔ **Vérification des rôles à l'authentification** :
  Lors de la connexion de l'utilisateur, son rôle est déterminé et stocké dans son token JWT ou une session. Cela permet au serveur de connaître le niveau d'accès de l'utilisateur pour chaque nouvelle requête. Par exemple, un étudiant ne pourra pas accéder aux fonctionnalités d'administration.
- ➔ **Contrôle d'accès basé sur les rôles (RBAC)** : Lors de chaque requête, le serveur vérifie si l'utilisateur a les permissions appropriées pour effectuer l'action demandée.

3. Gestion des Utilisateurs

Afin de garantir la sécurité des utilisateurs et de leurs données sensibles sur la plateforme, il est impératif de mettre en place une stratégie robuste de gestion des utilisateurs, en intégrant des mécanismes de protection avancés à chaque niveau du processus.

3.1 Stockage Sécurisé des Informations Utilisateurs

1.4.1 Hachage des mots de passe :

Tous les mots de passe doivent être hachés à l'aide d'algorithmes sécurisés comme bcrypt. Ce mécanisme empêche qu'un mot de passe soit stocké en clair et garantit sa sécurité même en cas de fuite de base de données.

1.4.2 Salt pour le hachage :

Chaque mot de passe doit être combiné avec un **Salt** unique avant d'être haché. Cela rend les attaques par dictionnaire ou force brute beaucoup plus complexes, en rendant impossible la prédiction de plusieurs mots de passe en parallèle.

1.4.3 Cryptage des données sensibles :

Les informations sensibles comme les numéros de téléphone ou les informations bancaires doivent être cryptées pour empêcher l'accès non autorisé.

Les clés de chiffrement doivent être stockées de manière sécurisée, idéalement dans un gestionnaire de secrets ou un HSM (Hardware Security Module).

4. Logs et Monitoring

Pour assurer la **sécurité et la traçabilité** des actions sur la plateforme, un système de **logs et de monitoring efficace** doit être mis en place afin de détecter les comportements anormaux, prévenir les intrusions et faciliter l'analyse des incidents de sécurité.

4.1 Journalisation des événements

4.1.1 Enregistrés et horodatés les événements critiques

- Tentatives de connexion réussies et échouées.
- Modifications des permissions et des rôles utilisateurs.
- Accès aux données sensibles.
- Actions administratives et mises à jour système.

Les **logs ne doivent jamais contenir de données sensibles** (ex : mots de passe, informations bancaires).

4.2 Stockage sécurisé des logs

- Les logs doivent être **stockés dans un environnement sécurisé**, **protégé contre toute altération ou suppression non autorisée**.
- Mise en place d'une **signature numérique** pour garantir **l'intégrité des logs**.
- Conservation des logs pendant une **durée suffisante** pour répondre aux exigences légales.

4.2.1 Monitoring et détection des anomalies :

Mise en place d'un **système de détection des comportements anormaux** :

- ➢ Détection des **tentatives de connexion suspectes** (ex : plusieurs échecs consécutifs, connexion depuis un pays inhabituel).
- ➢ Identification des **pics d'activité inhabituels** pouvant signaler une attaque.
- ➢ Surveillance des accès aux données sensibles et aux endpoints critiques.

4.2.2 Alertes et réponse aux incidents :

Définition de **seuils d'alerte** pour signaler **toute activité suspecte**.
Mise en place d'un **système de notification** pour informer immédiatement l'équipe de sécurité en cas d'incident critique.
Automatisation de certaines actions (blocage temporaire d'un compte après plusieurs tentatives de connexion échouées**).**

5. Bases de Données

La sécurisation des bases de données est essentielle pour protéger les informations sensibles et garantir l'intégrité et la confidentialité des données stockées. Un accès non contrôlé aux bases de données peut entraîner des fuites d'informations, des attaques par injection SQL, ou encore des altérations de données.

5.1 Contrôle d'accès à la base de données

5.1.1 Gestion des utilisateurs et des permissions(BDD):

Seules les personnes autorisées doivent avoir accès à la base de données, avec des permissions minimales selon leurs besoins.
Mise en place de rôles et permissions clairs, en suivant le principe du **moindre privilège**.

5.1.2 Chiffrement des données sensibles :

- Toutes les données sensibles (mots de passe, informations bancaires, données personnelles) doivent être **chiffrées** à l'aide d'algorithmes robustes (AES-256).
- Le chiffrement doit être appliqué **tant au repos qu'en transit** pour garantir la sécurité des données, même en cas d'accès non autorisé à la base de données.

5.1.3 Protection contre les injections SQL :

- Utilisation de **requêtes paramétrées** pour éviter les risques d'injection SQL.
- Mise en place de mécanismes de filtrage des entrées et de validation pour empêcher l'exécution de commandes malveillantes.

5.1.4 Sauvegardes sécurisées :

- Les données doivent être sauvegardées de manière régulière, avec des copies **chiffrées** et stockées dans un emplacement sécurisé.
- Mise en place d'un **plan de récupération après sinistre (DRP)** pour assurer une récupération rapide en cas de perte ou de corruption des données.

