Documentation

Intro

This code defines an Express server that connects to a MongoDB database and has two endpoints for user authentication: `/user/signup` and `/user/login`. The authentication uses JSON Web Tokens (JWT) to secure communication between the server and client.

The server listens on port 8082 and logs to the console when it starts up. It also handles CORS requests by allowing requests from any origin and accepting headers specified in the `Access-Control-Allow-Headers` header. The routes are defined in separate modules and are imported into `app.js`.

The `signup` endpoint checks if the user already exists in the database and returns an error if they do. If the user doesn't exist, the password is hashed with a salt and stored in the database. The `login` endpoint verifies the user's email and password by comparing the hashed password with the one entered by the user. If the credentials are correct, a JWT is generated and returned to the user, which they can use to authenticate with other endpoints that require authentication.

Middleware

The provided code is an implementation of stateless authentication middleware for protecting resources in a Node.js application.

The middleware uses the `jsonwebtoken` package to verify a JSON Web Token (JWT) included in the request header's `authorization` field. If the token is valid, the middleware extracts the user data from the decoded token and attaches it to the request object for further use by downstream middleware or handlers. If the token is invalid or missing, the middleware returns a 401 Unauthorized response with an error message.

To use the middleware, you need to import it into your Node.js application by requiring the file that contains the middleware function. You should also ensure that you have the `jsonwebtoken` package and the `dotenv` package installed. The latter is used to load environment variables from a `.env` file, which should contain a `JWT_KEY` variable with a secret key used to sign and verify JWTs.4

Once the middleware is imported, you can add it to the routes that you want to protect. This is typically done by passing the middleware function as a second argument to the HTTP method handler function (e.g., `router.get('/protected', checkAuth, (req, res) => {...})`).

It's important to note that the `next()` function call in the middleware should only be called when the token is valid, as it passes control to the next middleware or handler in the chain. If the token is invalid or missing, the middleware should return an error response and not call `next()`.

In additon don’t forget to add this to the top of your routes/yourfilename.js

“”const checkAuth = require('../middleware/check-auth');””

Dotenv

dotenv is more secure because it allows developers to store sensitive information such as passwords, API keys, and other secrets in a separate configuration file that is not included in the source code repository. This means that even if an attacker gains access to the source code, they will not be able to access sensitive information directly.

In contrast, hardcoding sensitive information directly into the source code or storing them in plain text files can expose the information to potential attackers who can gain access to the source code or the file system. Additionally, dotenv uses environment variables to store the configuration information, which are more secure than using global variables or command line arguments because they are not visible to other processes running on the same machine.

Furthermore, dotenv supports encryption and decryption of sensitive information using third-party tools, making it even more secure for storing confidential data. Overall, dotenv provides a more secure way to manage sensitive configuration information in an application.

To use make .env file with the following

JWT_KEY=YourKey

Your mongoDB address like this

MONGODB_URI="mongodb+srv://Scott:[PASSWORD]@choco.tw7hylw.mongodb.net/?retryWrites=true&w=majority"

Load the .env file in your Node.js application using dotenv. This can be done by calling require('dotenv').config() at the top of your application's entry file, such as app.js or server.js.

Like this

require('dotenv').config({ path: 'sec.env' });

You can also pass a configuration object to `config()` to specify the path of the `.env` file, like this:

(I did not do this not and this text was generated by ChatGPT your mileage may vary)

require('dotenv').config({ path: '/full/custom/path/to/your/env/vars' })

Access the environment variables in your code using process.env. For example, to access the DB_URL variable from the .env file, you can use process.env.DB_URL.

const dbUrl = process.env.DB_URL;

Dependencies:

```
 "dependencies":
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "config": "^3.3.9",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.0.0",
    "morgan": "^1.10.0",
    "multer": "^1.4.5-lts.1"

  "devDependencies":
    "nodemon": "^2.0.20"
```

Sources:
Building a RESTful API with Node.js
Academind:
https://www.youtube.com/playlist?list=PL55RiY5tL51q4D-B63KBnygU6opNPFk_q
https://github.com/academind/node-restful-api-tutorial

Dotenv:
https://github.com/motdotla/dotenv