 B3 - SDV

2025

Mini-projet Application de gestion bibliothèque

* Mini-projet BDD

LEFEVRE-BONZON Tom & MAHOUDEAU Ilies

Mini-projet : Application de gestion de bibliothèque

1. Introduction

a) Contexte

Dans le cadre du module d'Administration de Bases de Données, nous avons développé une application complète de gestion de bibliothèque universitaire. Ce projet met en pratique les concepts de conception de bases de données relationnelles, de développement Python et d'architecture applicative orientée données.

b) Objectifs

L'objectif principal était de créer une application permettant de :

- Gérer un catalogue de livres
- Gérer les étudiants inscrits à la bibliothèque
- Suivre les emprunts et retours de livres
- Calculer automatiquement les retards et amendes

c) Choix techniques

Nous avons fait le choix d'aller au-delà d'un simple menu interactif en console pour proposer une architecture moderne client-serveur :

- Backend : API REST en Python avec Flask
- Frontend : Application web avec Angular
- Base de données : PostgreSQL

2. Architecture du projet

L'architecture suit le pattern MVC (Modèle-Vue-Contrôleur) avec une séparation claire entre les couches.

```
bibliotheque-app/
■■■ backend/                    # API Python Flask
■   ■■■ app.py                 # Point d'entrée de l'API
■   ■■■ config/                # Configuration BDD
■   ■■■ models/                # Modèles de données
■   ■■■ services/              # Logique métier
■   ■■■ utils/                 # Utilitaires
■
■■■ frontend/                   # Application Angular
    ■■■ src/app/
```

Cette architecture assure :

- Modèles qui gèrent l'accès aux données
- Contrôleurs (routes Flask) qui traitent les requêtes
- Vues (composants Angular) qui affichent l'interface

3. Partie Backend

a) Technologies utilisées

Technologie	Version	Rôle
Python	3.x	Langage principal
Flask	3.x	Framework web
psycopg2	2.9.x	Driver PostgreSQL
Flask-CORS	5.x	Gestion cross-origin

b) Structure de l'API REST

Étudiants (/api/etudiants)

Méthode	Route	Description
GET	/api/etudiants	Liste tous les étudiants
GET	/api/etudiants/{id}	Récupère un étudiant
POST	/api/etudiants	Crée un étudiant
PUT	/api/etudiants/{id}	Modifie un étudiant
DELETE	/api/etudiants/{id}	Supprime un étudiant

Emprunts (/api/emprunts)

Méthode	Route	Description
GET	/api/emprunts/en-cours	Emprunts non retournés
GET	/api/emprunts/en-retard	Emprunts en retard
POST	/api/emprunts/{id}/retourner	Marque un retour

c) Gestion de la base de données

La connexion à PostgreSQL est centralisée avec un pattern sécurisé utilisant des requêtes paramétrées :

```
def execute_query(query, params=None, fetch=False):
    with get_connection() as conn:
        with conn.cursor(cursor_factory=RealDictCursor) as cur:
            cur.execute(query, params)  # Requête paramétrée
            if fetch:
                return cur.fetchall()
            conn.commit()
```

d) Règles métier implémentées

- Limite emprunts : Un étudiant ne peut pas avoir plus de 5 emprunts simultanés
- Calcul des retards : Durée emprunt maximale de 14 jours
- Calcul des amendes : 0.50€ par jour de retard

4. Partie Frontend

a) Technologies utilisées

Technologie	Version	Rôle
Angular	21.x	Framework frontend
TypeScript	5.x	Langage typé
Bootstrap	5.3	Framework CSS

b) Interfaces utilisateur

L'application propose une interface moderne avec :

- Tableau de données avec affichage paginé
- Formulaires de création/modification dans des modals Bootstrap
- Filtres et recherche (notamment pour les emprunts)
- Messages de confirmation pour les suppressions
- Alertes de succès/erreur avec auto-dismiss

5. Base de données

a) Modèle Conceptuel de Données (MCD)

Le MCD repose sur trois entités ETUDIANT, LIVRE, et EMPRUNT

Règles de cardinalités :

