ECF_EcoRide_TP – Développeur Web et Web Mobile

NOM	**AGOSTINHO**
Prénom	**Yannis**
Date de naissance	**14/08/2000**

Lien du git :

Lien de l'outil de gestion de projet :

Lien du déploiement :

Login et mot de passe administrateur :

Partie 1 : Analyse des besoins

Résumé du projet – EcoRide

Ecoride, est une plateforme de covoiturage pensée pour les personnes qui veulent se déplacer autrement tout en limitant l'impact sur l'environnement. L'idée est simple : mettre en contact des conducteurs et des passagers qui partagent le même trajet et qui roulent avec des véhicules propres ou à faibles émissions.

Aujourd'hui, il existe déjà des sites de covoiturage, mais aucun qui soit vraiment centré sur l'aspect écologique. EcoRide vient combler ce manque avec une interface claire, rapide à prendre en main et sécurisée. Le but est que n'importe qui puisse rechercher, réserver ou proposer un trajet en quelques clics, sans se prendre la tête.

Chaque utilisateur pourra créer son compte, voir les trajets disponibles, réserver avec un système de crédits, ou proposer ses propres trajets s'il est conducteur. En coulisses, un back-office permettra aux administrateurs et aux employés de gérer les comptes, modérer les avis et garder le service propre et fiable

Côté technique, le site sera développé en HTML5, CSS3 Bootstrap 5 et JavaScript pour le front, avec un back-end en PHP natif (architecture MVC) et une base de données MySQL (PDO). Une base NoSQL (MongoDB) pourra être utilisée pour certains types de données. Le code sera versionné avec Git/GitHub et le déploiement se fera via FTP sur un hébergeur mutualisé, et MongoDB Atlas pour la base NoSQL

En bref, EcoRide veut rendre le covoiturage plus vert, plus simple et plus accessible à tous.

Problématique (UX)

Comment permettre à des voyageurs soucieux de l'environnement de rechercher, proposer et réserver facilement des trajets en covoiturage, tout en garantissant une expérience simple, sécurisée et adaptée à leurs attentes ?

Persona (profil type)

Nom : Claire Marchand, 29 ans, ingénieure en environnement

Situation : vit en périphérie urbaine, utilise régulièrement les transports pour ses déplacements pro/perso.

Besoins : réduire son empreinte carbone, trouver des trajets fiables à moindre coût.

Freins : peur des plateformes trop complexes ou peu sécurisées.

Attentes : application claire, accès rapide aux trajets éco, confiance dans les conducteurs grâce aux avis.

MVP retenu (Minimum Viable Product)

Pour lancer la plateforme EcoRide rapidement et répondre au besoin essentiel, 3 pages sont prioritaires :

1. **Accueil** : formulaire de recherche + présentation rapide du service.

2. **Liste trajets** : affichage des résultats avec infos essentielles (conducteur, prix, places, éco ou non).
3. **Fiche trajet** : détail complet + CTA “Réserver”.

Spécifications fonctionnelles (User Stories US1 → US13 résumées)

- **US1–US2** : Accueil avec présentation, recherche et menu global.
- **US3–US4** : Liste trajets avec conducteur, prix, places, éco ou non + filtres (prix, durée, note).
- **US5–US6** : Fiche trajet avec détail complet + système de réservation par crédits avec double confirmation.
- **US7** : Création de compte avec pseudo, mail, mot de passe sécurisé (+ 20 crédits offerts).
- **US8–US9** : Espace utilisateur avec gestion du rôle (chauffeur/passager), infos véhicule et création trajet.
- **US10–US11** : Historique, annulation, démarrage/arrêt de trajet avec validation par avis.
- **US12** : Espace employé pour modérer les avis et gérer les incidents.
- **US13** : Espace administrateur pour statistiques, crédits et gestion des comptes.

Partie 2 : Spécifications technique

1. Spécifiez les technologies que vous avez utilisé en justifiant les conditions d’utilisation et pourquoi le choix de ses éléments

Stack technique utilisée

Le projet EcoRide a été développé en respectant les contraintes imposées par l’ECF, tout en tenant compte du périmètre MVP, du niveau de maîtrise des technologies et du temps de réalisation disponible.

