Rapport module PHP

Ce document présente l'architecture technique, les choix de conception et le rapport fonctionnel de la plateforme web de génération de mosaïques (briques). Le projet est une application composite intégrant un front-office web, un moteur de traitement d'image et une simulation de gestion de commande/usine.

AISSYNE Amine, ALKHASTOV Zhabrail,
ESSAIDI Rayan, POLIN Ethan

1. Architecture Globale

Le projet repose sur une architecture modulaire hétérogène, divisée en trois briques principales:

1. **Application Web (PHP)** : Interface utilisateur (Storefront), gestion des comptes, upload d'images, panier et commandes.
2. **Moteur de Traitement (Java & C)** : Algorithmes de transformation d'images en instructions de montage (pixel art / briques) et optimisation du pavage.
3. **Backend Usine (Go)** : Simulation de la logistique, gestion bancaire et API de commande de pièces (Factory).

Diagramme de flux simplifié

Utilisateur -> Web (PHP) -> Base de Données (MySQL) <-> Traitement (Java/C) <-> Factory

2. Application Web (PHP)

2.1. Architecture Logicielle

L'application web utilise une architecture **MVC (Modèle-Vue-Contrôleur)** faite sur mesure ("Custom Framework"), sans dépendance à un framework lourd (comme Symfony ou Laravel), ce qui assure légèreté et contrôle total.

- **Point d'entrée (Routing)** :
  - Le fichier Public/index.php est le point d'entrée unique.
  - Le fichier .htaccess redirige toutes les requêtes vers index.php.
  - La classe App\Core\Main analyse l'URL pour instancier le contrôleur et la méthode correspondants (ex: /commande/index -> CommandeController::index()).
- **Structure des Dossiers** :
  - App/Core : Cœur du framework (Gestion BDD, Contrôleur parent, Routeur).
  - App/Controllers : Logique métier (ex: CartController, ImagesController).
  - App/Models : Accès aux données (ex: UsersModel, StockModel).
  - App/Views : Templates d'affichage (fichiers .php mixant HTML et PHP).
  - Public/ : Assets statiques (CSS, JS, Images) et point d'entrée.

2.2. Choix Techniques Frontend

- **CSS Modulaire** : Contrairement à une feuille de style monolithique, chaque vue possède son propre fichier CSS (ex: login_views.css, cart_views.css), ce qui facilite la maintenance et évite les conflits de styles.
- **JavaScript** : Utilisation de Vanilla JS pour des fonctionnalités spécifiques (Captcha, Drag & Drop, Rognage d'image).

2.3. Fonctionnalités Web Clés

1. **Gestion Utilisateur** : Inscription, Connexion, Récupération de mot de passe, Vérification de compte.
2. **Gestion des Images** :
    - Téléversement (ImagesController).
    - Rognage et prévisualisation (CropImagesController, JS/crop_images.js).
3. **E-commerce** :
    - Gestion du panier (CartController).
    - Passage de commande et Facturation (CommandeController, Invoice_views).
    - Simulation de paiement.
4. **Administration** :
    - Vue des stocks et fournisseurs (AdminController, Stock_views).
    - Statistiques (AdminStats).

3. Sécurité (robustesse)

1. **MVC** : Isolation de la logique métier et de l'affichage.
2. **Protection Injection SQL** : Utilisation de requêtes préparées dans les modèles (App/Core/Model.php).
3. **Mots de passe** : Hachage (password_hash en PHP).
4. **Captcha** : Implémentation d'un captcha maison en JS (Public/JS/captcha.js) pour protéger les formulaires.
5. **Preuve de travail (PoW)** : Sécurité avancée sur les commandes via le backend Go.