Cahier des charges simplifié — SchoolReg

Lionel YEMELE — Projet de fin d'étude (PFE)

Partie 1 : Objectif du développement de l'application

SchoolReg a pour objectif de numériser et centraliser l'inscription scolaire, en remplaçant les formulaires papier et fichiers dispersés par une application web simple. L'outil vise à réduire les erreurs, accélérer le traitement des dossiers et améliorer l'expérience des parents, élèves et administrateurs.

Partie 2 : Fonctionnalités principales

- Inscription en ligne : création de compte (parent/élève), saisie des informations, téléversement des pièces (acte de naissance, photo).
- Gestion des classes : création/édition des classes et affectation des élèves par l'administrateur.
- Suivi des paiements : frais d'inscription et reçus (intégration Stripe en mode test).
- Tableau de bord (admin) : statistiques d'inscriptions, paiements, alertes.
- Exports : génération des listes en PDF/CSV/Excel.
- Rôles et sécurité : authentification JWT, rôles ADMIN | PARENT | ÉLÈVE.
- Notifications (optionnel) : confirmations par e-mail ou rappel de paiements.
- Brouillons & validation : possibilité d'enregistrer une inscription en brouillon puis de soumettre pour validation.

Partie 3 : Technologie utilisée et justification du choix

Frontend : React + TypeScript (Vite), UI : Tailwind CSS ou Material UI.

Justification : écosystème riche, composants réutilisables, productivité élevée pour une application web moderne. Plus simple à prendre en main qu'un stack PHP/Laravel complet pour un PFE.

Backend : Python (Flask) — API REST (JSON).

Justification : micro-framework léger, rapide à prototyper, idéal pour exposer des endpoints clairs (CRUD élèves, classes, paiements).

Base de données : SQLite (développement/POC), migration possible vers PostgreSQL en production.

Justification : SQLite simplifie l'installation et les tests en local; PostgreSQL offrira la robustesse en production si nécessaire.

Auth & paiements : JWT pour l'authentification/autorisation ; Stripe (mode test) pour la gestion des paiements.

Outils & déploiement : Git/GitHub, tests unitaires (pytest), Docker (optionnel), hébergement Render/Railway (ou équivalent).

Partie 4 : Calendrier de répartition des tâches (8 semaines ≈ 240 h)

Semaine 1 (35 h) : Analyse & maquettes

• Recueil des besoins et rédaction des user stories.

• Maquettes low-fi des écrans clés (connexion, inscription, tableau de bord).

• Schéma de données initial (Élève, Classe, Paiement, Utilisateur).

Semaines 2–3 (80 h) : Fondation technique & module d'inscription

• Mise en place du repo, CI simple, base React/TS et API Flask.

• Conception/CRUD Élève + téléversement de pièces ; comptes Parent/Élève.

• Auth JWT (login/logout, rôles) ; BD SQLite configurée.

Semaines 4–5 (70 h) : Gestion des classes & paiements

• CRUD Classe et affectation des élèves.

• Intégration paiements (Stripe test) + génération de reçus.

• Sécurisation des endpoints et validations côté serveur.

Semaine 6 (35 h) : Tableau de bord & exports

• Statistiques (inscriptions par période, paiements par statut).

• Exports CSV/PDF/Excel ; filtres de recherche et pagination.

Semaine 7 (20 h) : Tests & documentation

• Tests unitaires (backend) et tests UI critiques (frontend).

• Guide d'installation, manuel utilisateur et schéma BD final.

Semaine 8 (20 h) : Déploiement & démo finale

• Déploiement (Render/Railway) en pré-production.

• Démonstration finale et collecte des retours pour itérations.

Partie 5 : Date de début et de fin du stage

Date de début : 2025-09-22

Date de fin : 2025-11-17