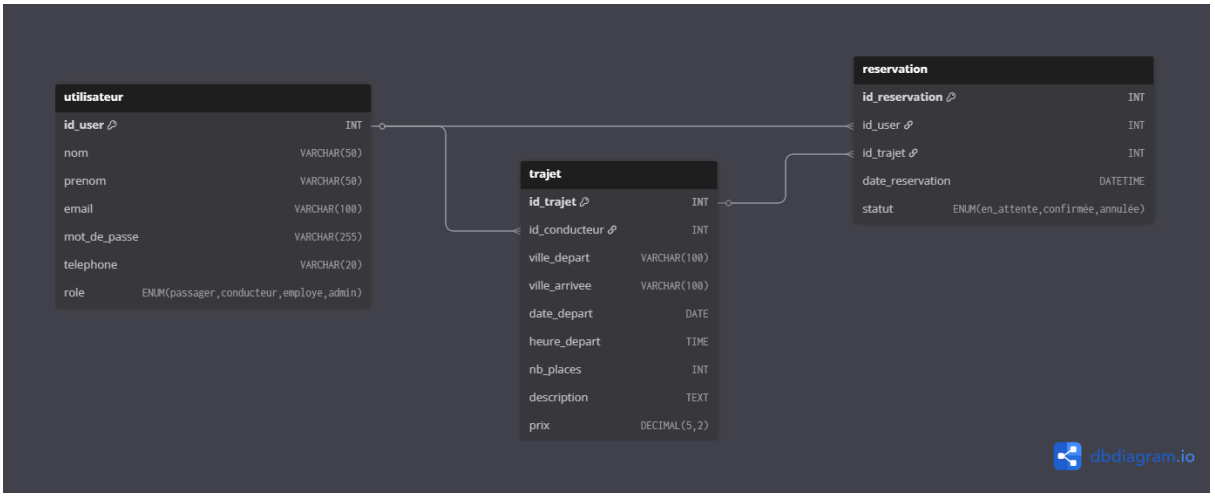
Domaine	Technologie	Justification
Front-end	HTML5, CSS3	Standards web, maintenables, supportés partout
	Bootstrap 5	Grille responsive rapide, composants UI intégrés
	JavaScript (Vanilla)	Suffisant pour interactions simples (formulaires, toggle, etc.)
Back-end	PHP natif (MVC manuel)	Requis par l'ECF, logique métier claire, bonne maîtrise
	MySQL via PDO	Requêtes préparées sécurisées, cohérent avec PHP MVC
NoSQL	Simulé via classe PHP ou JSON	Permet de stocker les avis/logs sans implémenter MongoDB réel
Versioning	Git + GitHub	Branches `main`, `dev`, `feature/*` selon Git Flow
UI/Design	Figma	Mockups + charte graphique mobile-first (AAA contrast, composant UI)
Projet	Notion	Kanban de production, suivi des tâches, checklist ECF
Modélisation BDD	dbdiagram.io	Génération du MCD propre et exportable dans la documentation

Modèle de données – MCD

Le schéma suivant représente le MCD du projet **EcoRide**, basé sur le périmètre MVP défini dans les User Stories US1 à US9.

Il comprend les entités essentielles : `utilisateur`, `trajet`, `reservation`, et un champ `role` pour gérer les accès.

Les autres rôles (`admin`, `employé`) seront activés plus tard via ce champ, mais **ne sont pas modélisés ici pour ne pas alourdir la base inutilement**, conformément à l'approche MVP.



Justification des choix :

- Respect du **périmètre MVP** imposé par l'énoncé (US1 à US9), sans surcharger inutilement la base de données
- Structure simple, évolutive, et conforme à la logique PHP MVC
- Le champ `role` permet de différencier les accès :
  - `passager` (utilisateur de base)
  - `conducteur` (peut publier des trajets)
  - `employe` (modère les avis – US12)
  - `admin` (gère les comptes et les crédits – US13)
- Le rôle `visiteur` n'est pas stocké en base (il n'a pas de compte)
- Les **avis** sont stockés via MongoDB simulé (classe PHP ou JSON), donc exclus de la base relationnelle
- Le champ `role` centralise la gestion des droits sans multiplier les tables (`admin`, `employe`, etc.)

> 💡 Remarque : ce MCD a été généré avec `dbdiagram.io` puis exporté pour la documentation.

UML de classes – EcoRide

Le schéma suivant représente le **diagramme UML de classes** du projet EcoRide.

Il illustre la structure orientée objet de l'application, basée sur les entités principales : **Utilisateur**, **Trajet**, et **Réservation**.

