Table des matières

Compétences du référentiel couvertes par le projet

Voici la liste des compétences que j’ai pu développé pendant ma période d’activité en entreprise :

- Développer la partie front-end d’une application web ou web mobile sécurisée
  - ✔ Installer et configurer son environnement de travail en fonction du projet web ou web mobile
  - ✔ Maquetter des interfaces utilisateur web ou web mobile
  - ✔ Réaliser des interfaces utilisateur statiques web ou web mobile
  - ✔ Développer la partie dynamique des interfaces utilisateur web ou web mobile
- Développer la partie back-end d’une application web ou web mobile sécurisée
  - ✔ Mettre en place une base de données relationnelle
  - ✔ Développer des composants d’accès aux données SQL et NoSQL
  - ✔ Développer des composants métier coté serveur

Contexte du projet

Ying Zhang : présentation : avant, pourquoi et une photo

L’entreprise

Expression des besoins

J’avais pour mission de développer une plateforme de formation en ligne offrant des capsules vidéo, des parcours structurés et des activités interactives via une architecture moderne.

Les cibles sont les amateurs souhaitant se professionnaliser dans les domaines du digital, les juniors et les apprenants préparant un titre professionnel ainsi que les centres de formation.

Au lancement du projet **Koabana**, les besoins n’étaient pas formalisés par écrit. Afin de mieux structurer la vision du projet et d’identifier les fonctionnalités essentielles, une phase d’enquête a été menée auprès du donneur d’ordre et des parties prenantes.

Les user stories

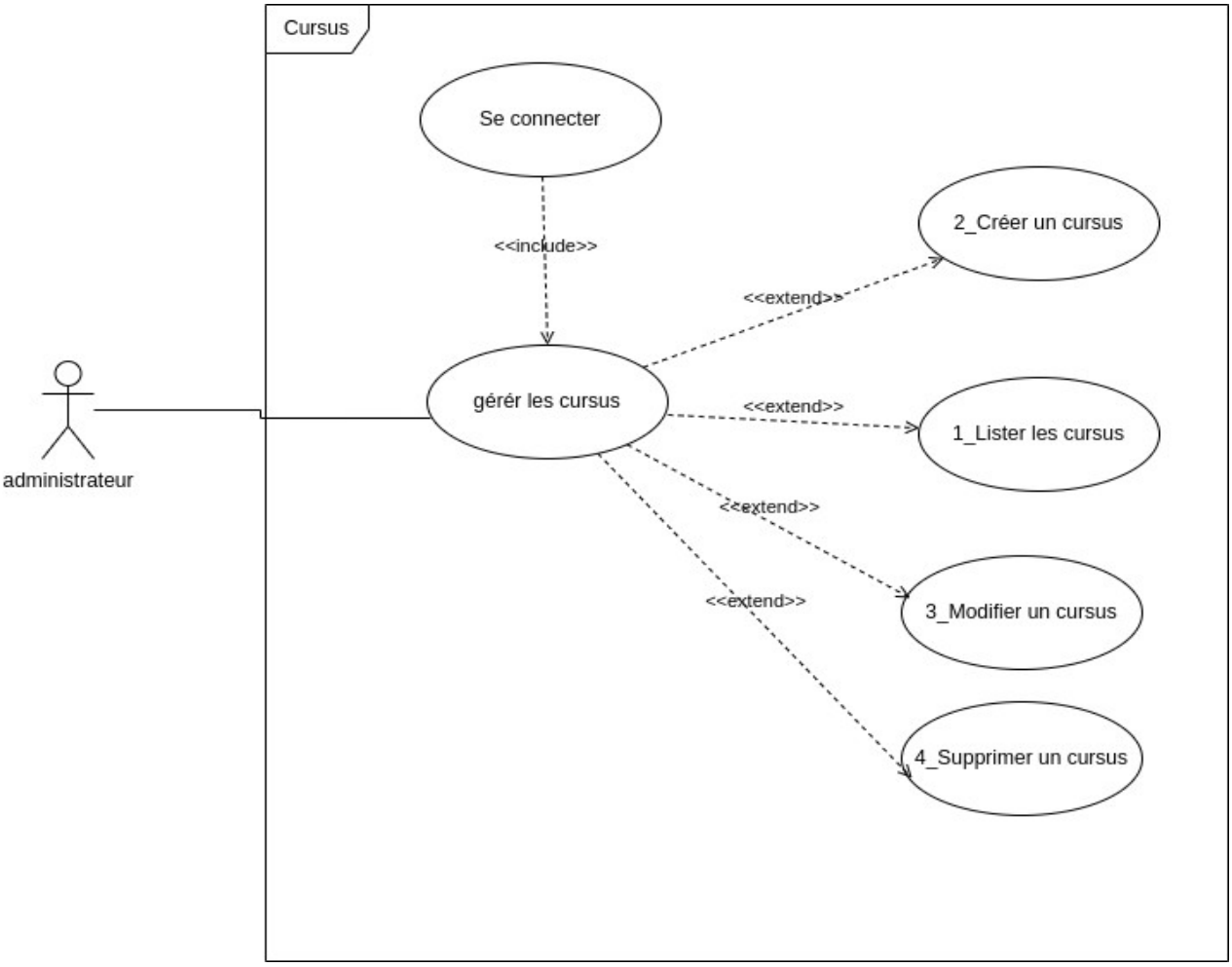
Sur la base des informations recueillies, les besoins ont été **structurés sous forme de user stories**, classées par **epics** pour regrouper les grandes fonctionnalités du projet. Chaque user story exprime un besoin métier du point de vue de l’utilisateur final et permet de cadrer les attentes de manière claire et compréhensible.

