**EasyGenerator Test Project Documentation**

**Table of Contents**

1. Introduction
2. Project Overview
3. Backend Architecture
   * Features
   * API Endpoints
   * Modules and Components
   * Security Best Practices
4. Frontend Architecture
   * Features
   * Components
   * Routing
5. System Design
   * Diagrams
     + Backend Architecture Diagram
     + Frontend Architecture Diagram
     + Complete Application Diagram
6. Implementation Details
   * Backend
     + Dependencies
     + File Structure
     + Key Features
   * Frontend
     + Dependencies
     + File Structure
     + Key Features
7. Setup Instructions
   * Prerequisites
   * Backend Setup
   * Frontend Setup
8. Testing
   * Manual Testing
   * Automated Testing
9. Future Enhancements
10. **Introduction**

This project is a secure, scalable, and interactive full-stack web application built using NestJS for the backend and React with TypeScript for the frontend. The system includes user authentication, profile management, rate limiting, CORS and other security best practices.

1. **Project Overview**

* Backend: Built using NestJS with MongoDB for data storage.
* Frontend: Built using React with TypeScript.
* Security Implementations: Implements rate limiting, and CORS and other practices for secure communication.
* Purpose: Demonstrates a clean architecture for authentication-based applications with modern development practices.

1. **Backend Architecture**

* **Features**
* User authentication (sign up, sign in, and logout).
* Profile management.
* Stateless authentication using JWT.
* Rate limiting for API endpoints.
* CSRF protection for state-changing operations.
* Logging and error handling.
* **API Endpoints**

|  |  |  |
| --- | --- | --- |
| **Endpoint** | **Method** | **Description** |
| /auth/signup | POST | Registers a new user. |
| /auth/signin | POST | Authenticates a user. |
| /auth/logout | POST | Logs out a user. |
| /auth/csrf-token | GET | Fetches the CSRF token. |
| /users/profile | GET | Fetches the user profile. |

* **Modules and Components**
* **AuthModule:**
  + **AuthController:** Handles authentication routes**.**
  + **AuthService:** Implements authentication logic**.**
  + **JwtStrategy:** Validates JWT tokens**.**
  + **UserSchema:** Defines the MongoDB schema for users**.**
* **UsersModule:**
  + **UsersController:** Handles user-related routes**.**
  + **UsersService:** Implements business logic for user data**.**
  + **UsersRepository:** Manages database queries for users**.**
* **ThrottlerModule:**
  + Implements rate limiting globally**.**
* **DatabaseConfig:**
  + Manages MongoDB connection**.**

**Security Best Practices**

* **Password Hashing**
  + Passwords are hashed using bcrypt before storing them in the database.
  + Ensures raw passwords are never stored, protecting users even if the database is compromised.
* **JWT Authentication**
  + JWT tokens are signed with a secret (JWT\_SECRET) and have a limited validity period (JWT\_EXPIRES\_IN).
  + Protects endpoints and ensures that authentication tokens cannot be easily forged.
* **Environment Variables**
  + Sensitive data like JWT\_SECRET and MONGO\_URI are stored in a .env file and loaded using dotenv.
  + Prevents exposing sensitive information in the source code.
* **Validation of Input Data**
  + Input validation is enforced using class-validator and ValidationPipe.
  + Prevents malformed or unexpected data from entering the system, mitigating attacks like SQL injection and XSS.
* **Whitelisting Validation**
  + The ValidationPipe is configured with whitelist: true and forbidNonWhitelisted: true.
  + Removes unexpected properties from requests and blocks unauthorized fields.
* **Error Handling**
  + A global exception interceptor handles and formats errors consistently.
  + Prevents sensitive application details from being exposed in error responses.
* **Rate Limiting**
  + Limits API requests using ThrottlerModule (10 requests per 60 seconds globally).
* **CORS**
  + Restricts backend access to trusted frontend origins.
  + Allows cookies and specific HTTP methods.

1. **Frontend Architecture**

**Features**

1. User authentication (sign up, sign in, and logout).
2. Profile page with protected routes.
3. Integration with the backend using Axios.
4. Context API for global authentication state management.

**Components**

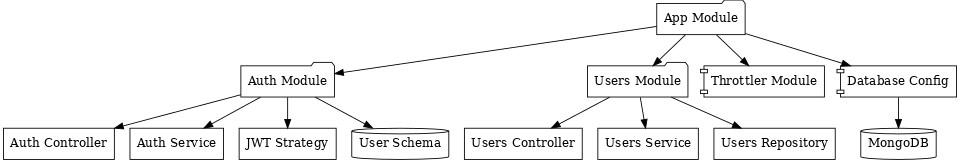
1. **AuthProvider:**
   * Manages authentication state and JWT tokens.
2. **AppRoutes:**
   * Defines the application routes.
3. **Pages:**
   * **SignUp**: Registration page.
   * **SignIn**: Login page.
   * **Profile**: Protected profile page.
4. **Utilities:**
   * **PrivateRoute:** Protects private routes.

