Backend >
.gitignore

This file is a `.gitignore` file used to specify which files and directories should be ignored by Git, preventing them from being tracked in the repository. Below is a breakdown of each section and the purpose of the entries:

Dependencies

```
/node_modules
/.env
/.pnp
.pnp.js
```

- **/node_modules**: This directory contains all the npm packages installed for the project. It can be large and is generally not included in version control, as it can be regenerated using `npm install`.
- **/.env**: This file typically contains environment-specific variables. It is often excluded for security reasons, as it may contain sensitive information like API keys or database credentials.
- **/.pnp and .pnp.js**: These files are related to Plug'n'Play, a feature of Yarn that changes how Node.js modules are resolved. They can be regenerated by Yarn and are not necessary to include in version control.

Testing

```
/coverage
```

- **/coverage**: This directory is usually created by test coverage tools (like Jest) and contains the test coverage reports. It is often excluded as it can be large and is dynamically generated.

Production

```
/build
```

- **/build**: This directory often contains the build output of the project, which is generated during the build process and does not need to be included in version control.

Miscellaneous

```
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local
```

- **.DS_Store**: This is a hidden file created by macOS to store custom attributes of a folder. It is unnecessary for the project and can be ignored.
- **.env.local, .env.development.local, .env.test.local, .env.production.local**: These files contain environment-specific configurations. They are often excluded to avoid exposing sensitive information and to ensure environment-specific configurations are not shared.

Logs

npm-debug.log*
yarn-debug.log*
yarn-error.log*
```
- **npm-debug.log***: These files are created when npm encounters errors. They are useful for debugging but should not be included in version control.
- **yarn-debug.log* and yarn-error.log***: Similar to `npm-debug.log`, these files are created by Yarn when errors occur. They are helpful for debugging but should not be tracked in the repository.

**### Summary**
The `.gitignore` file ensures that temporary files, build artifacts, dependency directories, environment files, and debug logs are not included in the version control system. This helps keep the repository clean and prevents unnecessary files from being committed, reducing the risk of exposing sensitive information and keeping the repository size manageable.

---

**index.js**

This code sets up an Express.js server with MongoDB as the database. Below is a detailed explanation of each part of the code:

## Importing Dependencies

- **express**: A minimal and flexible Node.js web application framework.
- **cors**: Middleware to enable Cross-Origin Resource Sharing.
- **mongoose**: An ODM (Object Data Modeling) library for MongoDB and Node.js.
- **dotenv**: A module that loads environment variables from a `.env` file into `process.env`.

## Configuring Environment Variables

- Loads the environment variables from a `.env` file into `process.env`.

## Creating Express App

- Initializes an Express application.
```

- Imports route handlers from the `route.js` file.

Setting Up MongoDB Connection

- Retrieves the MongoDB URI and port number from environment variables, with fallback defaults.

Middleware Setup

- **express.json()**: Middleware to parse incoming JSON requests with a body size limit of 10MB.
- **cors()**: Middleware to enable CORS with default settings.

Connecting to MongoDB

- **mongoose.connect()**: Connects to the MongoDB database using the provided URI and options.
- **then()**: Logs a success message when connected.
- **catch()**: Logs an error message if the connection fails.

Using Routes

- Mounts the routes defined in the `route.js` file to the root path.

Error Handling Middleware

- A basic error-handling middleware that logs the error stack and sends a 500 response with a message.

Starting the Server

- Starts the Express server on the specified port and logs a message indicating that the server is running.

Summary

This code initializes an Express.js application, connects to a MongoDB database, sets up middleware for JSON parsing and CORS, defines routes, handles errors, and starts the server. The use of environment variables makes the application configuration flexible and secure. The routes are imported from an external file, keeping the main server file clean and organized.