- Back-Office
  - Gestion des Contenus :
    - En tant qu’administrateur, je veux pouvoir créer, modifier et supprimer des cursus pour structurer les parcours pédagogiques.
    - En tant qu’administrateur, je veux gérer les sections pour organiser les contenus de manière hiérarchique.
    - En tant qu’administrateur, je veux associer des vidéos et des textes explicatifs aux capsules pour enrichir l'expérience utilisateur.
    - En tant qu’administrateur, je veux créer des capsules premium et pouvoir héberger des vidéos associées à ces capsules
  - Gestion des Utilisateurs :
    - En tant qu’administrateur, je veux visualiser la liste des utilisateurs inscrits pour suivre leur activité sur la plateforme.
    - En tant qu’administrateur, je veux attribuer ou révoquer des droits d'accès (ex. : KoaPro, KoaCampus) pour contrôler les abonnements.
  - ...etc
- Front-Office
  - Exploration et Consommation :
    - En tant qu'utilisateur, je veux explorer les cursus et sections disponibles pour choisir un parcours adapté à mes besoins.
    - En tant qu'utilisateur, je veux accéder aux capsules gratuites directement sur le site pour apprendre à mon rythme.

        - ...etc
    - ..,etc

Les cas d’utilisation

En complément, **des cas d’utilisation (use cases) ont été rédigés** pour chaque user story. Ces use cases sont accompagnés de **scénarios détaillés**, décrivant les interactions possibles entre l’utilisateur et le système, en précisant les conditions normales et les exceptions.



Exemple d'un use case

La validation

Enfin, une **validation des besoins a été effectuée avec le donneur d’ordre**, au fur et à mesure, garantissant que la documentation produite correspond aux attentes et aux objectifs du projet.

Charges techniques

Les contraintes du projet

Les contraintes du projet m'ont été dictées par la configuration du serveur qui devra hébergé l'application :

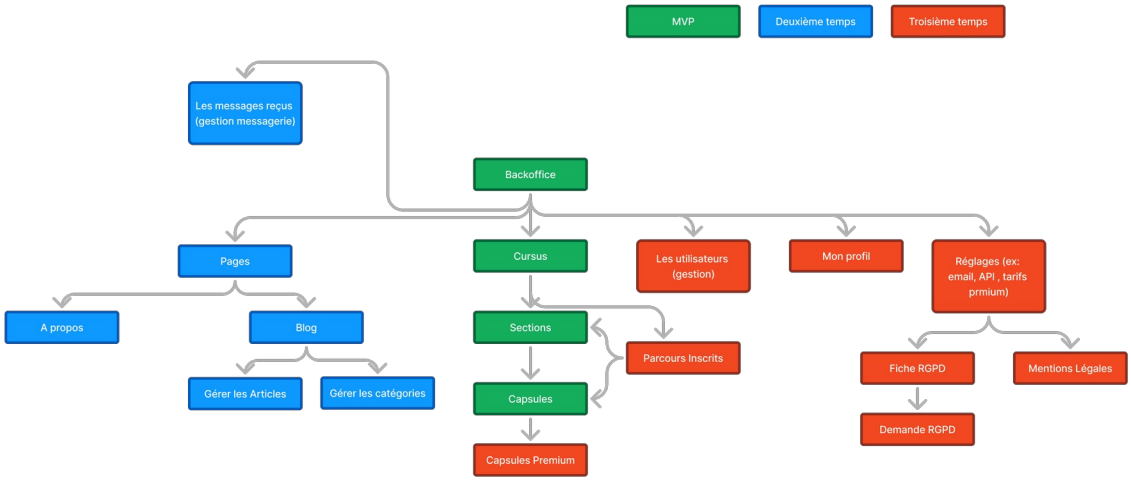
- Debian 12 (sans docker)
- Apache 2.4.62
- PHP 8,3
- 10.11.6-MariaDB
- Composer / GIT ...etc

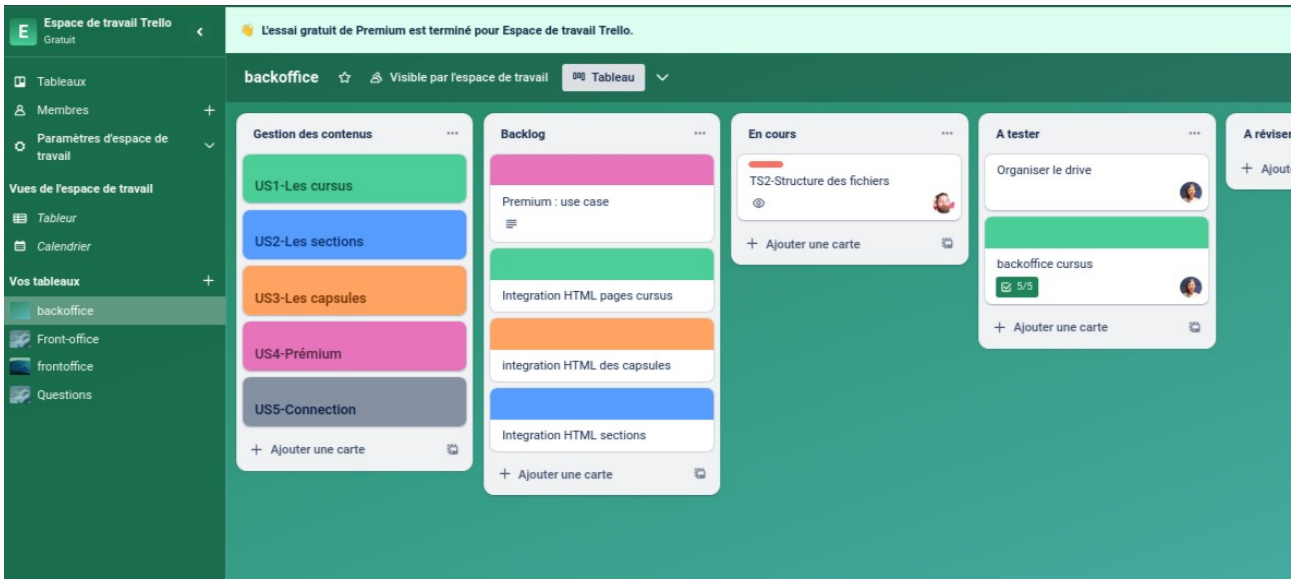
En bref, un serveur LAMP classique

Les livrables attendus

Vu le nombres de user stories important, nous avons déterminé un MVP

J'ai découpé ce MVP en tâches géré par un Kanban sous trello





Kanban début de projet

A ce jour, je n'ai pas totalement terminé le MVP (2-3 semaines manquantes)

L'environnement humain et technique

J'ai en autonomie sur ce projet mais avec des validations périodique de mon donneur d'ordre.
Ce dernier me proposait parfois quelques pistes quand j'étais un peu bloqué

Pour cette période de stage, j'ai eu l'occasion de travailler sur un PC nu, avec seulement le système d'exploitation d'installé

……

…….