Ces classes définissent les attributs, méthodes et relations nécessaires pour supporter la logique **MVC en PHP natif**.

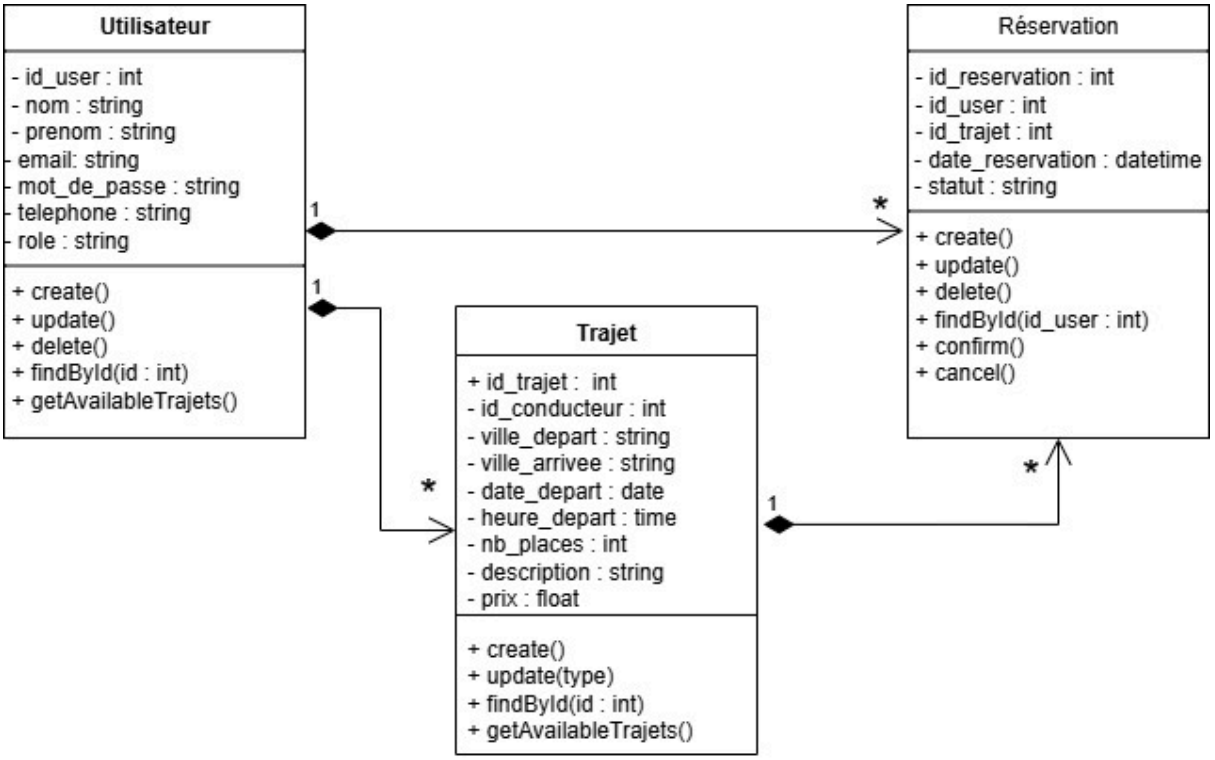


Diagramme UML de classes - ecoride

Justification des choix :

