Securing a REST API Using JWT Tokens

JWT (JSON Web Token) stands out as a widely adopted method for securing REST APIs, offering a stateless and scalable approach to authentication and authorization. In this tutorial, we'll explore an example of securing a REST API using JWT tokens, implemented in the Nest.js framework. We'll leverage popular packages like jsonwebtoken for token generation and validation, as well as other dependencies such as bcrypt for password hashing and dotenv for managing environment variables.

Let's start by setting up the package.json file. Ensure that essential dependencies like Nest.js for building the API and jsonwebtoken for JWT functionalities are included. Additionally, include dependencies like bcrypt for password hashing.



In an ideal setup, we would store the JWT secret inside a .env envoirment variable file to store sensitive information. It's crucial to keep these values secure and not expose them directly in the code, in this tutorial we will use 'secret' as the value.

In the core application of Nest.js, create a controller that utilizes an Auth Guard. The Auth Guard is responsible for extracting the JWT token from the Authorization request header and validating it using the secret variable used for generating it. The Auth Guard also modifies the request by setting the .user attribute on the request object, allowing access to the user's ID inside the controller if needed.



In the auth guard we can implement multiple policy implementation, like API keys for example, the auth guard is built in a way that allows to be extended for ther authentication strategies and also it is decorator based ,meaning that we can decorate any Controller class or Controller method with this @UseGuards(AuthGuard([...policies])) decorator and we'll know that the controller or method (rest endpoint) will be secure.



In this tutorial, we will only implement the JWT policy by implementing a method that extracts and validates the authorization header and returns an object containing the id of the user





For user login functionality, create a separate method inside the controller that fetches the user based on their email. Utilize bcrypt's compare function to compare the hashed password from the fetched user with the plain password provided in the request body.



In summary, this tutorial provides a step-by-step guide on securing a Nest.js REST API application using JWT tokens. The Auth Guard implementation allows for flexible and extensible authentication strategies, and the overall structure ensures a secure and scalable approach to user authentication and authorization in a Nest.js environment.