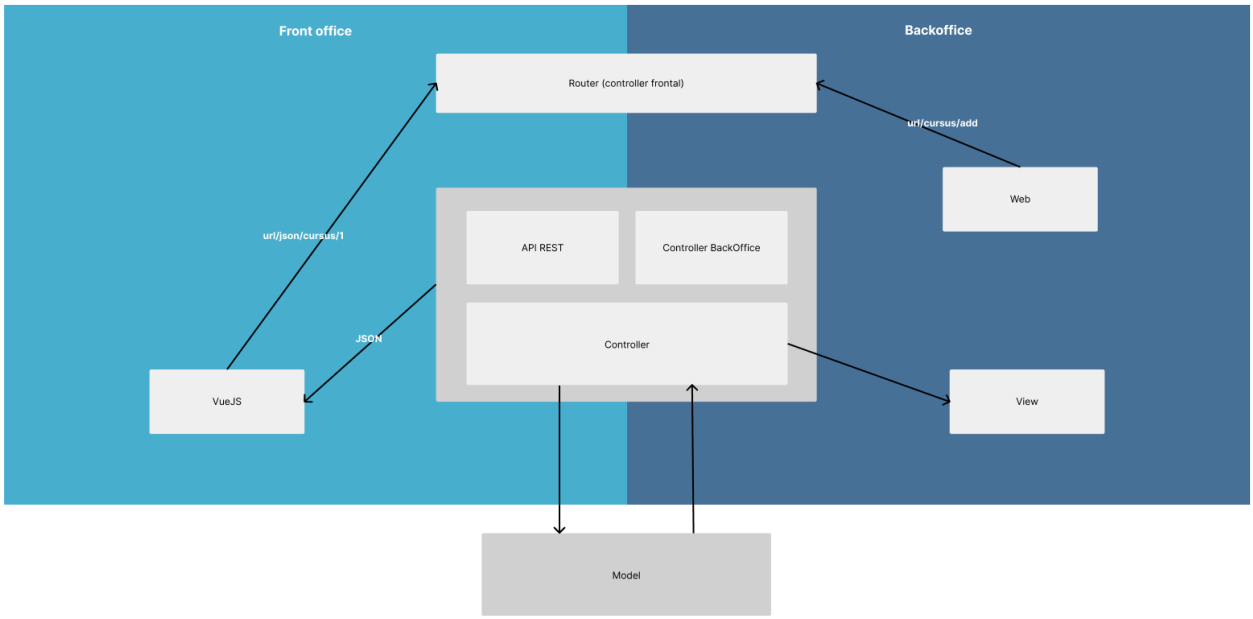
Avant d'installer ces logiciels, je me suis formé à Debian, serveur qui hébergera l'application

Réalisations

Maquettage

Architecture

Vue d’ensemble



Koabana.fr repose sur une architecture multi-couches réparties, garantissant une séparation claire des responsabilités entre le front-office et le back-office, tout en facilitant l’évolution du projet et la maintenabilité du code.

Front-Office (Vue.js)

Le front-office est développé avec Vue.js et assure l’interface utilisateur pour les apprenants et abonnés. Il communique avec le back-office via des requêtes API REST en JSON. Voici son fonctionnement :

- Affichage dynamique des cursus : L’application front consomme des données via des requêtes comme `url/json/cursus/1`, qui retourne les informations d’un cursus spécifique.
- Expérience utilisateur fluide : Vue.js permet une navigation rapide et interactive en mode Single Page Application (SPA).

Back-Office (Développement from scratch)

Le back-office est entièrement développé from scratch en PHP, sans framework. Il gère les différentes entités du projet et assure l'administration du contenu. Son architecture repose sur le principe MVC. J’ai choisi de développer l’application en pur PHP dans un soucis pédagogique afin de comprendre ce langage en profondeur tout en me calquant sur les bonnes pratiques des Framework

- Router (Contrôleur Frontal) : Il oriente les requêtes HTTP entrantes vers les bons contrôleurs en fonction des URLs demandées.

- API REST : Un ensemble d’API REST renvoie des réponses JSON au front-office.
- Contrôleur BackOffice : Il traite les requêtes liées à la gestion des contenus (cursus, vidéos, utilisateurs...).
- Modèle (Gestion des données) : Il gère la persistance des données en base de données et applique les règles métiers définies.
- Vue (Web) : Génération des pages administratives accessibles via des URLs comme url/cursus/add.

Communication et Interactions

L’architecture repose sur une communication fluide entre les différentes couches :

