Documentation Technique - Projet Système De Gestion De Blog ISI1

Lo Brutto Mathys 12511881
Dumas Yvan 12302033

I - Architecture générale **2**
II - Modèles **3**
III - Contrôleurs **4**
IV - Vues **5**
V - Design Pattern : Singleton **5**

Ce projet, réalisé dans le cadre du cours ISI1 a pour objectif la création d'un blog en PHP (sans framework lourd, structure MVC manuelle) utilisant Twig pour le moteur de template, Bootstrap 5 pour le design et Alpine.js. Il gère l'authentification, les articles, les commentaires et une administration complète.

I - Architecture générale

Ce blog a été construit selon le modèle MVC (Modèle-Vue-Contrôleur) afin de garantir une séparation claire et organisée des responsabilités ainsi qu'une meilleure maintenabilité. L'architecture est la suivante:

```
/BLOG-PROJET-ISI1
├── app/
│   ├── controllers/
│   ├── models/
│   ├── views/
│   └── logger.php
├── config/
│   ├── database.php
├── logs/
│   ├── app/
│   ├── error/
│   └── security/
├── public/
│   ├──css/
│   ├──fonts/
│   ├──image/
│   ├──js/
│   └── index.php
├── bdd.sql
└── README.md
```

Représentation de l'arborescence de notre projet

Dossier app/ :

Ce dossier contient l'ensemble de la logique applicative

- controllers/
  Contient les contrôleurs responsables du contrôle d'accès (rôles et permissions) et de l'orchestration entre les modèles et les vues.
- models/
  Contient les modèles qui encapsulent les requêtes SQL via PDO et assurent la cohérence des données (articles, commentaires, utilisateurs, tags).
- views/
  Contient les vues Twig chargées de l'affichage.

- logger.php
  Service applicatif permettant d'enregistrer des logs des événements importants du système (création, modification, suppression, erreurs).
  Il est utilisé par les contrôleurs et les modèles.

Dossier config/ :

Contient un fichier database.php, permettant ainsi de configurer l'accès à la base de données.

Dossier logs/ :

Contient les fichiers de logs du système, organisés par type (app, error, security).

Dossier public/ :

Contient les ressources accessibles publiquement

- fichiers css, javascript, images et polices.
- index.php le router.

bdd.sql:

Contient le script de création pour la base de donnée.

Le fichier index.php agit comme un routeur central : il intercepte toutes les requêtes HTTP entrantes, analyse l'URI puis appelle la méthode appropriée du contrôleur correspondant.

II - Modèles

Dans notre architecture MVC, les Modèles sont la couche responsable de la logique des données. Ils encapsulent toutes les interactions avec la base de données MySQL.

Gestion des données et requêtes SQL via PDO :

Connexion centralisée : Les modèles (`Blog.php`, `Admin.php`) n'instancient pas eux-mêmes la connexion. Ils récupèrent l'instance unique via le Singleton `Database::getInstance()->getConnection()`.

Sécurité : Pour sécuriser l'application contre les injections SQL, toutes les requêtes impliquant des données variables utilisent `PDO::prepare()` et `execute()`.
Exemple : `SELECT * FROM utilisateurs WHERE email = :email`.

Mode d'erreur : PDO est configuré en `ERRMODE_EXCEPTION`, ce qui permet aux modèles de capturer les erreurs SQL via des blocs `try/catch` (notamment pour les contraintes d'unicité sur les slugs ou emails).

Encapsulation des opérations CRUD :

Le projet sépare les responsabilités en deux modèles principaux :

Modèle Blog (Front-End) :

Lecture : getAllArticles(), getArticleBySlug() pour l'affichage public.
Écriture : createComment() pour poster un commentaire, createUser() pour l'inscription.
Gestion "Mes Articles" : Méthodes CRUD limitées aux articles de l'utilisateur connecté.

Modèle Admin (Back-Office) :

Statistiques : getArticleCount(), getPendingCommentCount().
Modération : updateCommentStatus(), deleteArticle().
Gestion Globale : updateUserRoles(), createTag().

III - Contrôleurs

Les contrôleurs jouent un rôle central dans l'architecture MVC de notre blog. Ils servent d'intermédiaire entre les modèles et les vues. Ils traitent les actions de l'utilisateur, valident les données, vérifient les permissions, et appellent les méthodes des modèles pour manipuler les données.

Chaque contrôleur est spécialisé :

- BlogController.php : articles, commentaires, pages publiques et "Mes articles".
- AdminController.php : gestion des articles, commentaires, utilisateurs et tags.
- AuthController.php : gestion de l'authentification (inscription, connexion, déconnexion)

Le contenu des articles est stocké en Markdown dans la base de données. Lors de l'affichage, le contrôleur le convertit en HTML via League\CommonMark avant de le transmettre aux vues. Cela garantit un rendu formaté tout en préservant la simplicité du stockage et la sécurité de l'application.

Les contrôleurs ne contiennent pas de logique métier ou de requêtes SQL ; ils orchestrent simplement les opérations et transmettent les données aux vues Twig. Ils gèrent

également la sécurité (authentification, rôles, permissions) et utilisent le Logger pour journaliser différentes actions : création, modification, suppression d'articles ou commentaires, et opérations des administrateurs.

IV - Vues

Les vues sont responsables de l'affichage des données et de l'interface utilisateur. Dans ce projet, nous utilisons Twig comme moteur de templates pour séparer clairement le HTML de la logique PHP. Elles reçoivent des données déjà traitées par les contrôleurs et les affichent en utilisant la syntaxe Twig, injectant les variables php dans le HTML.

Chaque page du blog (accueil, article, administration, formulaire) a son template dédié.

Bootstrap 5 est utilisé pour la mise en page et les styles, garantissant un affichage responsive. En complément, nous avons intégré Alpine.js, un framework JavaScript léger. Il est utilisé pour dynamiser l'interface utilisateur directement depuis le HTML, sans nécessiter de scripts lourds. Dans notre projet, il gère notamment le menu de préférences (changement de thème clair/sombre et ajustement de la police), offrant une interactivité immédiate et fluide pour l'utilisateur.

Grâce à la séparation suivant le pattern MVC, toute modification du design ou du front-end n'impacte pas la logique métier.

V - Design Pattern : Singleton

Dans ce projet, le design pattern Singleton est utilisé pour garantir qu'une seule instance d'une classe existe à un moment donné, et pour fournir un point d'accès global à cette instance. Nous l'avons utilisé à deux reprises:

- Connexion à la base de données:
  La classe Database utilise le Singleton pour créer et partager une seule connexion PDO à la base de données. Cela évite la création multiple de connexions, pouvant créer des anomalies.

- Logger:
  La classe Logger est également un Singleton afin de centraliser l'écriture des logs. Chaque contrôleur ou modèle accède à la même instance pour enregistrer les événements applicatifs, les erreurs ou les actions de sécurité.

Exemple:

```
$db = Database::getInstance()->getConnection();
$logger = Logger::getInstance();
$logger->info("Nouvel article créé", [...]);
```