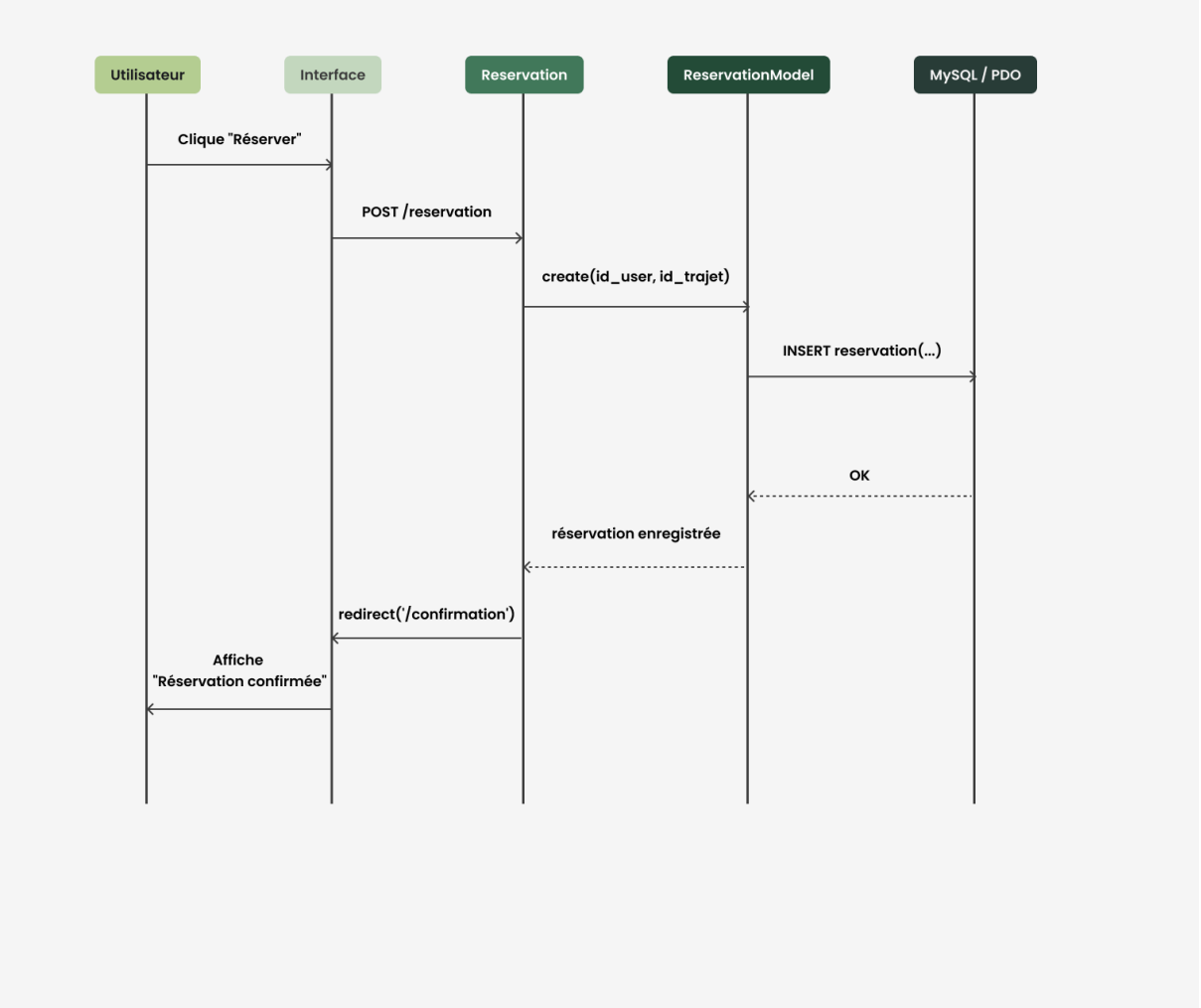
- Respect du périmètre MVP (US1 → US9) : seules les classes essentielles sont modélisées.
- Structure conforme à l'architecture MVC : chaque classe correspond à un **Model** PHP.
- Les attributs traduisent directement les colonnes SQL (`id_user`, `ville_depart`, `statut`...), assurant la cohérence entre base et logique métier.
- Les méthodes standard CRUD (`create()`, `update()`, `delete()`, `findById()`) garantissent la maintenabilité et l'évolutivité.
- Les relations sont clairement exprimées :
  - **Utilisateur ↔ Trajet** : un conducteur peut publier plusieurs trajets.

    - **Utilisateur ↔ Réservation** : un passager peut effectuer plusieurs réservations.
    - **Trajet ↔ Réservation** : un trajet peut regrouper plusieurs réservations.
- Les cardinalités `1..*` valident la logique métier : un utilisateur peut être lié à plusieurs trajets ou réservations, mais chaque trajet/réservation appartient à un utilisateur unique.
- Ce modèle UML sert de **pont** entre le MCD (niveau BDD) et l'implémentation PHP (niveau code).

UML de séquence – *Réserver un trajet* (EcoRide)

Le schéma ci-dessous modélise le **scénario dynamique** de réservation d'un trajet dans EcoRide.

Il décrit l'enchaînement des interactions entre l'**Interface** (front), le **ReservationController**, le **ReservationModel** et la **base MySQL via PDO**, depuis le clic utilisateur jusqu'à l'affichage de la confirmation.