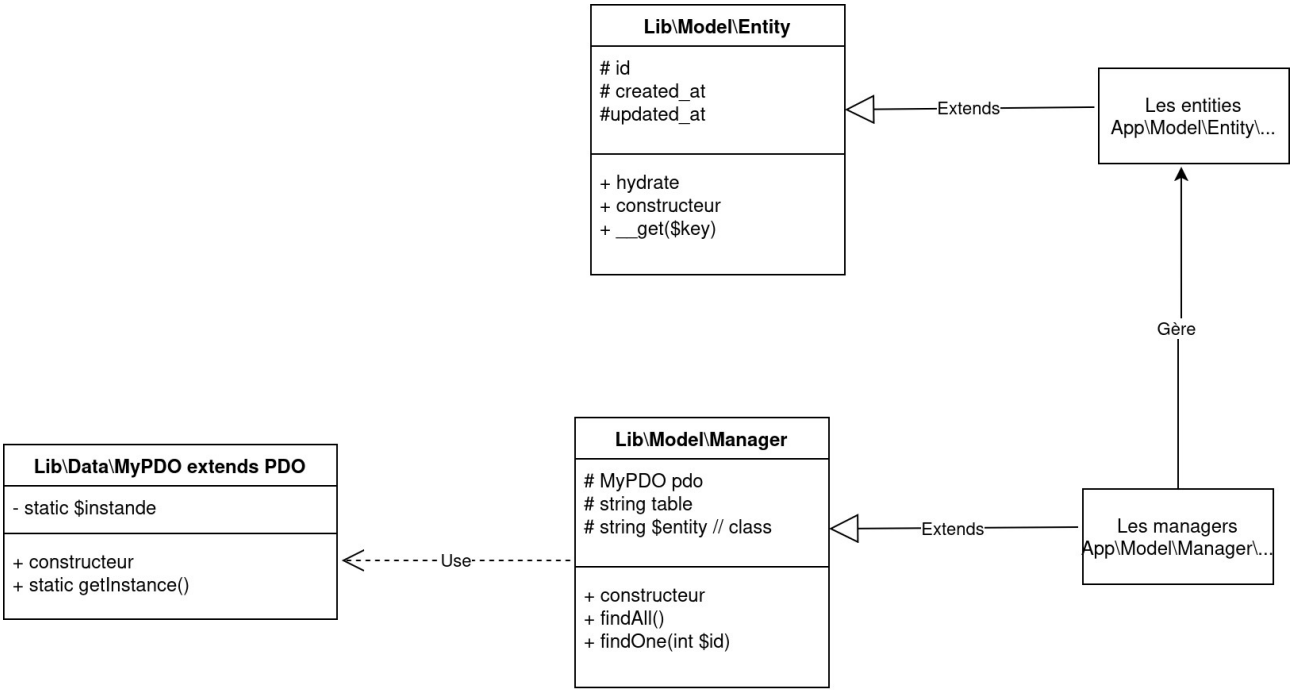
- Vue.js envoie des requêtes API au back-office pour récupérer des données.
- Le back-office traite ces requêtes via son système de routing et de contrôleurs.
- Le modèle interagit avec la base de données pour stocker et récupérer les informations.
- Les réponses sont retournées sous forme de JSON pour le front-office, ou sous forme de vues HTML pour le back-office.

Pourquoi cette architecture ?

- Modularité : Chaque couche est indépendante et peut être améliorée séparément.
- Scalabilité : L’architecture permet d’ajouter facilement de nouvelles fonctionnalités.
- Optimisation des performances : Le choix d’un développement from scratch en PHP garantit un code léger et maîtrisé.

Grâce à cette architecture, Koabana.fr offre une solution technique robuste et évolutive, adaptée aux besoins des formateurs et des apprenants.

Le modèle



L'architecture du modèle de données de Koabana.fr repose sur une structure modulaire et extensible, permettant une gestion efficace des entités et de la base de données. Elle suit une approche inspirée des ORM, tout en restant légère et optimisée pour les besoins spécifiques du projet.

Principe général

L'architecture est organisée en trois composants principaux :

- Les entités (Entity) : Représentation des objets métier (Cursus, Formation, etc.).
- Les managers (Manager) : Gestion des requêtes et interactions avec la base de données.
- Le gestionnaire de connexion (MyPDO) : Singleton partiel, centralisant l'accès à la base.

Cette organisation assure une bonne séparation des responsabilités, garantissant maintenabilité et évolutivité.

Gestion des Entités

Les entités représentent les données sous forme d'objets PHP. Une classe mère définit :

- Des propriétés communes : id, created_at, updated_at, permettant un suivi standardisé des objets.
- Une méthode hydrate() : Remplit automatiquement les attributs à partir d'un tableau associatif.
- Un getter magique (__get($key)) : Facilite l'accès aux propriétés dynamiquement.

Les entités filles (CursusEntity, FormationEntity) héritent de cette base pour ajouter leurs propres attributs et méthodes.

Les managers
Les managers sont responsables de la communication avec la base de données. Ils exécutent les requêtes SQL et renvoient des objets entité.

La classe mère implémente ces propriétés :

- MyPDO $pdo → Instance de la base de données.
- string $table → Nom de la table associée.
- Entity $entity → Classe associée à l'entité.

Ainsi que ces méthodes

- findAll() → Récupère tous les enregistrements de la table.
- findOne(int $id) → Récupère un enregistrement par son identifiant.

Les managers spécialisés héritent de Lib\Model\Manager, tout en ajoutant leurs propres méthodes métiers spécifiques.

Gestion de la Connexion à la Base de Données

Le gestionnaire de base de données s'inspire du design pattern Singleton, bien qu'il ne soit pas totalement un singleton, car son constructeur reste public.

- Stockage de l'instance via static $instance.
- Méthode getInstance() → Retourne l'instance unique ou l'instancie.