- Un étudiant peut effectuer 0 à n emprunts
- Un livre peut être emprunté 0 à n fois
- Un emprunt concerne exactement un étudiant et exactement un livre

b) Script SQL de création

```
-- Table ETUDIANT
CREATE TABLE etudiant (
  id SERIAL PRIMARY KEY,
  nom VARCHAR(100) NOT NULL,
  prenom VARCHAR(100) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE
);

-- Table LIVRE
CREATE TABLE livre (
  isbn VARCHAR(20) PRIMARY KEY,
  titre VARCHAR(255) NOT NULL,
  editeur VARCHAR(200) NOT NULL,
  annee_publication INTEGER
);

-- Table EMPRUNT
CREATE TABLE emprunt (
  id SERIAL PRIMARY KEY,
  etudiant_id INT REFERENCES etudiant(id),
  livre_isbn VARCHAR(20) REFERENCES livre(isbn),
  date_emprunt DATE NOT NULL,
  date_retour DATE
);
```

c) Contraintes d'intégrité

Contrainte	Table	Description
PRIMARY KEY	Toutes	Identifiant unique
FOREIGN KEY	emprunt	Liens vers etudiant et livre
UNIQUE	etudiant.email	Pas de doublons email
ON DELETE RESTRICT	emprunt	Empêche suppression si liés

6. Fonctionnalités implémentées

a) Gestion des ETUDIANT

- Affichage de la liste complète
- Ajout avec validation email
- Modification des informations
- Suppression (vérifie les emprunts liés)

b) Gestion des LIVRE

- Catalogue complet avec ISBN, titre, éditeur, année
- Gestion du nombre d'exemplaires disponibles
- Badge visuel de disponibilité

c) Gestion des EMPRUNT

- Création d'emprunt avec sélection étudiant/livre
- Filtrage : tous / en cours / en retard
- Enregistrement des retours
- Calcul automatique des jours de retard
- Calcul automatique des amendes (0.50€/jour)

d) Tableau de bord statistiques

- Nombre total d'étudiants, livres, emprunts
- Emprunts en cours et en retard
- Total des amendes
- Top 5 des étudiants les plus actifs
- Top 5 des livres les plus empruntés

7. Sécurité et robustesse

a) Protection contre les injections SQL

Toutes les requêtes utilisent des paramètres placeholders. Jamais de concaténation :

```
# Correct - Requête paramétrée
cur.execute("SELECT * FROM etudiant WHERE id = %s", (id,))

# DANGER - Concaténation
cur.execute(f"SELECT * FROM etudiant WHERE id = {id}")  # JAMAIS !
```

b) Validation des entrées

- Validation du format email avec regex
- Vérification des champs obligatoires
- Validation des années de publication

c) Gestion des erreurs

- Logging centralisé des erreurs
- Messages d'erreur explicites pour l'utilisateur
- Codes HTTP appropriés (400, 404, 500)
- Gestionnaire de contexte (with) pour les connexions

8. Guide d'installation

a) Prérequis

- Python 3.8+
- Node.js 18+
- PostgreSQL 14+

b) Installation du backend

```
cd backend
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python app.py
```

c) Installation du frontend

```
cd frontend
npm install
npm start
```

d) Accès à l'application

- Frontend : http://localhost:4200
- API Backend : http://localhost:5001/api

9. Conclusion

a) Bilan

Ce projet nous a permis de mettre en pratique l'ensemble des compétences du module :

- Conception BDD : Modélisation relationnelle, contraintes, index
- SQL avancé : Jointures, agrégations, requêtes paramétrées
- Développement Python : Architecture modulaire, gestion erreurs, API REST
- Développement applicatif : Interface utilisateur complète et ergonomique

b) Difficultés rencontrées

- Gestion du contexte this en TypeScript pour les callbacks
- Synchronisation des modals Bootstrap avec Angular
- Traduction des erreurs PostgreSQL en messages utilisateur

c) Améliorations possibles

- Authentification des utilisateurs
- Historique des modifications
- Export des données en CSV/PDF
- Notifications par email pour les retards
- Application mobile

Projet réalisé par Tom LEFEVRE-BONZON & Ilies MAHOUDEAU
B3 Développement - Sup de Vinci - 2025