Orange Backend Project Structure

Overview

This document provides an overview of the backend structure for the **Orange** project. The backend is built with **Node.js** and **Express** and uses **PostgreSQL** as the database, which is managed via **Sequelize** ORM (**Sequelize** is an **Object-Relational Mapping (ORM) library** for Node.js that makes it easier to work with SQL databases like PostgreSQL).

Project Structure

Here's a quick breakdown of the folders and their purposes:

```
src
 ├── config          # Configuration files for database, environment, and logging
 ├── controllers     # Business logic for handling requests, interacting with services
 ├── docs            # Documentations files
 ├── middlewares     # Middleware functions for request handling, auth, error handling
 ├── models          # Sequelize models for database tables
 ├── routes          # API route definitions and setup
 ├── service         # Business logic, interacts with models (e.g., CRUD operations)
 ├── utils           # Helper functions for consistent responses and utility tasks
 └── test            # Tests for unit, integration, and end-to-end testing
```

Folder & File Descriptions

1. **config**
    - **database.js**: Sets up and exports the database connection.
2. **controllers**
   Controllers handle HTTP requests and responses. Each controller focuses on a specific entity.
    - **groupController.js**: Logic for managing groups.
    - **studentController.js**: Logic for managing students.
    - Similar controllers exist for notifications, join requests, etc.
3. **docs**
    - Contains documentation resources for developers, such as API specs, architectural diagrams, and workflows.

4. **middlewares**
    - **authMiddleware.js**: Checks for and verifies JWT tokens for protected routes.
    - **errorHandler.js**: Catches and handles errors, sending appropriate responses.
    - Additional middlewares can be added here for other reusable logic.
5. **models**
    Database models that define the structure of database tables, are managed via Sequelize ORM.
    - **Student.js**: Defines fields like studentId, tel_account for the Student table.
    - **Group.js**: Defines fields for the Group table, such as name, description, and membersLimit.
    - Additional models for entities like Notification, JoinRequest, FbToken, etc.
6. **routes**
    - **index.js**: Sets up all the API routes and links them to their respective controllers.
7. **service**
    The service layer handles complex business logic and database interactions, separating it from controllers.
    - **group_service.js**: Manages group-related business logic and database queries.
    - **student_service.js**: Handles student-related operations.
    - Similar services exist for notifications, join requests, etc.
8. **utils**
    Utility/helper functions.
    - **response.js**: Formats responses for consistency across the API.
9. **test**
    Holds test files to ensure functionality and reliability.
    - **Unit Tests**: For individual functions.
    - **Integration Tests**: For testing how components work together.
    - **End-to-end Tests**: For simulating real user flows (if applicable).

Key Dependencies

- **express**: For building the API server.
- **sequelize**: ORM for database management.
- **pg**: PostgreSQL client for Node.js.
- **dotenv**: Manages environment variables.

- **jsonwebtoken**: Handles JWT-based authentication.
- **bcrypt**: Used for password hashing.
- **cors**: Enables Cross-Origin Resource Sharing for frontend-backend interactions.
- **body-parser**: Parses JSON and URL-encoded data from requests.
- **nodemon**: Restarts the server automatically during development.

Development Guidelines

1. **Environment Variables**: Keep sensitive information like database credentials in the .env file.
2. **Coding Standards**: Use **Prettier** and **ESLint** to maintain code quality and formatting consistency.
3. **API Documentation**: Refer to docs/api-docs.md for API endpoint details.
4. **Testing**: Place all test files in the test folder. Run tests before pushing changes.

Running the Project

1. **Install Dependencies**:

   ```
   npm install
   ```

2. **Set Up Environment Variables**: Create a .env file with relevant configurations.
3. **Run the Server**:

   ```
   npm start
   ```

   - For development with automatic restarts:

     ```
     npm run dev
     ```

4. **Database Migrations** (if using Sequelize migrations):

   ```
   npx sequelize db:migrate
   ```