Rapport de Projet

CL0V

Réalisé par :

Daibbar Mohamed
Deroui Mouad

Encadré par :

Pr. Yasser Rochd

École Nationale des Sciences Appliquées
ENSA Khouribga

Table des matières

1 Introduction générale

De nos jours, le développement d'applications web modernes est devenu incontournable pour répondre aux besoins croissants de digitalisation. Le projet intitulé **CL0V** s'inscrit dans cette dynamique, visant à concevoir et réaliser une solution informatique complète. Ce rapport détaille l'ensemble du processus de développement, de l'analyse des besoins à l'implémentation technique.

1.1 Contexte du projet

L'évolution rapide des technologies web offre de nouvelles opportunités pour résoudre des problèmes complexes de gestion et de communication. Le projet CL0V naît d'un besoin spécifique : *faciliter la gestion des activités parascolaires de l'ENSAK.*

Ce projet a été réalisé pour mettre en pratique des compétences avancées en ingénierie logicielle, en s'appuyant sur des technologies actuelles telles que Next.js pour le frontend et une base de données relationnelle SQL.

1.2 Objectifs du projet

L'objectif principal est de développer une application web fonctionnelle. Les objectifs spécifiques du projet se déclinent comme suit :

- Concevoir le système d'information à travers les modèles conceptuel, logique et physique de données (MCD, MLD, MPD).
- Développer une interface utilisateur réactive et intuitive avec Next.js.
- Implémenter une logique métier robuste et une gestion efficace des données.
- Assurer la qualité du code et la sécurité de l'application.

2 Conception du système d'information

2.1 Modèle Conceptuel de Données (MCD)

Le Modèle Conceptuel de Données (MCD) permet de représenter de manière formelle les données qui seront utilisées dans le système d'information. Voici le schéma du MCD pour le projet CL0V :