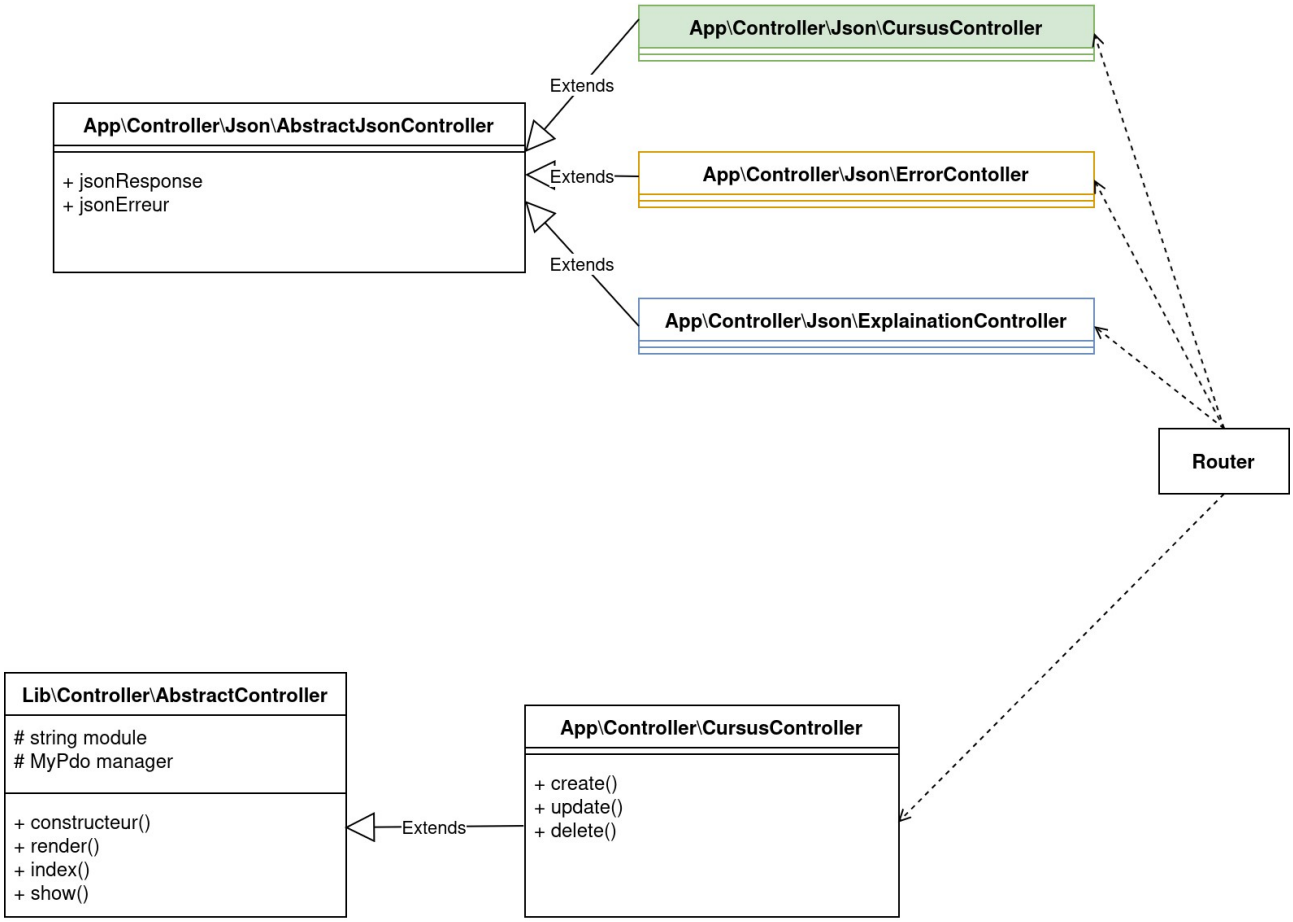
Limitation identifiée

La connexion à la base de données est réalisée avec des paramètres fournis par une constante (App:dotenv).
Il faudra à terme ajouter une injection de dépendance.

Conclusion

- Approche légère, inspirée des ORM, mais sans les lourdeurs de Doctrine.
- Séparation stricte des responsabilités (connexion, entités, gestionnaires).
- Extensibilité : Ajout de nouvelles entités et managers possible sans modifier la structure de base.

Les contrôleurs



Dans le cadre du projet Koabana.fr, nous avons mis en place une architecture de contrôleurs basée sur une séparation claire entre la gestion des vues du backoffice et la gestion des API REST.

Hiérarchie et Organisation des Contrôleurs

L'architecture repose sur deux grandes familles de contrôleurs :

- Les contrôleurs MVC classiques, qui gèrent l'affichage des vues et l'exécution des actions en interaction avec les modèles.
- Les contrôleurs REST (JSON), destinés à la communication avec le front-office Vue.js via des API.

Le Router et la Gestion des Routes

L'application repose sur un Router basé sur AltoRouter, un micro-routeur léger en PHP. Il permet d'aiguiller les requêtes vers les bons contrôleurs et d'assurer une séparation entre MVC et API REST.

Deux formats principaux sont utilisés pour la gestion des requêtes :

- API REST : http(s)://url-dynamique/json/controller/parametre
- MVC classique : http(s)://url-dynamique/controller/action/parametre

Le Router est responsable du dispatching des requêtes vers les bons contrôleurs :

- App\Controller\Json\CursusController (Gestion des cursus en JSON)
- App\Controller\Json\ErrorController (Gestion centralisée des erreurs)
- *App\Controller\Json\ExplainationController (Explications et informations en JSON)
- App\Controller\CursusController (Gestion des cursus en MVC)

AbstractController : Base des contrôleurs MVC

Le contrôleur Lib\Controller\AbstractController est la classe mère des contrôleurs MVC traditionnels. Il fournit :

- Un accès centralisé au Model.
- Une méthode render(), qui génère l'affichage des vues associées.
- Des méthodes comme index() et show() pour gérer les opérations CRUD.

AbstractJsonController : Base des contrôleurs REST

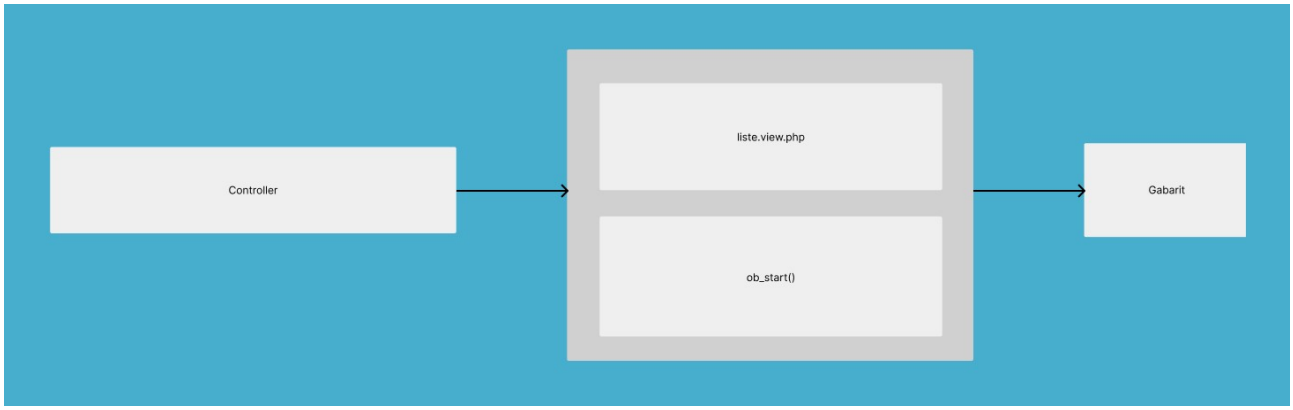
Les contrôleurs REST héritent de AbstractJsonController. Cette classe mère propose :

- jsonResponse() : Permet d'envoyer des réponses JSON structurées.
- jsonErreur() : Gère les erreurs métier et système de manière centralisée.