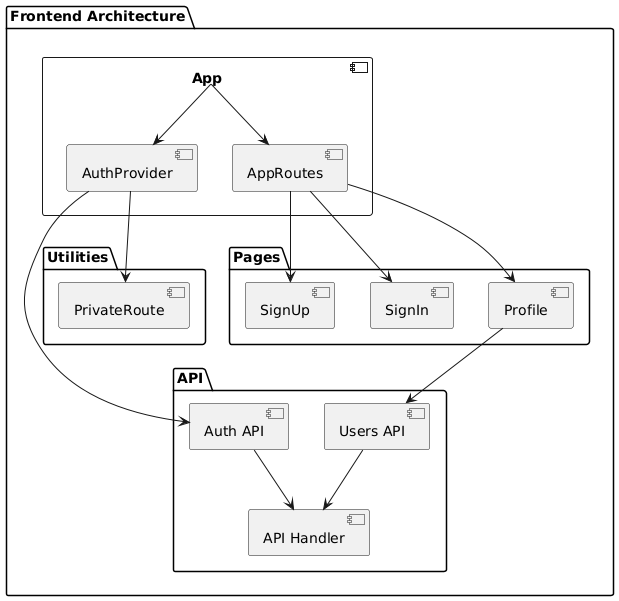
**Routing**

* **Public Routes:**
  + /signup
  + /signin
* **Protected Routes:**
  + /profile

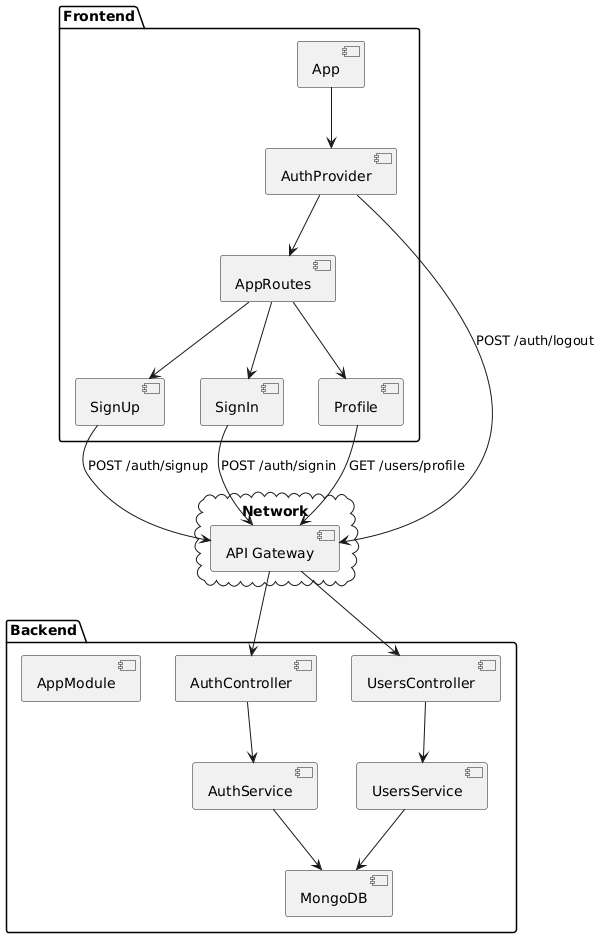
1. **System Design**

**Diagrams**

**Backend Architecture Diagram**

**Frontend Architecture Diagram**

**Complete Application Diagram**

****

1. **Setup Instructions**

**Backend Setup**

1. Install dependencies:

npm install

1. Create a .env file:

MONGO\_URI=mongodb://localhost:27017/nest-auth

JWT\_SECRET=your\_secret\_key

JWT\_EXPIRES\_IN=1h

PORT=3000

1. Start the backend server:

npm run start:dev

**Frontend Setup**

1. Install dependencies:

npm install

1. Start the frontend application:

npm run dev

1. **Testing**

**Backend**:

* + Unit tests for services and controllers.
  + Integration tests for APIs using supertest.

**Frontend**:

* + Unit tests for components and context.
  + End-to-end tests using Cypress.

**Run tests:**

* **Backend**: npm run test for unit and integration tests.
* **Frontend**: npm test for component tests and npx cypress open for E2E tests.