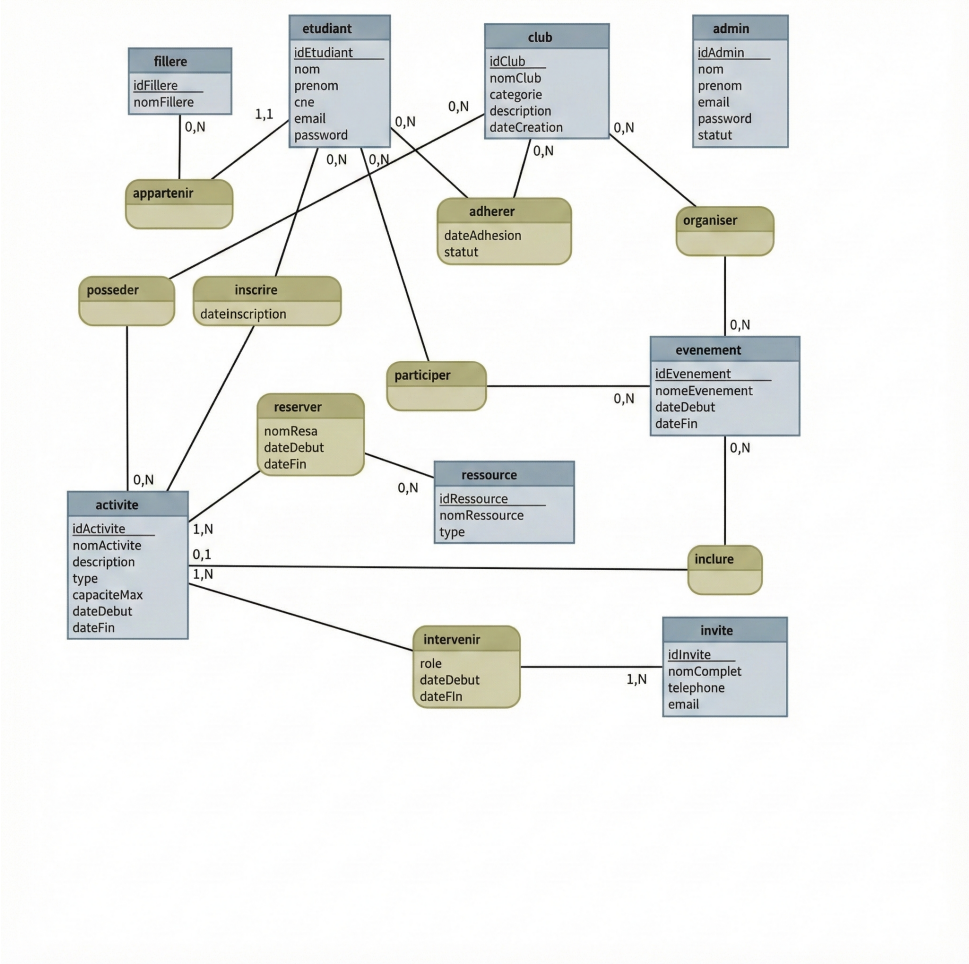

FIGURE 1 – Modèle Conceptuel de Données - Projet CL0V

2.2 Modèle Logique de Données (MLD)

Le Modèle Logique de Données suivant détaille la structure des tables, les clés primaires (soulignées) et les clés étrangères (précédées d'un #).

ADMINS (adminId, firstName, lastName, email, password, status)

MAJORS (majorId, majorName)

CLUBS (clubId, clubName, category, description, createdAt)

EVENTS (eventId, eventName, startDate, endDate)

RESOURCES (resourceId, resourceName, type)

GUESTS (guestId, fullName, phone, email)

STUDENTS (studentId, firstName, lastName, cne, email, password, *#majorId*)

ACTIVITIES (activityId, activityName, description, type, maxCapacity, startDate, endDate, *#clubId*, *#eventId*)

CLUBMEMBERSHIPS (membershipId, joinedAt, status, *#clubId*, *#studentId*)

RESERVATIONS (reservationId, reservationName, startDate, endDate, *#resourceId*, *#activityId*)

REGISTRATIONS (registrationId, registeredAt, *#studentId*, *#activityId*)

GUESTROLES (guestRoleId, roleDescription, startDate, endDate, *#guestId*, *#activityId*)

EVENTORGANIZERS (*#eventId*, *#clubId*)

EVENTPARTICIPANTS (*#studentId*, *#eventId*)

2.3 Modèle Physique de Données (MPD)

Le Modèle Physique de Données (MPD) représente l'implémentation concrète de la base de données SQLite pour le projet CL0V. Il définit les types de données, les contraintes d'intégrité et les relations entre les tables.

