CAHIER DES CHARGES - PROJET CAMPUSPAY

Sommaire

1. Introduction

1.1 Contexte

Dans le cadre d ' un stage de qualification professionnelle (DQP), le projet **CampusPay** a été développé par Aubin Boris Simo Tsebo pour répondre aux besoins de gestion centraliée des tudiants et des paiements scolaires. Destinée aux écoles, universités et centres de formation, cette plateforme web offre une interface intuitive avec des tableaux de bord dynamiques, des rapports automatisés et un suivi en

temps réel, facilitant ainsi les processus administratifs et financiers des établissements éducatifs.

1.2 Objectifs

- Automatiser la gestion des données relatives aux étudiants, frais, filières, spécialités et niveaux.
- Proposer une interface utilisateur sécurisée, responsive et adaptée à différents rôles (administrateurs, comptables).
- Permettre un suivi en temps réel des paiements et des informations des étudiants.
- Offrir des fonctionnalités de recherche et de filtrage avancées pour une gestion efficace.
- Garantir une plateforme évolutive, facilitant l ' ajout de nouvelles fonctionnalités ou modules.
- Satisfaire les exigences pédagogiques du stage DQP tout en proposant une solution professionnelle.

2. Description du projet

2.1 Présentation générale

CampusPay est une application web développée avec le framework Laravel, conçue pour centraliser et simplifier la gestion administrative et financière des établissements éducatifs. Elle permet aux utilisateurs

autorisés de gérer les données des étudiants, des paiements et des programmes académiques via une interface moderne, sécurisée et accessible sur tous les écrans.

2.2 Public cible

- **Établissements** : Écoles, universités, centres de formation.
- **Utilisateurs** : Administrateurs scolaires, comptables, responsables académiques.
- **Besoins** : Gestion efficace des données, suivi des paiements, génération de rapports automatisés, interface intuitive et sécurisée.

3. Exigences fonctionnelles et non fonctionnelles

3.1 Exigences fonctionnelles

- **Authentification et gestion des rôles** : Système sécurisé avec rôles (administrateur, comptable) et restrictions d' accès via middleware.
- **Opérations CRUD** : Gestion complète (création, lecture, mise à jour, suppression) pour :
    - Étudiants (nom, email, sexe, année de naissance).
    - Filières, spécialités, niveaux.
    - Frais et paiements.

- **Tableau de bord** : Interface responsive avec sidebar et header adaptatifs, affichant des statistiques clés.
- **Recherche et filtrage** : Recherche avancée des étudiants (par nom, email, sexe, année) avec pagination dynamique.
- **Gestion des paiements** : Suivi des paiements avec affichage du total cumulé en bas des tableaux.
- **Pages d'erreur** : Page 404 personnalisée pour une expérience utilisateur optimale.

3.2 Exigences non fonctionnelles

- **Sécurité** : Protection des routes via middleware et authentification robuste.
- **Performance** : Temps de réponse rapide pour les requêtes et chargement des pages.
- **Responsivité** : Compatibilité avec tous les appareils (mobile, tablette, desktop).
- **Évolutivité** : Architecture modulaire pour l'ajout de nouveaux modules ou rôles.
- **Accessibilité** : Respect des standards d'accessibilité (ex. : contraste suffisant selon WCAG).
- **Maintenance** : Code structuré et documenté pour faciliter les mises à jour.

4. Spécifications techniques

4.1 Technologies utilisées

- **Backend** : Laravel (PHP >= 8.1), Composer pour la gestion des dépendances.
- **Frontend** : Templates Blade, Tailwind CSS pour le style, JavaScript via Node.js/npm.
- **Base de données** : MySQL/MariaDB pour le stockage des données.
- **Outils** : Middleware Laravel pour la sécurité, migrations pour la gestion des schémas.

4.2 Architecture

- **Contrôleurs** : Situés dans app/Http/Controllers/ pour gérer les fonctionnalités (utilisateurs, frais, filières, spécialités, niveaux, paiements).
- **Vues** : Templates Blade dans resources/views/ pour le tableau de bord, la gestion des étudiants, des paiements et la page 404.
- **Routes** : Définies dans routes/web.php, avec restrictions par rôle via middleware.
- **Assets** : Images (ex. : photos des étudiants) dans public/images/, styles et scripts compilés via npm.
- **Base de données** : Schéma défini dans database/migrations/ pour toutes les tables.

4.3 Charte graphique

La plateforme utilise une palette de couleurs sobre et professionnelle, basée sur Tailwind CSS :

- **Couleur principale** : Bleu foncé (#1B4F72, blue-700) pour les boutons, titres et éléments interactifs, évoquant la fiabilité.
- **Couleur secondaire** : Gris clair (#E5E7EB, gray-200) pour les fonds des sections et tableaux, assurant une lisibilité optimale.
- **Couleur d'accent** : Bleu clair (#60A5FA, blue-400) pour les interactions (survol, liens).
- **Texte** : Gris foncé (#1F2A44, gray-800) pour le contenu principal, avec un contraste élevé.
- **Page d'erreur (404)** : Fond gris clair avec un accent bleu pour une apparence moderne et non agressive.
- **Accessibilité** : Contraste respectant les normes WCAG (minimum 4.5:1).
- **Personnalisation** : Palette modifiable via tailwind.config.js pour s'adapter à une charte graphique spécifique.

5. Planning prévisionnel

5.1 Étapes clés

- **Conception (2 semaines)** : Analyse des besoins et conception de l'architecture (effectuée).

- **Développement (6 semaines)** : Implémentation des fonctionnalités CRUD, tableau de bord, gestion des paiements et sécurité (effectuée).
- **Tests (1 semaine)** : Validation des fonctionnalités, tests de sécurité et de responsivité.
- **Déploiement (1 semaine)** : Configuration du serveur et mise en production.
- **Maintenance (continue)** : Corrections de bugs et ajout de fonctionnalités futures.

Note : Les durées sont estimatives, basées sur un stage de 2 à 3 mois. Ajuster selon le calendrier réel.

6. Contraintes

- **Techniques** : Dépendance à PHP >= 8.1, MySQL/MariaDB, Node.js et Composer.
- **Sécurité** : Protection des données sensibles (ex. : informations des étudiants) via middleware et bonnes pratiques.
- **Délais** : Développement limité par la durée du stage DQP (à préciser si connue).
- **Ressources** : Projet réalisé par un seul développeur, limitant la parallélisation des tâches.
- **Responsivité** : Interface adaptée à tous les écrans, nécessitant des tests approfondis.

- **Évolutivité** : Architecture devant permettre l’ ajout de nouveaux modules ou rôles.

7. Livrables

- **Code source** : Disponible sur https://github.com/boris2442/campuspay.git.
- **Documentation** : README détaillant l’ installation, la configuration et l’ utilisation.
- **Base de données** : Schéma complet avec migrations pour toutes les tables.
- **Interface utilisateur** : Plateforme web responsive et sécurisée, incluant tableau de bord, gestion des étudiants, paiements et page d’ erreur.
- **Rapport de stage** : Documentation des travaux réalisés (si requis par le stage).

8. Conclusion

CampusPay est une plateforme robuste, sécurisée et évolutive, répondant aux besoins des établissements éducatifs en matière de gestion administrative et financière. Développée dans le cadre du stage DQP, elle allie simplicité d ’ utilisation, performance et flexibilité. Ce cahier des charges formalise les objectifs, fonctionnalités et contraintes du projet, servant de guide pour son déploiement, sa maintenance et son évolution future.