**Presentation: Authentication Middleware with JWT Tokens**

**1. What is Authentication Middleware?**

Authentication middleware is a part of the software that processes requests and responses in a web application, ensuring that only authenticated users can access certain routes or resources. It intercepts requests and checks for user credentials to verify their identity before allowing access.

**2. What is JWT (JSON Web Token)?**

JWT (JSON Web Token) is a compact, URL-safe token that securely transmits information between parties as a JSON object. It's used for authentication and information exchange. A typical JWT consists of three parts:

* **Header**: Contains information about the token's type and the hashing algorithm.
* **Payload**: Contains the user's claims (information like user ID or role).
* **Signature**: Verifies the token’s integrity, ensuring that the payload hasn’t been tampered with.

**3. How JWT Authentication Middleware Works**

* A user logs in by providing credentials (e.g., email and password).
* If the credentials are valid, the server generates a JWT and sends it back to the client.
* The client stores the JWT (usually in localStorage or cookies).
* For each subsequent request, the client sends the JWT in the Authorization header.
* The middleware checks the JWT and verifies its validity. If valid, the request proceeds; otherwise, it returns an error.

**4. Code Snippet: JWT Authentication Middleware in Express.js**

Below is an example using Node.js, Express, and JWT.

**Step 1: Install Required Packages**

**4. Code Snippet: JWT Authentication Middleware in Express.js**

**Below is an example using Node.js, Express, and JWT.**

**Step 1: Initialize a New Node.js Project**

**First, let's create a new project folder and initialize it.**

1. **Create a new folder for your project:**

**bash**

**Copy code**

**mkdir jwt-authentication-demo**

**cd jwt-authentication-demo**

1. **Initialize a Node.js project:**

**bash**

**Copy code**

**npm init -y**

1. **Install required dependencies: You'll need the following packages:**
   * **express: For handling HTTP requests.**
   * **jsonwebtoken: For generating and verifying JWT tokens.**
   * **bcryptjs: For hashing and comparing passwords.**
   * **mongoose: For connecting to MongoDB (optional, if you want to use MongoDB).**

**Run the following command:**

**bash**

**Copy code**

**npm install express jsonwebtoken bcryptjs mongoose**

1. **Install nodemon (optional, but useful for development to auto-restart the server):**

**bash**

**Copy code**

**npm install --save-dev nodemon**

**Step 2: Setup Express and Basic Server**

**Create an index.js file to initialize the Express app and create routes.**

**javascript**

**Copy code**

**// index.js**

**import express from 'express';**

**import mongoose from 'mongoose';**

**import { registerUser, loginUser } from './authController.js';**

**import { authenticateJWT } from './authMiddleware.js';**

**const app = express();**

**app.use(express.json());**

**// Connect to MongoDB (optional, if using MongoDB)**

**mongoose.connect('mongodb://localhost:27017/jwtDemo', { useNewUrlParser: true, useUnifiedTopology: true })**

**.then(() => console.log('MongoDB connected'))**

**.catch(err => console.log(err));**

**// Public routes**

**app.post('/register', registerUser); // Register user**

**app.post('/login', loginUser); // Login user**

**// Protected route**

**app.get('/dashboard', authenticateJWT, (req, res) => {**

**res.json({ message: `Welcome, ${req.user.username}` });**

**});**

**app.listen(3000, () => console.log('Server running on port 3000'));**

**Step 3: User Model (MongoDB)**

**For storing users in a database, we'll use MongoDB. Here's a simple User model:**

**javascript**

**Copy code**

**// userModel.js**

**import mongoose from 'mongoose';**

**import bcrypt from 'bcryptjs';**

**const userSchema = new mongoose.Schema({**

**username: { type: String, required: true, unique: true },**

**email: { type: String, required: true, unique: true },**

**password: { type: String, required: true }**

**});**

**// Hash the password before saving**

**userSchema.pre('save', async function (next) {**

**if (!this.isModified('password')) return next();**

**this.password = await bcrypt.hash(this.password, 10);**

**next();**

**});**

**export const User = mongoose.model('User', userSchema);**

**Step 4: User Registration and Login Controller**

**Now let's create the authController.js file to handle user registration and login.**

**javascript**

**Copy code**

**// authController.js**

**import { User } from './userModel.js';**

**import bcrypt from 'bcryptjs';**

**import jwt from 'jsonwebtoken';**

**// Register user**

**export const registerUser = async (req, res) => {**

**const { username, email, password } = req.body;**

**try {**

**// Check if user already exists**

**let user = await User.findOne({ email });**

**if (user) return res.status(400).json({ message: 'User already exists' });**

**// Create new user**

**user = new User({ username, email, password });**

**await user.save();**

**res.status(201).json({ message: 'User registered successfully' });**

**} catch (err) {**

**res.status(500).json({ message: 'Server error', error: err.message });**

**}**

**};**

**// Login user**

**export const loginUser = async (req, res) => {**

**const { email, password } = req.body;**

**try {**

**// Check if user exists**

**const user = await User.findOne({ email });**

**if (!user) return res.status(400).json({ message: 'User not found' });**

**// Compare passwords**

**const isMatch = await bcrypt.compare(password, user.password);**

**if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });**

**// Generate JWT**

**const token = jwt.sign({ id: user.\_id, username: user.username }, 'yourSecretKey', { expiresIn: '1h' });**

**res.json({ token });**

**} catch (err) {**

**res.status(500).json({ message: 'Server error', error: err.message });**

**}**

**};**

**Step 5: Authentication Middleware**

**Create the authMiddleware.js file to handle JWT verification.**

**javascript**

**Copy code**

**// authMiddleware.js**

**import jwt from 'jsonwebtoken';**

**export const authenticateJWT = (req, res, next) => {**

**const token = req.header('Authorization')?.split(' ')[1]; // Extract token from "Bearer <token>"**

**if (!token) return res.status(401).json({ message: 'No token, authorization denied' });**

**try {**

**const decoded = jwt.verify(token, 'yourSecretKey'); // Verify token with your secret key**

**req.user = decoded; // Attach user info to request object**

**next();**

**} catch (err) {**

**res.status(401).json({ message: 'Token is not valid' });**

**}**

**};**

**Step 6: Running the Project**

1. **Start your MongoDB server (if using MongoDB locally):**

**bash**

**Copy code**

**mongod**

1. **Run the Node.js server: If you have nodemon, use:**

**bash**

**Copy code**

**npx nodemon index.js**

**Otherwise, use:**

**bash**

**Copy code**

**node index.js**

**Step 7: Testing the API**

**Use a tool like Postman or cURL to test the API.**

1. **Register a new user (POST request to /register):**

**json**

**Copy code**

**POST http://localhost:3000/register**

**Content-Type: application/json**

**{**

**"username": "john",**

**"email": "john@example.com",**

**"password": "password123"**

**}**

1. **Login with the registered user (POST request to /login):**

**json**

**Copy code**

**POST http://localhost:3000/login**

**Content-Type: application/json**

**{**

**"email": "john@example.com",**

**"password": "password123"**

**}**

**This will return a JWT token, e.g.:**

**json**

**Copy code**

**{**

**"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."**

**}**

1. **Access the protected route (GET request to /dashboard): Include the token in the Authorization header:**

**makefile**

**Copy code**

**Authorization: Bearer <yourJWT>**

**Response if successful:**

**json**

**Copy code**

**{**

**"message": "Welcome, john"**

**}**