2.3.1 Dictionnaire de données

2.4 Implémentation Physique (MPD)

Le script SQL suivant constitue le Modèle Physique de Données. Il définit la structure exacte des tables, les types de données SQLite et les contraintes métier (Clés Primaires, Étrangères et clauses `CHECK`).

```
PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS admins (
    adminId INTEGER PRIMARY KEY AUTOINCREMENT,
    firstName TEXT NOT NULL,
    lastName TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    status TEXT CHECK(status IN ('pending', 'accepted', 'rejected')
       ) DEFAULT 'pending' NOT NULL
);

CREATE TABLE IF NOT EXISTS majors (
    majorId INTEGER PRIMARY KEY AUTOINCREMENT,
    majorName TEXT CHECK (majorName IN ('IID', 'GI', 'IRIC', 'GE',
       'GPEE', 'MGSI')) NOT NULL
);

CREATE TABLE IF NOT EXISTS clubs (
    clubId INTEGER PRIMARY KEY AUTOINCREMENT,
    clubName TEXT NOT NULL UNIQUE,
    category TEXT CHECK(category IN ('art et culture', 'sport', '
       technologie', 'entrepreneuriat social')) NOT NULL,
    description TEXT,
    createdAt DATETIME NOT NULL
);

CREATE TABLE IF NOT EXISTS events (
    eventId INTEGER PRIMARY KEY AUTOINCREMENT,
    eventName TEXT NOT NULL,
    startDate DATETIME NOT NULL,
    endDate DATETIME NOT NULL,
```

```
    CHECK (endDate >= startDate)
);

CREATE TABLE IF NOT EXISTS resources (
    resourceId INTEGER PRIMARY KEY AUTOINCREMENT,
    resourceName TEXT NOT NULL,
    type TEXT CHECK (type IN ('Salle', 'Amphitheatre', 'Buvette', '
       Bibliotheque', 'Terrain', 'Labo')) NOT NULL
);

CREATE TABLE IF NOT EXISTS students (
    studentId INTEGER PRIMARY KEY AUTOINCREMENT,
    firstName TEXT NOT NULL,
    lastName TEXT NOT NULL,
    cne TEXT UNIQUE NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    majorId INTEGER,
    FOREIGN KEY (majorId) REFERENCES majors(majorId)
);

CREATE TABLE IF NOT EXISTS activities (
    activityId INTEGER PRIMARY KEY AUTOINCREMENT,
    activityName TEXT NOT NULL,
    description TEXT,
    type TEXT CHECK(type IN ('Voyage', 'Atelier', 'Conference', '
       Competition') NOT NULL),
    maxCapacity INTEGER CHECK(maxCapacity > 0),
    startDate TIMESTAMP NOT NULL,
    endDate TIMESTAMP NOT NULL,
    clubId INTEGER,
    eventId INTEGER,

    FOREIGN KEY (clubId) REFERENCES clubs(clubId) ON DELETE CASCADE
       ,
    FOREIGN KEY (eventId) REFERENCES events(eventId) ON DELETE
       CASCADE,

    CONSTRAINT checkOwner CHECK (
        (clubId IS NULL AND eventId IS NOT NULL) OR
        (clubId IS NOT NULL AND eventId IS NULL)
    ),
    CHECK (endDate >= startDate)
);

CREATE TABLE IF NOT EXISTS clubMemberships (
    membershipId INTEGER PRIMARY KEY AUTOINCREMENT,
    joinedAt DATE DEFAULT CURRENT_DATE,
    clubId INTEGER NOT NULL,
    studentId INTEGER NOT NULL,
    status TEXT CHECK(status IN ('pending', 'accepted', 'rejected')
```

```
        ) DEFAULT 'pending' NOT NULL,
    FOREIGN KEY (clubId) REFERENCES clubs(clubId) ON DELETE CASCADE
        ,
    FOREIGN KEY (studentId) REFERENCES students(studentId) ON
        DELETE CASCADE,
    UNIQUE(clubId, studentId)
);

CREATE TABLE IF NOT EXISTS reservations (
    reservationId INTEGER PRIMARY KEY AUTOINCREMENT,
    reservationName TEXT NOT NULL,
    startDate DATETIME NOT NULL,
    endDate DATETIME NOT NULL,
    resourceId INTEGER NOT NULL,
    activityId INTEGER NOT NULL,
    FOREIGN KEY (resourceId) REFERENCES resources(resourceId) ON
        DELETE CASCADE,
    FOREIGN KEY (activityId) REFERENCES activities(activityId) ON
        DELETE CASCADE,
    CHECK (endDate >= startDate),
    CHECK (startDate >= CURRENT_DATE)
);

CREATE TABLE IF NOT EXISTS eventOrganizers (
    eventId INTEGER NOT NULL,
    clubId INTEGER NOT NULL,
    FOREIGN KEY (eventId) REFERENCES events(eventId) ON DELETE
        CASCADE,
    FOREIGN KEY (clubId) REFERENCES clubs(clubId) ON DELETE CASCADE
        ,
    UNIQUE(eventId, clubId)
);

CREATE TABLE IF NOT EXISTS eventParticipants (
    eventId INTEGER NOT NULL,
    studentId INTEGER NOT NULL,
    FOREIGN KEY (studentId) REFERENCES students(studentId) ON
        DELETE CASCADE,
    FOREIGN KEY (eventId) REFERENCES events(eventId) ON DELETE
        CASCADE,
    UNIQUE(studentId, eventId)
);

CREATE TABLE IF NOT EXISTS registrations (
    registrationId INTEGER PRIMARY KEY AUTOINCREMENT,
    registeredAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    studentId INTEGER NOT NULL,
    activityId INTEGER NOT NULL,
    FOREIGN KEY (studentId) REFERENCES students(studentId) ON
        DELETE CASCADE,
    FOREIGN KEY (activityId) REFERENCES activities(activityId) ON
```

```
        DELETE CASCADE,
    UNIQUE(studentId, activityId)
);

CREATE TABLE IF NOT EXISTS guests (
    guestId INTEGER PRIMARY KEY AUTOINCREMENT,
    fullName TEXT NOT NULL,
    phone TEXT NOT NULL,
    email TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS guestRoles (
    guestRoleId INTEGER PRIMARY KEY AUTOINCREMENT,
    guestId INTEGER NOT NULL,
    activityId INTEGER NOT NULL,
    roleDescription TEXT NOT NULL,
    startDate TIMESTAMP NOT NULL,
    endDate TIMESTAMP NOT NULL,
    FOREIGN KEY (guestId) REFERENCES guests(guestId) ON DELETE
        CASCADE,
    FOREIGN KEY (activityId) REFERENCES activities(activityId) ON
        DELETE CASCADE,
    CHECK (endDate >= startDate)
);
```

