Practice 3 - Role-Based Access Control

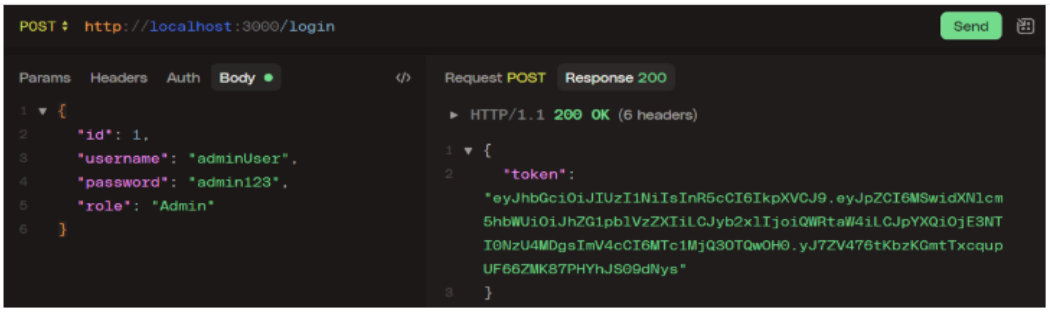
Instruction:
Perform the following tasks in the Nimbus.

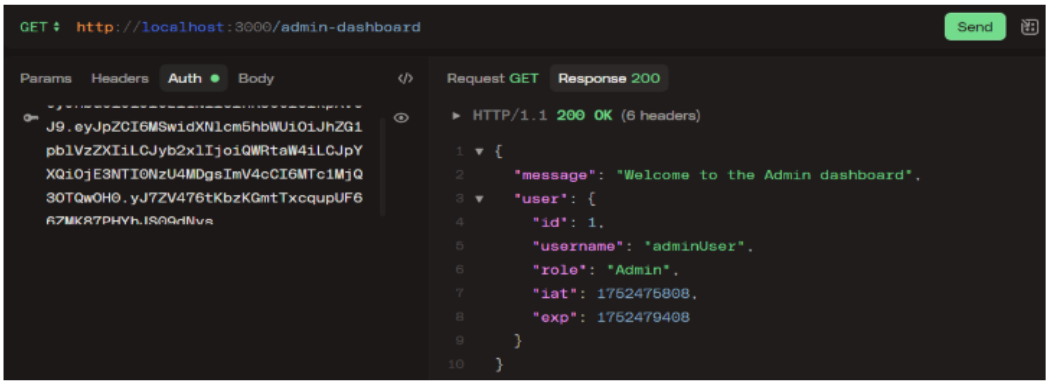
Title: Build Role-Based Access Control with Admin, User, and Moderator Roles

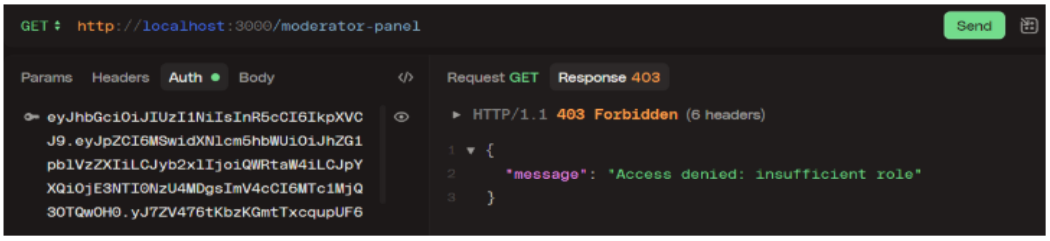
Objective:
Learn how to implement role-based access control (RBAC) in a Node.js and Express application to restrict access to certain routes or actions based on user roles. This task helps you understand how to store and check user roles in JWT tokens, create flexible authorization logic, and secure sensitive parts of your backend API.

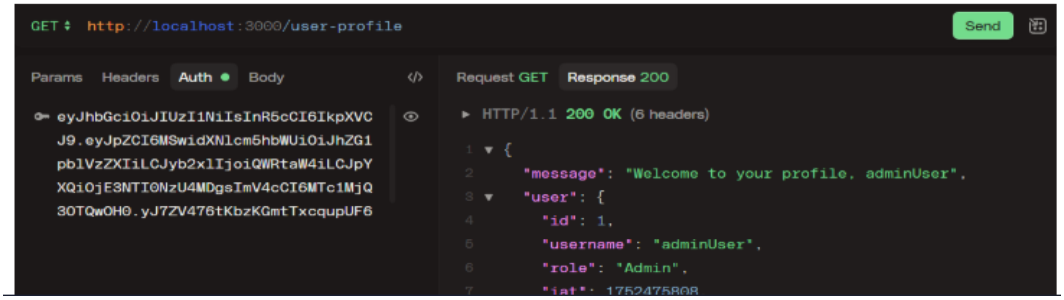
Task Description:
Create a Node.js and Express.js backend that supports user authentication with different roles, including Admin, User, and Moderator. Implement a login route that issues JWT tokens containing the user's role in the payload. Create middleware to verify the JWT and extract the user's role. Implement separate protected routes that can only be accessed by specific roles, for example, an Admin-only dashboard route, a Moderator management route, and a general User profile route. Ensure that requests with invalid tokens or insufficient roles are properly denied with clear error messages. Test your system by logging in with different roles and accessing each route to confirm that the role-based restrictions work as expected.

Expected Output:









Node.js Code:

```
// Import required modules
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

const SECRET_KEY = 'mysecretkey';

// Sample users
const users = [
  { id: 1, username: 'adminUser', password: 'admin123', role: 'Admin' },
  { id: 2, username: 'moderatorUser', password: 'mod123', role: 'Moderator' },
  { id: 3, username: 'normalUser', password: 'user123', role: 'User' }
];
// Login route
```

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username && u.password === password);

  if (user) {
    const token = jwt.sign({ id: user.id, username: user.username, role: user.role }, SECRET_KEY, { e
    res.json({ token });
  } else {
    res.status(401).json({ message: 'Invalid credentials' });
  }
});

// Middleware for verifying token
function verifyToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'Token missing' });

  jwt.verify(token, SECRET_KEY, (err, decoded) => {
    if (err) return res.status(403).json({ message: 'Invalid token' });
    req.user = decoded;
    next();
  });
}

// Role-based access middleware
function authorizeRoles(...allowedRoles) {
  return (req, res, next) => {
    if (!allowedRoles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Access denied: insufficient role' });
    }
    next();
  };
}

// Admin route
app.get('/admin-dashboard', verifyToken, authorizeRoles('Admin'), (req, res) => {
  res.json({ message: 'Welcome to the Admin dashboard', user: req.user });
});

// Moderator route
app.get('/moderator-panel', verifyToken, authorizeRoles('Moderator'), (req, res) => {
  res.json({ message: 'Welcome to the Moderator panel', user: req.user });
});

// User route
app.get('/user-profile', verifyToken, authorizeRoles('User', 'Admin', 'Moderator'), (req, res) => {
  res.json({ message: `Welcome to your profile, ${req.user.username}`, user: req.user });
});

// Start the server
app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```