Backend Development Report

Asu Kumar

January 1, 2025

Contents

1 Introduction **2**

2 Technologies Used **2**

3 Project Features **2**
3.1 API Endpoints 2
3.2 Authentication 2
3.3 Database 3
3.4 Swagger Documentation 3

4 Code Highlights **3**
4.1 JWT Authentication 3
4.2 Prisma Schema 4

5 Steps to Run the Project **4**

6 Future Enhancements **4**

7 Conclusion **5**

1 Introduction

This report documents the development of a backend application using Next.js, Prisma, PostgreSQL, and associated technologies. The primary goal was to implement APIs for managing and querying college-related data, along with features like authentication, secure password handling, and API documentation.

2 Technologies Used

- **Next.js**: React-based framework for server-side rendering and API routes.
- **Prisma**: ORM for managing database interactions.
- **PostgreSQL**: Database for storing application data.
- **bcrypt**: Library for hashing passwords.
- **jsonwebtoken (JWT)**: Library for secure user authentication.
- **Swagger**: Tool for generating API documentation.

3 Project Features

3.1 API Endpoints

The following API endpoints were developed:

- **/api/auth/signup**: Handles user registration with hashed passwords.
- **/api/auth/login**: Authenticates users and issues JWT tokens.
- **/api/colleges**: Fetches colleges based on city and state filters.
- **/api/college**$_{c}$$ourses/[id] : Retrievescoursesforaspecificcollege, sortedbycoursefee.$**/api/college**$_{d}$$at$

3.2 Authentication

- Implemented user signup and login functionality.
- Used bcrypt for secure password hashing.
- Used JWT for user authentication and session management.
- Middleware created to validate JWTs and protect routes.

3.3 Database

The database schema includes tables for:

- **Users**: Stores user data (e.g., name, email, password).
- **Colleges**: Stores college information.
- **College Placements**: Stores placement statistics.
- **Courses**: Stores course-related details for colleges.

Prisma was used to define the schema, manage migrations, and seed the database with test data.

3.4 Swagger Documentation

Swagger was used to generate API documentation. Users can view and test the APIs interactively. Documentation is available at '/api-docs'.

4 Code Highlights

4.1 JWT Authentication

Listing 1: JWT Middleware

```
import jwt from 'jsonwebtoken';

const JWT_SECRET = process.env.JWT_SECRET || 'your_secret_key';

export function authenticate(req, res, next) {
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
        return res.status(401).json({ error: 'Unauthorized' });
    }

    const token = authHeader.split(' ')[1];
    try {
        const decoded = jwt.verify(token, JWT_SECRET);
        req.user = decoded;
        next();
    } catch (err) {
        return res.status(401).json({ error: 'Invalid token' });
    }
}
```

4.2 Prisma Schema

Listing 2: Prisma Schema Example

```
model User {
    id         Int      @id @default(autoincrement())
    name       String
    email      String   @unique
    password   String
    createdAt  DateTime @default(now())
    updatedAt  DateTime @updatedAt
}
```

5 Steps to Run the Project

1. Clone the repository:

```
git clone <repository_url>
```

2. Install dependencies:

```
npm install
```

3. Configure environment variables in ‘.env‘:

```
DATABASE_URL=postgresql://<username>:<password>@<host>:<port
   >/<database>
JWT_SECRET=your_secret_key
```

4. Run Prisma migrations:

```
npx prisma migrate dev --name init
```

5. Seed the database (optional):

```
npm run seed
```

6. Start the development server:

```
npm run dev
```

6 Future Enhancements

- Implement refresh tokens for extended sessions.
- Add role-based access control (RBAC) for administrative features.
- Integrate pagination and search functionality in APIs.

7 Conclusion

This project demonstrates the implementation of a robust backend application with secure authentication, efficient database management, and comprehensive API documentation.