Listing 1 – Script SQL de création de la base de données CL0V

2.4.1 Contraintes d'intégrité et de performance

L'implémentation physique repose sur plusieurs piliers pour assurer la robustesse du système d'information :

- **Intégrité Référentielle :** Activation systématique des clés étrangères via `PRAGMA foreign_keys = ON` et utilisation de `ON DELETE CASCADE` pour éviter les données orphelines.
- **Validation Métier :** Utilisation de clauses `CHECK` au niveau du SGBD pour valider les domaines de valeurs (catégories, statuts) et la cohérence temporelle (dates).
- **Unicité :** Garantir qu'un étudiant ne peut pas s'inscrire deux fois à la même activité via une contrainte `UNIQUE(studentId, activityId)`.

3 Choix technologiques

3.1 Frontend (Next.js)

3.2 Backend

Pour le développement du backend, nous avons tiré profit des capacités de Next.js, notamment en utilisant le **Server Side Rendering (SSR)** pour les requêtes CRUD simples vers la base de données. Cette approche permet d'améliorer les performances et le référencement (SEO) en préparant les données côté serveur avant de les envoyer au client.

3.3 Base de données SQL

Pour la persistance des données, nous avons opté pour **SQLite**. C'est un moteur de base de données relationnelle léger, rapide et autonome. Contrairement aux systèmes client-serveur traditionnels (comme PostgreSQL ou MySQL), SQLite est intégré directement dans l'application. Ce choix simplifie grandement le déploiement et la gestion, tout en offrant une fiabilité robuste et le support complet du langage SQL, ce qui est idéal pour la structure et l'échelle de notre projet.

3.4 Autres outils et bibliothèques

En complément de Next.js, notre stack technique s'appuie sur plusieurs technologies clés :

- **TypeScript :** Un sur-ensemble typé de JavaScript qui apporte la sécurité du typage statique. Cela permet de détecter les erreurs en amont, d'améliorer la maintenabilité du code et d'offrir une meilleure expérience de développement grâce à l'autocomplétion.
- **Tailwind CSS :** Un framework CSS "utility-first" qui permet de concevoir des interfaces utilisateur modernes et réactives directement depuis le code HTML/JSX. Il accélère le développement en évitant de changer de contexte entre les fichiers CSS et les composants.
- **JSX (JavaScript XML) :** Une extension de syntaxe pour JavaScript qui permet d'écrire des structures de type HTML à l'intérieur du code JavaScript. C'est le langage standard pour décrire l'interface utilisateur dans l'écosystème React.

4 Architecture et structure du projet

4.1 Architecture globale

4.2 Structure des dossiers du projet

Le projet suit l'architecture recommandée par Next.js (App Router), organisée de la manière suivante :

- `app/` : Cœur de l'application, ce dossier contient les routes (pages), les layouts et l'API. Chaque sous-dossier correspond à une route de l'URL.
- `components/` : Regroupe les composants React réutilisables (boutons, formulaires, cartes, etc.) pour assurer une cohérence visuelle et fonctionnelle.
- `lib/` : Contient les utilitaires partagés, notamment la configuration de la base de données (`db.ts`) et les fonctions d'aide (`utils.ts`).
- `actions/` : Héberge les "Server Actions", des fonctions asynchrones exécutées côté serveur pour gérer la logique métier et les interactions sécurisées avec la base de données.
- `data/` : Dossier destiné à stocker le fichier de base de données SQLite (`cl0v.db`) et les scripts SQL d'initialisation.
- `types/` : Définit les interfaces et types TypeScript pour garantir la cohérence des données à travers l'application.

5 Conclusion générale

Ce projet nous a permis de concevoir et de développer une application web complète, baptisée **CL0V**, répondant aux besoins de gestion des activités parascolaires. En partant de l'analyse des besoins jusqu'à l'implémentation technique avec Next.js et SQLite, nous avons pu mettre en pratique les concepts fondamentaux du génie logiciel.

L'application réalisée est fonctionnelle, sécurisée et offre une interface utilisateur moderne. Ce travail a été une excellente opportunité pour approfondir nos compétences en développement full-stack et en gestion de base de données. À l'avenir, le projet pourrait être enrichi par de nouvelles fonctionnalités, telles qu'un système de notifications en temps réel ou une application mobile dédiée, pour améliorer encore l'expérience des utilisateurs.