Mamadou Baba Mbaye

Rapport détaillé du développement du backend

Projet : Plateforme de gestion d'examens

Technologies utilisées :

- Backend : Node.js avec Express.js
- Base de données : MySQL
- Authentification : JWT (JSON Web Token)
- Upload de fichiers : Multer
- Sécurité : bcrypt pour le hachage des mots de passe

1) Mise en place du projet backend

- **Initialisation**

Nous allons d'abord créer le projet ensuite initialiser les dépendances :

mkdir backend && cd backend

npm init -y

npm install express mysql2 bcrypt jsonwebtoken dotenv cors multer body-parser

Puis nous allons configurer package.json pour démarrer le serveur :

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

Enfin nous passons à la création du fichier .env afin de stocker les informations sensibles du projet tels que l'adresse de la base de donnée, son mot de passe et son user.

```
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=yourpassword
DB_NAME=exam_platform
JWT_SECRET=your_jwt_secret
```

- Configuration de la base de données MySQL

```sql
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  role ENUM('teacher', 'student') NOT NULL
);

CREATE TABLE exams (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  sujet_pdf VARCHAR(255),
  deadline DATETIME,
  teacher_id INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (teacher_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE submissions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  exam_id INT,
  student_id INT,
  file_url VARCHAR(255) NOT NULL,
  submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (exam_id) REFERENCES exams(id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES users(id) ON DELETE CASCADE
);
```

- Connexion à la base de données (config/db.js)

Backend > config > db.js > ...

```
const mysql = require('mysql2');
require('dotenv').config();
const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});

connection.connect((err) => {
  if (err) {
    console.error('Erreur de connexion à MySQL :', err);
    return;
  }
  console.log('Connecté à la base de données MySQL.');
});

module.exports = connection;
```

- Configuration du serveur (server.js)

Backend > server.js > port

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const port = 3000;

app.use(bodyParser.json());

// Import des routes
const authRoutes = require('./routes/auth');
const examRoutes = require('./routes/exams');
const submissionRoutes = require('./routes/submissions');

// Définition des endpoints
app.use('/api/auth', authRoutes);
app.use('/api/exams', examRoutes);
app.use('/api/submissions', submissionRoutes);
app.use('/uploads', express.static('uploads'));


app.get('/', (req, res) => {
  res.send("Plateforme de gestion d'examens - Backend");
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});
```

- Développement des routes API

Inscription (/api/auth/register) dans /routes/auth.js

```
60
61  // Endpoint d'inscription : POST /api/auth/register
62  router.post('/register', async (req, res) => {
63    const { email, password, name } = req.body;
64    try {
65      const saltRounds = 10;
66      const hashedPassword = await bcrypt.hash(password, saltRounds);
67      const query = `
68        INSERT INTO users (email, password, role, name)
69        VALUES ($1, $2, $3, $4) RETURNING *`;
70      const values = [email, hashedPassword, 'teacher', name];
71      const result = await pool.query(query, values);
72      res.status(201).json({ success: true, user: result.rows[0] });
73    } catch (error) {
74      console.error('Erreur lors de l\'enregistrement:', error);
75      res.status(500).json({ success: false, message: 'Erreur serveur' });
76    }
77  });
78
```

Connexion (api/auth/login) dans /routes/auth.js

```
79  // Endpoint de connexion : POST /api/auth/login
80  router.post('/login', async (req, res) => {
81    const { email, password } = req.body;
82    try {
83      const result = await pool.query('SELECT * FROM users WHERE email = $1', [email]);
84      if (result.rows.length > 0) {
85        const user = result.rows[0];
86        const isMatch = await bcrypt.compare(password, user.password);
87        if (isMatch) {
88          // Vous pouvez générer un token JWT ici pour sécuriser les endpoints réservés
89          return res.status(200).json({ success: true, userId: user.id });
90        }
91      }
92      res.status(401).json({ success: false, message: 'Identifiants invalides' });
93    } catch (error) {
94      console.error('Erreur lors du login:', error);
95      res.status(500).json({ success: false, message: 'Erreur serveur' });
96    }
97  });
98
```

Création d’un exam (api/exams/) dans /routes/exams.js

```
// Création d'un examen : POST /api/exams
router.post('/', upload.single('sujetPdf'), async (req, res) => {
  const { matiere, sujetText, submissionDeadline, teacher_id } = req.body;
  const file = req.file;
  const exam_pdf_path = file ? file.path : null;
  try {
    const query = `
      INSERT INTO exams (subject, exam_text, exam_pdf_path, submission_deadline, teacher_id)
      VALUES ($1, $2, $3, $4, $5) RETURNING *`;
    const values = [matiere, sujetText, exam_pdf_path, submissionDeadline, teacher_id];
    const result = await pool.query(query, values);
    // Vous pouvez déclencher ici la génération du corrigé type via DeepSeek si nécessaire
    res.status(200).json({ success: true, examId: result.rows[0].id, exam: result.rows[0] });
  } catch (error) {
    console.error('Erreur lors de la création de l\'examen:', error);
    res.status(500).json({ success: false, message: 'Erreur serveur' });
  }
});
```

Soumission d'un examen (api/etudiants/submit) dans /routes/submissions.js

```
// Soumission d'une copie : POST /api/etudiants/submit
router.post('/submit', upload.single('file'), async (req, res) => {
  const { name, className, examId } = req.body;
  const file = req.file;
  try {
    const query = `
      INSERT INTO submissions (exam_id, student_name, student_class, file_path)
      VALUES ($1, $2, $3, $4) RETURNING *`;
    const values = [examId, name, className, file ? file.path : null];
    const result = await pool.query(query, values);
    // Vous pouvez déclencher un job pour lancer en arrière-plan la correction automatique
    res.status(200).json({ success: true, submission: result.rows[0] });
  } catch (error) {
    console.error('Erreur lors de la soumission:', error);
    res.status(500).json({ success: false, message: 'Erreur serveur' });
  }
});
```