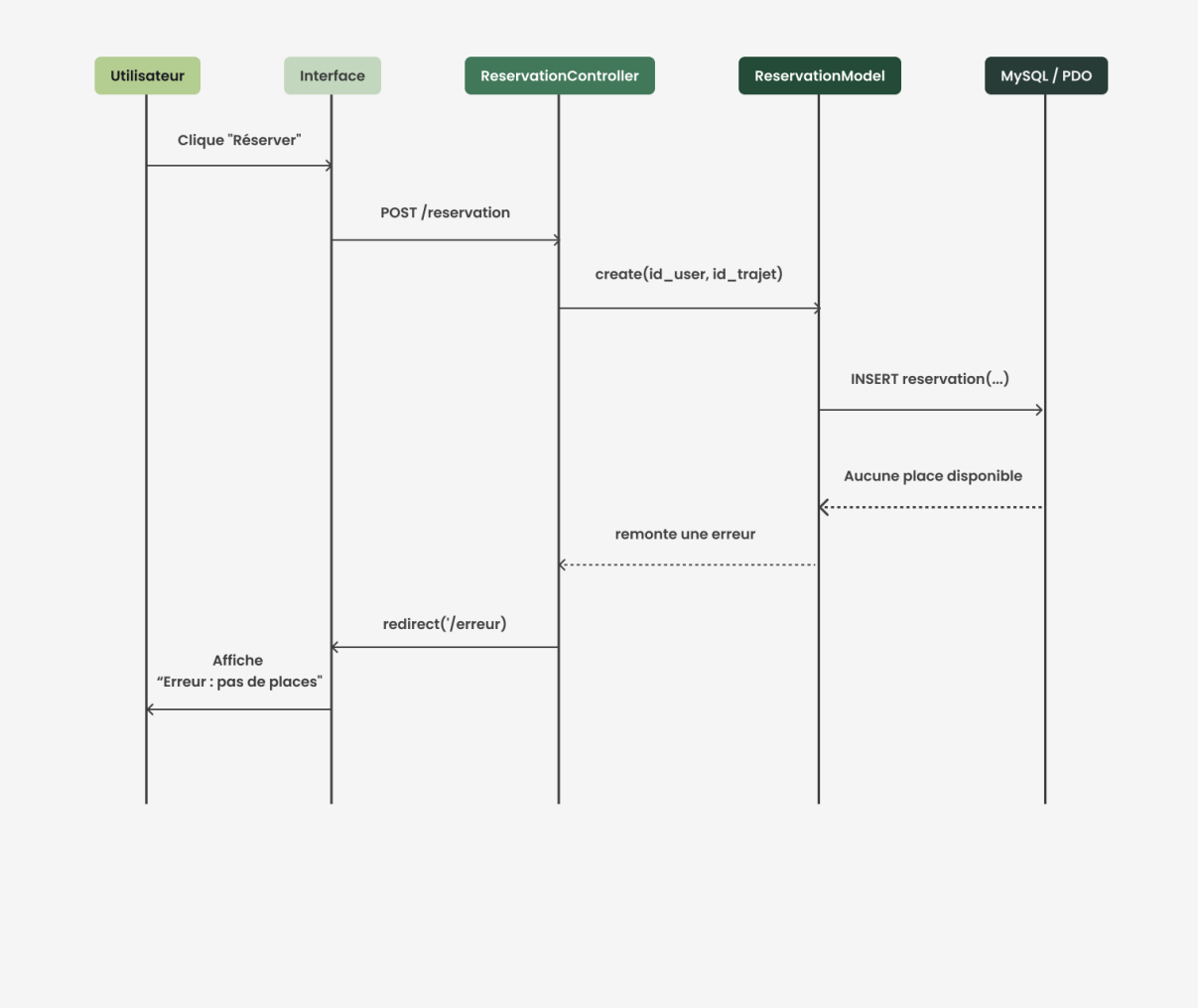


Justification des choix

- Couvre le **parcours critique du MVP** (US5–US6) : *réserver un trajet*.
- **Séparation MVC** explicite : Interface → Controller → Model → BDD (responsabilités claires).
- Messages clés normalisés :
  - `POST /reservation` (soumission du formulaire)
  - `create(id_user, id_trajet)` (logique métier côté modèle)
  - `INSERT reservation(...)` → retour **OK** (PDO)
  - `redirect('/confirmation')` → **affichage confirmation**
- Les **retours sont pointillés** (réponses) pour distinguer appels vs résultats.
- Prépare l'implémentation PHP : routes, contrôleur `ReservationController`, modèle `ReservationModel`, et couche PDO.

