Mohamed Abbou
Ginf-2

RAPPORT DE PROJET

Application de Gestion des Étudiants - ENSAT

1. INTRODUCTION

1.1 Contexte et Objectifs

Ce projet consiste en la conception et le développement d'une application web moderne pour la gestion administrative des étudiants de l'ENSAT. L'objectif principal est de fournir aux administrateurs une interface intuitive et sécurisée permettant d'effectuer l'ensemble des opérations CRUD (Create, Read, Update, Delete) sur les données étudiantes.

Le système adopte une architecture de type Single Page Application (SPA), offrant une séparation claire entre le backend (API REST) et le frontend (interface utilisateur), garantissant ainsi modularité et évolutivité.

1.2 Besoins Fonctionnels

- Authentification sécurisée via OAuth 2.0 (Google Sign-In)
- Gestion complète des profils étudiants (ajout, consultation, modification, suppression)
- Interface utilisateur moderne et responsive
- Sécurisation des accès par système de tokens (Bearer Authentication)

2. ARCHITECTURE TECHNIQUE

2.1 Architecture Globale

Le projet repose sur une architecture client-serveur découplée, permettant une maintenance et une évolution indépendantes des composants.

Backend - API REST (Laravel 11)

- Framework PHP moderne suivant le pattern MVC
- Exposition des données via une API RESTful

- Gestion de la logique métier et des accès à la base de données
- Authentification sécurisée avec Laravel Sanctum

Frontend - Interface Utilisateur (Vue.js 3)

- Framework JavaScript progressif utilisant la Composition API
- Communication avec l'API via Axios
- Gestion réactive de l'état et de l'interface utilisateur
- Expérience utilisateur fluide sans rechargement de page

Services Tiers

- Firebase Authentication pour la délégation de l'authentification OAuth 2.0
- MySQL comme système de gestion de base de données relationnelle

2.2 Flux d'Authentification

Le processus d'authentification combine Firebase et Laravel Sanctum :

1. L'utilisateur initie la connexion via Google Sign-In (Frontend)
2. Firebase valide l'identité et retourne les informations de l'utilisateur
3. Le Frontend transmet ces données à l'endpoint Laravel `/api/auth/google`
4. Laravel vérifie ou crée l'utilisateur en base de données
5. Un token d'accès (Bearer Token) est généré et retourné au client
6. Ce token sécurise toutes les requêtes API ultérieures

3. CONCEPTION DE LA BASE DE DONNÉES

3.1 Modèle Relationnel

Le système utilise MySQL avec deux tables principales :

Table `users` (Administrateurs)

Champ	Type	Description
id	BIGINT (PK)	Identifiant unique
name	VARCHAR(255)	Nom complet
email	VARCHAR(255)	Adresse email (unique)
google_id	VARCHAR(255)	Identifiant Google OAuth

password	VARCHAR(255)	Hash (requis par Laravel)
created_at	TIMESTAMP	Date de création
updated_at	TIMESTAMP	Date de modification

Table `students` (Étudiants)

Champ	Type	Description
id	BIGINT (PK)	Identifiant unique
name	VARCHAR(255)	Nom complet de l'étudiant
email	VARCHAR(255)	Email institutionnel
major	VARCHAR(100)	Filière (GI, GSTR, etc.)
created_at	TIMESTAMP	Date de création
updated_at	TIMESTAMP	Date de modification

4. SPÉCIFICATION DE L'API

4.1 Endpoints Disponibles

L'API expose les routes suivantes, protégées par le middleware `auth:sanctum` :

Méthode	Endpoint	Description	Authentification
POST	`/api/auth/google`	Authentification OAuth et génération de token	Publique
POST	`/api/logout`	Déconnexion (révocation du token)	Requise
GET	`/api/students`	Récupération de tous les étudiants	Requise

POST	`/api/students`	Création d'un nouvel étudiant	Requise
PUT	`/api/students/{id}`	Mise à jour d'un étudiant	Requise
DELETE	`/api/students/{id}`	Suppression d'un étudiant	Requise

4.2 Format des Réponses

Toutes les réponses de l'API sont au format JSON. Les codes de statut HTTP suivent les standards REST :