5.1.5 Mise à jour et gestion des vulnérabilités :

- **Mise à jour régulière** des systèmes de gestion de base de données (SGBD) pour appliquer les correctifs de sécurité et prévenir l'exploitation de vulnérabilités connues.
- Mise en place d'une **stratégie de patching** rapide en cas de détection de nouvelles vulnérabilités.

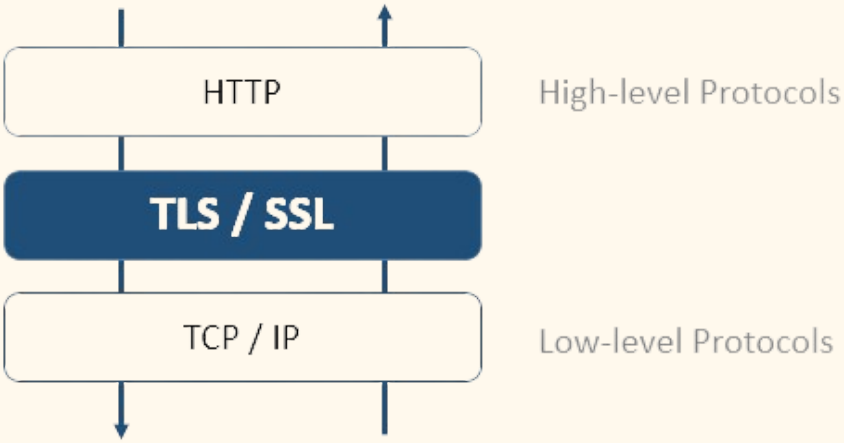
IV. Sécurisation des Échanges

La protection des échanges de données entre les utilisateurs et les serveurs est essentielle pour éviter les interceptions, manipulations ou vols d'informations sensibles. La sécurisation des communications permet de garantir **la confidentialité, l'intégrité et l'authenticité** des données échangées.

1. Protection des échanges via HTTPS (TLS)

Le protocole **TLS (Transport Layer Security)** est **obligatoire** pour sécuriser les échanges entre client et serveur. Il chiffre les données pour empêcher les attaques comme l'écoute clandestine (eavesdropping) ou l'attaque Man-in-the-Middle (MitM).

Le protocole TLS empêche l'interception des données , vérifie que le serveur est bien celui qu'il prétend être via un certificat SSL/TLS, et empêche la modification des requêtes en transit.



2. Protection contre l'interception des requêtes (MitM)

Si un attaquant intercepte les échanges entre l'utilisateur et le serveur, il peut récupérer des données sensibles.

Éviter les réseaux Wi-Fi publics non sécurisés .

Désactiver les protocoles faibles (SSLv3, TLS 1.0).

Utiliser le chiffrement des données sensibles dans les requêtes et réponses.

3. Protection contre l'interception des requêtes (MitM)

Les **en-têtes HTTP** permettent de protéger contre plusieurs attaques :

- **Content Security Policy (CSP)** : Empêche l'injection de scripts malveillants.
- **Strict-Transport-Security (HSTS)** : Force HTTPS sur le long terme.
- **X-Content-Type-Options: nosniff** : Empêche l'interprétation des fichiers en tant que scripts malveillants.

v. Annexes :

Bases d'authentification
https://laconsole.dev/formations/authentification

Sécurité de connexion
https://www.microsoft.com/fr-fr/security/business/security-101/what-is-login-security

La méthode la plus sécurisée pour se connecter
https://safety.google/intl/fr_fr/authentication/

Sécurité par Mot de Passe
https://www.proofpoint.com/fr/threat-reference/password-protection

Regex (expression régulière)
https://datascientest.com/regex-tout-savoir

Bien gérer son mot de passe
https://www.cybermalveillance.gouv.fr/tous-nos-contenus/bonnes-pratiques/mots-de-passe

Créer un mot de passe sécurisé
https://www.economie.gouv.fr/particuliers/creer-mot-passe-securise

Conseils pour créer et gérer les mots de passe
https://www.priv.gc.ca/fr/sujets-lies-a-la-protection-de-la-vie-privee/technologie/protection-de-la-vie-privee-en-ligne-surveillance-et-temoins/protection-de-la-vie-privee-en-ligne/tips_pw/

Choisir un bon mot de passe
https://cri.centrale-med.fr/fr/faq/choisir-un-bon-mot-passe

Gestion des sessions d'authentification
https://www.vaadata.com/blog/fr/comment-securiser-les-systemes-dauthentification-de-gestion-de-sessions-et-de-controle-dacces-de-vos-applications-web/csp:https://www.writesoftwarewell.com/content-security-policy/

TLS (Transport Layer Security)
https://fr.wikipedia.org/wiki/Transport_Layer_Security

Détournement de clic
https://fr.wikipedia.org/wiki/D%C3%A9tournement_de_clic

JWT (Json Web Token)
https://grafikart.fr/tutoriels/json-web-token-presentation-958