Depuis cette classe, plusieurs contrôleurs REST sont dérivés :

- Json\CursusController : Expose des API liées aux cursus (/json/cursus/1).
- Json\ErrorController : Centralise la gestion des erreurs.
- Json\ExplainationController : Fournit des explications liées aux cursus.

Les vues



La gestion des vues est basée sur une architecture utilisant un gabarit (template) pour assurer une structure uniforme et modulaire des pages.

1. Rôle du Contrôleur

Le contrôleur détermine la vue à afficher en fonction du contexte, comme une page de liste, une page de détail ou une page de mise à jour. Une fois la vue choisie, le contrôleur lui transmet les données nécessaires.

2. Chargement de la Vue Spécifique

La vue spécifique correspond au contenu dynamique de la page. Selon le contexte, il peut s'agir d'une liste d'éléments, d'un formulaire ou d'autres informations à afficher. Ce fichier contient uniquement la structure propre à ce contenu, sans inclure les éléments communs tels que l'en-tête, le menu ou le pied de page.

Avant d'être intégrée au gabarit, la vue est mise en mémoire tampon `ob_start()` pour éviter l'affichage immédiat. Cela permet de récupérer son contenu et de l'insérer proprement dans la structure principale.

3. Intégration dans le Gabarit

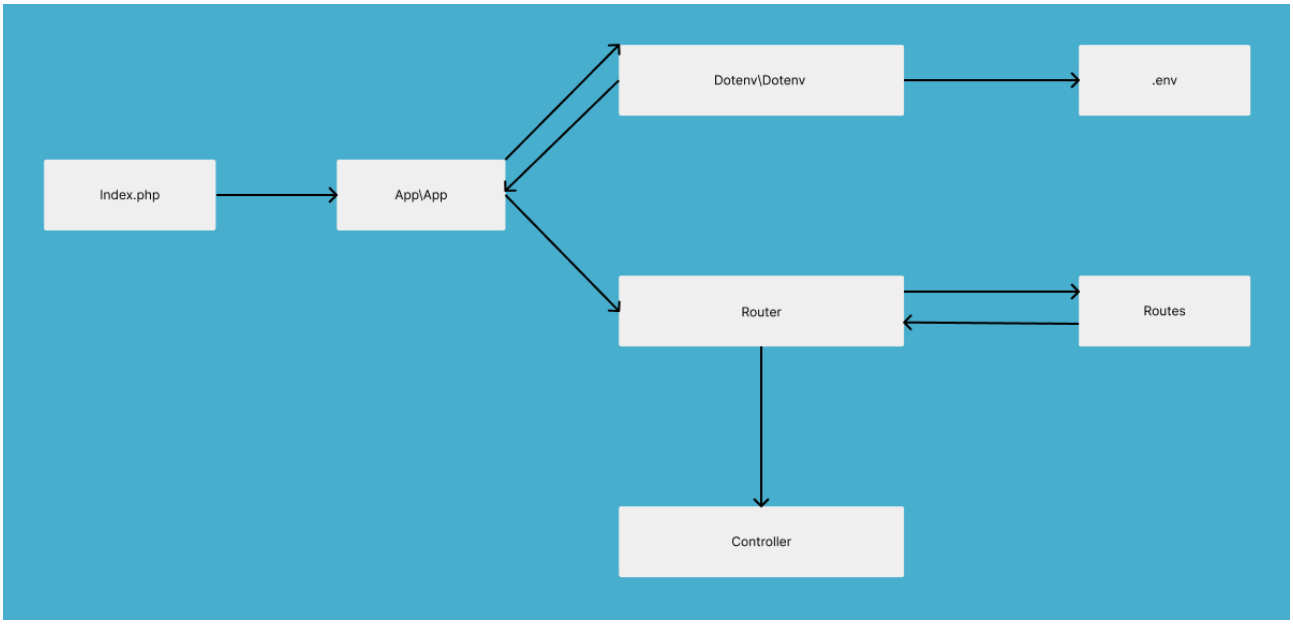
Le gabarit est le fichier principal qui définit la mise en page globale du site. Il contient les éléments communs à toutes les pages, comme l'en-tête avec la barre de navigation, le pied de page et les feuilles de style. Le contenu de la vue spécifique est ensuite injecté dans la zone dédiée du gabarit pour assurer une cohérence visuelle sur l'ensemble du site. `(ob_get_clean())`

Objectifs et Avantages

- Modularité : Les vues spécifiques sont indépendantes et facilement réutilisables.
- Cohérence visuelle : L'ensemble du site conserve une mise en page homogène.
- Facilité de maintenance : Toute modification de la structure globale du site peut être effectuée directement dans le gabarit, sans impacter chaque vue individuellement.

Cette approche assure une gestion efficace et évolutive des interfaces, en garantissant une clarté et une organisation optimales du code.
Dans un temps futur, il faudra passer par un moteur de template (Twig, Blade, ...)

Initialisation de l’application



Point d’entrée de l’application

`index.php` est le fichier principal qui sert de point d’entrée à l’application. Ses responsabilités incluent :

- Inclure le fichier d’autoloading (généralement autoload.php de Composer).
- Instancier App\App, qui initialise les composants principaux (routeur, gestion des variables d’environnement, base de données, etc.).

Ce fichier est placé dans un dossier public pour des raisons de sécurité

La classe App\App

App\App est chargé de configurer et initialiser les composants essentiels de l’application, notamment :

- Le chargement des variables d’environnement depuis .env, pour la configuration de la base de données, des clés API, des paramètres de débogage, etc.
- L’instanciation du Router et le chargement des routes (Routes), pour gérer le routage de l’application.
- L’exécution des routines de démarrage, comme la gestion des erreurs ou des sessions.

Gestion des variables d’environnement

- `Dotenv\Dotenv` est utilisé pour lire et charger les variables d’environnement depuis le fichier .env.
- Ces variables stockent des informations sensibles et dynamiques, telles que :

    - Les identifiants de connexion à la base de données (hôte, utilisateur, mot de passe).
    - Les paramètres liés aux API (clés, URL des services externes).
    - Le mode de débogage (activé ou désactivé).
- App\App utilise ces configurations pour assurer la flexibilité et la sécurité du projet.