- Sert de **référence de test** : ce flux devient le fil conducteur pour les tests d'intégration (succès/erreur/annulation).

Diagramme UML de séquence – *Réserver un trajet (échec)*



Justification des choix

- Ce diagramme représente un **scénario alternatif** au parcours MVP : l'utilisateur essaie de réserver un trajet complet.
- L'architecture **MVC** est toujours respectée : l'interface déclenche un `POST`, le controller délègue à un modèle, qui interagit avec la base via PDO.
- La base retourne une **erreur métier** : `Aucune place disponible`.
- Le modèle remonte cette erreur au controller, qui redirige l'utilisateur vers `/erreur`.
- L'interface affiche un message clair : *Erreur : pas de places*.

- Ce cas permet de tester la **résilience de l'application** et la gestion des retours utilisateurs.
- Ce scénario est conforme à l'US6 et au périmètre MVP, car il anticipe les cas d'**échec de réservation** sans casser l'expérience utilisateur.

2. Comment avez-vous mis en place votre environnement de travail ? Justifiez vos choix. (README.md)

J'ai configuré un environnement **LAMP** (Linux Mint, Apache2, MySQL, PHP 8.2) pour travailler en conditions proches d'un serveur de production.

- **Apache2** : gestion des VirtualHost (http://ecoride.local) pour séparer le projet des autres sites.
- **PHP natif (MVC manuel)** : respect des contraintes ECF, meilleure compréhension du fonctionnement interne.
- **MySQL (PDO)** : gestion sécurisée des requêtes préparées et compatibilité large.
- **Git/GitHub** : suivi des versions avec branches `main`, `dev`, `feature/*`.
- **VS Code** : IDE léger avec extensions Git + PHP.
- **phpMyAdmin** : gestion rapide et visuelle de la base de données.

Ce choix permet d'avoir un environnement proche de la réalité professionnelle et facilement déployable ensuite via FTP.

---

3. Énumérez les mécanismes de sécurité que vous avez mis en place, aussi bien sur vos formulaires que sur les composants front-end ainsi que back-end.

- **Hashage des mots de passe** avec `password_hash()` (algorithme Bcrypt).
- **CSRF Token** intégré dans les formulaires via `Security::csrfField()`.
- **Validation et sanitation** : `filter_input`, `htmlspecialchars`, `trim`.
- **Requêtes préparées PDO** → prévention SQL Injection.
- **Sessions sécurisées** : `session_regenerate_id(true)`, `httponly`.

- **Protection XSS** : échappement des variables affichées côté vues.
- **Contrôles côté client** (HTML5, Bootstrap) + **côté serveur** (PHP) pour renforcer la robustesse.

---

4. Décrivez une veille technologique que vous avez effectuée, sur les vulnérabilités de sécurité.

J'ai étudié les recommandations de l'**OWASP (Open Web Application Security Project)** et la documentation PHP officielle pour identifier les failles les plus critiques (XSS, CSRF, injections SQL).

Cette veille m'a permis d'intégrer des pratiques modernes de sécurisation des formulaires, d'utilisation de sessions et de gestion des requêtes préparées.

Partie 3 : Recherche

1. Décrivez une situation de travail ayant nécessité une recherche durant le projet à partir de site anglophone. N'oubliez pas de citer la source.

Lors de la mise en place de la sécurisation des formulaires, j'ai recherché comment éviter les attaques **CSRF (Cross-Site Request Forgery)**.

Source : OWASP – CSRF Prevention Cheat Sheet

2. Mentionnez l'extrait du site anglophone qui vous a aidé dans la question précédente en effectuant une traduction en français.

Extrait (EN) :

"CSRF attacks allow an attacker to perform actions on behalf of authenticated users without their consent."

Traduction (FR) :

"Les attaques CSRF permettent à un attaquant d'effectuer des actions au nom d'utilisateurs authentifiés sans leur consentement."

Partie 4 : Informations complémentaire

Autres ressources

- Dépôt GitHub : https://github.com/Potato-Technical/ecoride
- Kanban Notion (gestion projet).
- Documentation projet dans `/docs` du dépôt (déploiement, GitFlow, manuel utilisateur).
- Figma

Informations complémentaires

Le déploiement complet via FTP n'a pas été réalisé, mais tous les tests ont été faits en local (Linux Mint).

Les identifiants de test (admin) :

- Email : `admin@example.test`
- Mot de passe : `Admin123!`