- 200 : Succès (GET, PUT)
- 201 : Création réussie (POST)
- 204 : Suppression réussie (DELETE)
- 401 : Non authentifié
- 422 : Erreur de validation
- 500 : Erreur serveur

5. IMPLÉMENTATION

5.1 Backend (Laravel)

Contrôleur d'Authentification

- Validation des données Google reçues
- Création ou récupération de l'utilisateur via `firstOrCreate()`
- Génération et retour du token Sanctum

Contrôleur des Étudiants

- Implémentation des méthodes CRUD (index, store, update, destroy)
- Validation des données entrantes via Form Requests
- Retour des réponses JSON formatées

Middleware de Sécurité

- Protection des routes par `auth:sanctum`
- Configuration CORS pour accepter les requêtes du frontend
- Gestion des tokens dans la table `personal_access_tokens`

5.2 Frontend (Vue.js)

Composants Principaux

- Gestion de l'état avec la Composition API (ref, computed, onMounted)
- Configuration d'une instance Axios avec intercepteurs
- Formulaire de gestion avec validation côté client
- Tableau dynamique avec actions CRUD

Gestion de l'État

- Stockage du token dans localStorage pour la persistance
- Gestion réactive de la liste des étudiants
- États de chargement et gestion des erreurs

6. SÉCURITÉ ET BONNES PRATIQUES

6.1 Mesures de Sécurité Implémentées

- **Authentification OAuth 2.0** : Délégation à Google pour éviter la gestion de mots de passe
- **Tokens Sanctum** : Authentification stateless avec tokens révocables
- **Validation des données** : Validation stricte côté serveur avec Laravel Form Requests
- **Protection CSRF** : Gestion automatique par Laravel pour les requêtes sensibles
- **Hachage sécurisé** : Utilisation de bcrypt pour les données sensibles

6.2 Gestion des Erreurs

- Affichage de messages d'erreur explicites à l'utilisateur
- Logging des erreurs serveur pour faciliter le débogage
- Gestion des cas de déconnexion et de tokens expirés

7. RÉSULTATS ET DISCUSSION

7.1 Fonctionnalités Réalisées

Le système développé répond aux exigences initiales avec succès :

- Interface moderne et intuitive accessible via navigateur web
- Authentification fluide via Google Sign-In
- Opérations CRUD complètes et fonctionnelles
- Sécurisation efficace des données et des accès

7.2 Avantages de l'Architecture Choisie

- **Séparation des préoccupations** : Frontend et backend indépendants
- **Scalabilité** : Possibilité d'ajouter facilement de nouvelles fonctionnalités
- **Maintenabilité** : Code structuré suivant les conventions des frameworks
- **Performance** : SPA offrant une expérience utilisateur fluide

8. CONCLUSION ET PERSPECTIVES

8.1 Conclusion

Ce projet a permis de développer une solution complète et moderne pour la gestion des étudiants de l'ENSAT. L'utilisation de technologies actuelles (Laravel 11, Vue.js 3) garantit la pérennité du système, tandis que l'intégration de Firebase simplifie et sécurise l'authentification.

L'architecture choisie offre une base solide pour des évolutions futures, et le système est prêt pour un déploiement en production après des tests d'acceptation.

8.2 Perspectives d'Évolution

Court terme :

- Implémentation d'un système de gestion des rôles (RBAC)
- Ajout de filtres et de recherche avancée dans la liste des étudiants
- Export des données au format CSV/Excel

Moyen terme :

- Module de gestion des notes et des évaluations
- Tableau de bord statistique pour l'administration
- Système de notifications par email

Long terme :

- Application mobile (iOS/Android) avec API partagée
- Génération automatique de rapports et documents administratifs
- Intégration avec d'autres systèmes de l'école (bibliothèque, scolarité)

9. RÉFÉRENCES

Frameworks et Bibliothèques

- Laravel 11 Documentation : https://laravel.com/docs/11.x
- Vue.js 3 Documentation : https://vuejs.org/guide/
- Firebase Authentication : https://firebase.google.com/docs/auth
- Laravel Sanctum : https://laravel.com/docs/11.x/sanctum

Standards et Protocoles

- OAuth 2.0 : RFC 6749
- REST API Design : Best Practices
- HTTP Status Codes : RFC 9110