Gestion du routage

- Router est responsable de l'analyse des requêtes entrantes et de leur correspondance avec une route définie.
- Routes contient une liste de toutes les routes disponibles, généralement sous forme de correspondance :
    - /users → UserController@index
    - /posts/{id} → PostController@show
- Une fois la route correspondante trouvée, Router transmet la requête au Controller approprié.

Avantages de cette architecture

- ✔ Modularité : Chaque composant est indépendant, facilitant l'évolution et la maintenance.
- ✔ Séparation des responsabilités : Gestion des variables, routage et logique métier sont bien distincts.
- ✔ Sécurité accrue : Utilisation de .env pour les configurations sensibles, évitant leur exposition dans le code.
- ✔ Flexibilité : Routes permet une gestion centralisée des chemins d'accès et Router assure une navigation fluide.

Logiciels tierces

Dans le cadre du développement de **Koabana**, nous avons choisi d'utiliser **Composer** comme gestionnaire de dépendances pour notre projet en **PHP Vanilla**. Bien que notre application ne repose pas sur un framework comme Symfony ou Laravel, Composer nous permet d'intégrer et de gérer facilement des bibliothèques tierces essentielles au bon fonctionnement du projet.

L'intégration de Composer nous apporte plusieurs avantages :

- **Gestion centralisée des bibliothèques**
- **Autoloading optimisé**
- **Mises à jour et sécurité**

Le fichier `composer.json` définit l'ensemble des dépendances nécessaires à **Koabana**, facilitant ainsi le déploiement et la reproductibilité du projet sur différents environnements de développement et de production.

```
{
    "name":
    "autoload": {
        "psr-4": {
            "App\\": "src/",
            "Lib\\": "lib/"
        }
    },
    "authors": [
        {
            "name":
            "email":
        }
    ],
    "require-dev": {
        "symfony/var-dumper": "^7.2",
        "phpunit/phpunit": "^11.5"
    },
    "require": {
        "phpmailer/phpmailer": "^6.9",
        "altorouter/altorouter": "^2.0",
        "php": ">=8.0",
        "vlucas/phpdotenv": "^5.6",
        "tinymce/tinymce": "^7.6",
        "verot/class.upload.php": "^2.1"
    }
}
```

- **var-dumper** → Affichage lisible et détaillé des variables pour le débogage.
- **phpunit** → Framework de tests unitaires automatisés pour PHP.
- **PHPMailer** → Envoi d’e-mails avec support SMTP, pièces jointes et authentification.
- **AltoRouter** → Gestion des routes et URLs en PHP sans framework.
- **phpdotenv** → Chargement des variables d’environnement depuis un fichier `.env`.
- **TinyMCE** → Éditeur de texte enrichi WYSIWYG pour les formulaires web.
- **class.upload.php** → Gestion avancée de l’upload et du redimensionnement d’images.

Le front-office

Backoffice

Frontoffice

La persistance

Conception et Modélisation

Réalisation

Le back-end

MVC

Les entités et les manager

Les contrôleurs

Les vues

Le backoffice

Le frontoffice

Vuejs

API Rest

Les composants

Vue Routeur

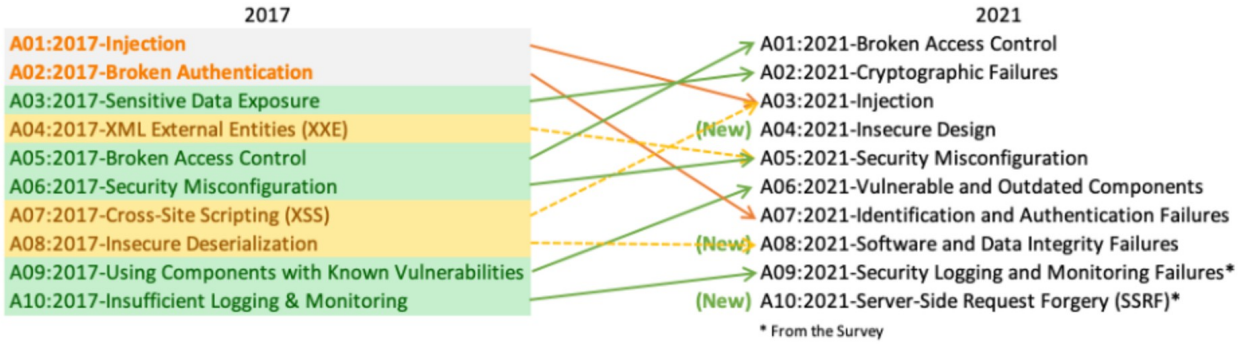
La sécurité

La veille

Owasp

OWASP (Open Worldwide Application Security Project) est une communauté en ligne dédiée à la sécurité des applications Web, totalement gratuite et ouverte à tous. Son objectif est de fournir des recommandations en matière de sécurité Web et d'aider les particuliers, les administrateurs et les entreprises à évaluer et améliorer la sécurité de leurs applications. OWASP propose également des méthodes et des outils pratiques pour faciliter l'analyse et l'optimisation du niveau de sécurité des applications Web.

Le classement **OWASP Top 10** est l'un de ses projets les plus connus. Il liste les dix principales failles de sécurité des applications web et est mis à jour tous les quelques années.



OWASP Top 10 - Principales vulnérabilités
La dernière version de l'OWASP Top 10 (2021) inclut :

- A01:2021 - Contrôle d'accès défaillant (Broken Access Control)
- A02:2021 - Échec de cryptographie (Cryptographic Failures)
- A03:2021 - Injection (SQL Injection, Command Injection, etc.)
- A04:2021 - Conception non sécurisée (Insecure Design)
- A05:2021 - Mauvaise configuration de sécurité (Security Misconfiguration)
- A07:2021 - Échec d'identification et d'authentification (Identification and Authentication Failures)
- A08:2021 - Échec d'intégrité logicielle et des données (Software and Data Integrity Failures)
- A09:2021 - Échec de journalisation et de surveillance (Security Logging and Monitoring Failures)

Mesures de protection :

- Adopter des pratiques de codage sécurisées (*Adopter des pratiques de programmation sécurisée*)
- Utiliser un pare-feu applicatif web (WAF - Web Application Firewall) (*Utiliser un pare-feu d'application web (WAF)*)
- Effectuer des tests d'intrusion pour identifier et corriger les vulnérabilités (*Réaliser des tests d'intrusion pour détecter et corriger les failles de sécurité*)

https://owasp.org/www-project-top-ten/

Faille-xss(Cross-site Scripting)

Qu’est-ce qu’une faille XSS ?

