# JWT

JWT (JSON Web Token) authentication is a secure way to handle authentication in a Node.js application. It allows users to log in and receive a token, which is then used to access protected resources.

## What is JWT?

JWT is a compact, URL-safe token format used to securely transmit information between parties as a JSON object. It consists of three parts:

1. **Header** – Contains the type of token and hashing algorithm.
2. **Payload** – Contains user data (e.g., `id`, `email`).
3. **Signature** – Ensures the token has not been tampered with.

## Steps to Implement JWT Authentication in Node.js

We will create a simple Node.js authentication system using JWT with the following steps:

1. **Set up a Node.js project**
2. **Install necessary dependencies**
3. **Create an Express server**
4. **Implement user authentication (login, protected routes)**
5. **Use JWT for authentication**
6. **Test the authentication flow**

Middleware

```
import { NextFunction, Response } from "express";
import jwt from "jsonwebtoken";
import { AuthRequest } from "../types/authRequest";

const secretKey = "secretkey";

export const authenticateUser = (
  req: AuthRequest, // Custom request type
  res: Response,
  next: NextFunction
) => {
  //  Corrected: Use req.headers.get("authorization") OR type assertion
  const authHeader = (req.headers as unknown as Record<string,
string>)["authorization"];

  if (!authHeader || !authHeader.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Access denied. No token provided."
});
  }

  const token = authHeader.split(" ")[1];

  try {
    const decoded = jwt.verify(token, secretKey);
    req.user = decoded; // Attach decoded user to request
    next();
  } catch (error) {
    res.status(401).json({ error: "Invalid token." });
  }
};
```

## Generate a Random Secret Key using Node.js

Run the following command in your terminal:

```
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

```
Types/authRequest.ts


export interface AuthRequest extends Request {
  user?: any; //
}
```

### createUser in Repository

```
async createUser(name: string, email: string, password: string) {
        const pool = await poolPromise;
        const result = await pool
          .request()
          .input("name", sql_.VarChar, name)
          .input("email", sql_.VarChar, email)
          .input("password", sql_.VarChar, password)
          .query("INSERT INTO Users (name, email, password) VALUES (@name,
@email, @password)");
        return result;
      }
```

**authService.ts**

```
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const userAuthRepository = require("../repositories/userRepository");

const SECRET_KEY = "secretKey";

class AuthService {
  static async registerUser(name: any, email: any, password: any) {
    const hashedPassword = await bcrypt.hash(password, 10);
    return await userAuthRepository.createUser(name, email, hashedPassword);
  }

  static async loginUser(email: any, password: any) {
    const user = await userAuthRepository.getUserByEmail(email);
    if (!user) throw new Error("User not found");

    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) throw new Error("Invalid credentials");

    const token = jwt.sign({ name: user.name, email: user.email }, SECRET_KEY,
{ expiresIn: "1h" });
    return { user, token };
  }
}

module.exports = AuthService;
```

authController.ts

```
const authService = require("../services/authService");
 import { Request, Response } from "express";

class AuthController {
  static async register(req: Request, res: Response) {
    try {
      const { name, email, password } = req.body;
      await authService.registerUser(name, email, password);
      res.status(201).json({ message: "User registered successfully" });
    } catch (error) {
      if (error instanceof Error) res.status(500).json({ error: error.message
});
    }
  }

  static async login(req: Request, res: Response) {
        try {
          const { email, password } = req.body;
          const data = await authService.loginUser(email, password);
          res.status(200).json({ message: "Login successful", ...data });
        } catch (error) {
            if(error instanceof Error)
          res.status(401).json({ error: error.message });
        }
      }
}

module.exports = AuthController;
```

AuthRoute.ts

```
const expr = require("express");
const authController = require("../controllers/authController");

const authRouter = expr.Router();

authRouter.post("/register", authController.register);
authRouter.post("/login", authController.login);

module.exports = authRouter;
```

**route.ts**

```
router.get("/", authenticateUser,userController.getAllUsers);
```

**app.ts**

```
const express = require("express");
const userRoutes = require("./routes/userRoutes");
const authRoutes = require("./routes/authRoute");
const app = express();

app.use(express.json());

// Routes
app.use("/users", userRoutes);
app.use("/auth", authRoutes);

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

**Role Based Authentication**

## Create Role-Based Middleware

Create a middleware to **check user roles**:

**`authMiddleware.js`**

```
import jwt from "jsonwebtoken";

const SECRET_KEY = "your_secret_key"; // Must match the key used in
AuthService

// Middleware to verify JWT token
export const authenticateUser = (req, res, next) => {
    const authHeader = req.headers["authorization"];
    if (!authHeader || !authHeader.startsWith("Bearer ")) {
        return res.status(401).json({ error: "Access denied. No token
provided." });
    }

    const token = authHeader.split(" ")[1];

    try {
        const decoded = jwt.verify(token, SECRET_KEY);
        req.user = decoded; // Attach user data to request
        next();
    } catch (error) {
        res.status(401).json({ error: "Invalid token." });
    }
};

// Middleware to check if user has the required role
export const authorizeRole = (requiredRoles) => {
    return (req, res, next) => {
        if (!req.user || !requiredRoles.includes(req.user.role)) {
            return res.status(403).json({ error: "Access denied.
Insufficient permissions." });
        }
        next();
    };
};
```

## Protect Routes with Role-Based Middleware

### `adminRoutes.ts` (Protected Admin Routes)

```
import express from "express";
import { authenticateUser, authorizeRole } from
"../middleware/authMiddleware";

const router = express.Router();
```

```
// Only "admin" role can access this route
router.get("/admin-dashboard", authenticateUser, authorizeRole(["admin"]),
(req, res) => {
    res.json({ message: "Welcome, Admin!", user: req.user });
});

export default router;
```

## Assignment: Role-Based Authentication in a Library Management System

### Scenario:

You are developing a **Library Management System** for a university. The system has two types of users:

1. **Admin** - Can add, update, and remove books.
2. **Student** - Can search, view, borrow, and return books.

To ensure security and access control, the system must implement **Role-Based Authentication and Authorization** using **Node.js, Express, JWT, and SQL Server**.