JWT Authentication in Express.js using jsonwebtoken

jsonwebtoken (JWT) is a package used to generate and verify **JSON Web Tokens (JWTs)** for user authentication in Express.js.

1. Installing jsonwebtoken

Install the package:

```
npm install jsonwebtoken
```

---

2. Generating & Verifying JWT in Express

Step 1: Setup Express and JWT

```
const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();
app.use(express.json()); // Middleware to parse JSON
const SECRET_KEY = 'mySecretKey'; // Secret key for signing tokens
```

Step 2: User Login & Token Generation

Simulate a user login and generate a JWT:

```
app.post('/login', (req, res) => {
    const { username, password } = req.body;
    // Dummy user validation (In real apps, check with database)
    if (username === 'admin' && password === 'password123') {
        // Generate a JWT token
        const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: '1h' });
        res.json({ message: 'Login successful', token });
    } else {
        res.status(401).json({ message: 'Invalid credentials' });
    }
});
```

➢ **Example Request:**

POST /login
Content-Type: application/json

```
{
   "username": "admin",
   "password": "password123"
}
```

- **Example Response:**

```
{
   "message": "Login successful",
   "token": "eyJhbGciOiJIUzI1NiIsIn..."
}
```

Step 3: Middleware to Verify JWT

Create a middleware to protect routes:

```
const verifyToken = (req, res, next) => {
   const token = req.headers['authorization'];

   if (!token) {
      return res.status(403).json({ message: 'Access denied, token missing' });
   }
   jwt.verify(token.split(' ')[1], SECRET_KEY, (err, decoded) => {
      if (err) {
         return res.status(401).json({ message: 'Invalid token' });
      }
      req.user = decoded; // Store user data in request object
      next();
   });
};
```

Step 4: Protect Routes with JWT

Apply the verifyToken middleware to secure an endpoint:

```
app.get('/protected', verifyToken, (req, res) => {
   res.json({ message: 'You have access!', user: req.user });
});
```

- **Example Request (Attach Token in Headers):**

```
GET /protected
Authorization: Bearer eyJhbGciOiJIUzI1NiIsIn...
```

➢ **Example Response:**

```
{
   "message": "You have access!",
   "user": { "username": "admin", "iat": 1700000000, "exp": 1700003600 }
}
```

---

5. Handling Token Expiration

JWT tokens **expire** (set with expiresIn). If expired, users must **log in again** to get a new token.

➢ **Modify token expiration in login route:**

```
const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: '30s' });
```

➢ **If expired, API will return:**

```
{
   "message": "Invalid token"
}
```

---

6. Refreshing JWT Tokens

If you want to implement **token refreshing**, use a refresh token:

- Store a refresh token in a database.
- Use it to get a new JWT when the old one expires.

---

7. Summary

Feature	Code Example
Install jsonwebtoken	npm install jsonwebtoken
Generate JWT	jwt.sign(payload, SECRET_KEY, { expiresIn: '1h' })
Verify JWT	jwt.verify(token, SECRET_KEY, callback)
Protect routes	app.use(verifyToken)
Handle expiration	expiresIn: '30s'