Les XSS sont un type de faille de sécurité, comme les injections SQL. Elles permettent à un hacker d’injecter du code malveillant dans une application web via des champs de saisie ou des URL.

Le but ? Pirater le site pour voler des cookies, des sessions, modifier des données ou même installer un virus. Les attaques XSS peuvent avoir des conséquences graves et donner un contrôle presque total sur le navigateur de la victime.

Comment identifier et exploiter une faille XSS ?

Pour trouver une faille XSS, il faut tester les champs de saisie d'un site (comme un formulaire ou une barre de recherche) en y mettant du code JavaScript malveillant.

Si le site affiche ce code sans le bloquer, alors il est vulnérable. Un hacker peut alors l'exploiter pour voler des informations ou modifier la page.

Quels sont les types d'attaques XSS ?

Il existe trois types d'attaques XSS :

- **XSS stocké** : Le code malveillant est enregistré dans la base de données du site et s'exécute à chaque fois qu'un utilisateur visite la page.
- **XSS réfléchi** : Le code est injecté dans une URL et s'exécute quand la victime clique sur le lien.
- **XSS DOM** : Le code est exécuté directement dans le navigateur en manipulant la page web via JavaScript.

Comment se protéger des failles XSS ?

- **Encoder les données (échappement HTML)** → Transformer les caractères spéciaux (`< > " ' &`) pour éviter que du code malveillant soit exécuté. Par exemple, `<` devient `<`, empêchant l'injection de scripts.
- **Filtrer les données côté client** → Supprimer les éléments dangereux comme `<script>` ou les attributs JavaScript (`onClick`, `onError`, etc.) pour éviter l'injection de code.
- **Valider les entrées utilisateurs** → Vérifier que les données saisies correspondent au format attendu (ex : un champ téléphone ne doit contenir que des chiffres).
- **Utiliser CSP (Content Security Policy)** → Bloquer les scripts venant de sources externes non autorisées pour limiter l'exécution de code malveillant.

https://www.vaadata.com/blog/fr/failles-xss-principes-types-dattaques-exploitations-et-bonnes-pratiques-securite/

Injection SQL(SQLi)

Les injections SQL (SQLi) sont une faille de sécurité qui permet à un attaquant de manipuler une base de données en insérant du code SQL malveillant dans un formulaire ou une URL. Cela peut lui permettre de :

- Voler des données sensibles (mots de passe, informations bancaires).
- Contourner l'authentification (se connecter sans mot de passe).
- Modifier ou supprimer des données dans la base.

- Prendre le contrôle du serveur en lisant/écrivant des fichiers.

Comment fonctionne une injection SQL ?

L’attaquant insère du code SQL spécial (ex : OR 1=1 --) pour tromper le système et modifier une requête. Par exemple, au lieu de vérifier un mot de passe, la requête devient toujours "vraie", donnant accès sans autorisation.

Comment se protéger des SQLi ?

1. Utiliser des requêtes préparées → Empêcher l’injection de code en séparant les données et la requête SQL.
2. Limiter les privilèges des utilisateurs → Un compte avec peu de permissions réduit les risques.
3. Filtrer et valider les entrées utilisateurs → Vérifier que les données entrées sont bien du bon type (ex : un champ "âge" ne doit accepter que des chiffres).
4. Désactiver les messages d’erreur détaillés → Éviter que les attaquants voient des indices sur la base de données.
5. Utiliser un pare-feu WAF → Bloquer les tentatives d’injection en analysant les requêtes.

https://www.vaadata.com/blog/fr/injections-sql-principes-impacts-exploitations-bonnes-pratiques-securite/

Faille CSRF(Cross-Site Request Forgery)

Les attaques CSRF permettent à un attaquant d’exécuter des actions non autorisées à la place d’un utilisateur sur un site web, sans qu’il s’en rende compte. Cela peut conduire à :

- Changer un mot de passe ou des informations personnelles
- Modifier les paramètres d’un compte
- Effectuer des actions avec des droits administrateur (si la victime est admin)

L’attaque fonctionne généralement en incitant l’utilisateur à cliquer sur un lien ou à visiter un site malveillant qui envoie une requête frauduleuse au site légitime.

Conditions nécessaires pour une attaque CSRF

1. Authentification basée uniquement sur les cookies → Le site utilise uniquement les cookies pour identifier l’utilisateur, sans autre protection.
2. Requête avec des paramètres prévisibles → L’attaquant peut deviner ou manipuler les paramètres de la requête.
3. Une action intéressante à exploiter → Comme la modification des paramètres de compte ou une action critique.

Comment se protéger contre CSRF ?

1. Utiliser des jetons CSRF → Un code unique est généré et vérifié à chaque requête, rendant l'attaque presque impossible.
2. Configurer l'attribut SameSite des cookies → Empêcher l'envoi des cookies lors de requêtes provenant d'autres sites.
    - SameSite=Strict → Bloque toutes les requêtes intersites.
    - SameSite=Lax → Autorise certaines requêtes GET.
3. Utiliser les protections intégrées des frameworks → Django (Python), Laravel (PHP) et d'autres frameworks intègrent une protection CSRF automatique.
4. Ne pas s'appuyer uniquement sur les cookies pour l'authentification → Ajouter d'autres mesures de sécurité comme des jetons d'accès.

Les navigateurs modernes et CSRF

Depuis 2020, Chrome, Firefox et d'autres navigateurs définissent par défaut les cookies en SameSite=Lax, ce qui limite les attaques CSRF. Cependant, les développeurs doivent toujours appliquer de bonnes pratiques pour assurer une protection optimale.

Conclusion

Les attaques CSRF exploitent la relation de confiance entre un utilisateur et un site web pour exécuter des actions non désirées. Pour s'en protéger, il est essentiel d'utiliser des jetons CSRF, de configurer les cookies avec SameSite et d'utiliser les protections des frameworks modernes. Ces bonnes pratiques permettent de sécuriser efficacement les applications contre les attaques CSRF.

ClassUpload

FileReader

Perspectives

Perspectives professionnelles

Perspectives du projet

Remerciements