**Assignment 9: User Authentication Basics**

**Learn about JWT (JSON Web Tokens) for authentication.**

**Implement basic user authentication using JWT in Express.js.**

JSON Web Tokens (JWT) are a standard for representing claims between two parties. They are commonly used for authentication and information exchange between a client and a server. Below, I'll guide you through implementing basic user authentication using JWT in an Express.js application.

Step 1: Install Required Packages

npm install express mongoose body-parser jsonwebtoken bcrypt

jsonwebtoken: For creating and verifying JSON Web Tokens.

bcrypt: For hashing user passwords.

Step 2: Update app.js

| const express = require('express');  const mongoose = require('mongoose');  const bodyParser = require('body-parser');  const jwt = require('jsonwebtoken');  const bcrypt = require('bcrypt');  const app = express();  const port = 3000;  // Connect to MongoDB  mongoose.connect('mongodb://localhost:27017/todo-app', { useNewUrlParser: true, useUnifiedTopology: true });  // Check MongoDB connection  const db = mongoose.connection;  db.on('error', console.error.bind(console, 'MongoDB connection error:'));  db.once('open', () => {  console.log('Connected to MongoDB');  });  // Use body-parser middleware for parsing JSON  app.use(bodyParser.json());  // Define a simple data model for a User  const userSchema = new mongoose.Schema({  username: { type: String, unique: true, required: true },  password: { type: String, required: true },  });  const User = mongoose.model('User', userSchema);  // JWT secret key (change this to a more secure value in a production environment)  const jwtSecret = 'your-secret-key';  // Middleware to verify JWT  const verifyToken = (req, res, next) => {  const token = req.headers['authorization'];  if (!token) {  return res.status(403).json({ message: 'Token not provided' });  }  jwt.verify(token, jwtSecret, (err, decoded) => {  if (err) {  return res.status(401).json({ message: 'Failed to authenticate token' });  }  req.userId = decoded.id;  next();  });  };  // Routes  // Register a new user  app.post('/register', async (req, res) => {  const { username, password } = req.body;  if (!username || !password) {  return res.status(400).json({ message: 'Username and password are required' });  }  try {  const hashedPassword = await bcrypt.hash(password, 10);  const user = new User({ username, password: hashedPassword });  await user.save();  res.status(201).json({ message: 'User registered successfully' });  } catch (error) {  res.status(400).json({ message: error.message });  }  });  // Login and get a JWT  app.post('/login', async (req, res) => {  const { username, password } = req.body;  if (!username || !password) {  return res.status(400).json({ message: 'Username and password are required' });  }  try {  const user = await User.findOne({ username });  if (!user || !(await bcrypt.compare(password, user.password))) {  return res.status(401).json({ message: 'Invalid credentials' });  }  const token = jwt.sign({ id: user.\_id }, jwtSecret, { expiresIn: '1h' });  res.json({ token });  } catch (error) {  res.status(500).json({ message: error.message });  }  });  // Protected route  app.get('/protected', verifyToken, (req, res) => {  res.json({ message: 'This is a protected route', userId: req.userId });  });  // Start the server  app.listen(port, () => {  console.log(`Server listening at http://localhost:${port}`);  }); |
| --- |

Step 3: Test User Authentication

Register a User:

Send a POST request to http://localhost:3000/register with JSON data containing a username and password.

Example:

{

"username": "testuser",

"password": "testpassword"

}

Login and Get JWT:

Send a POST request to http://localhost:3000/login with the same credentials.

Retrieve the JWT token from the response.

Access Protected Route:

Send a GET request to http://localhost:3000/protected with the JWT token in the Authorization header.

Example:

Authorization: Bearer YOUR\_JWT\_TOKEN

Verify that you can access the protected route.

This example demonstrates the basics of user authentication using JWT in an Express.js application. In a real-world scenario, you would typically store user information securely, handle password reset functionality, and use HTTPS in a production environment. Additionally, you might want to explore more advanced authentication mechanisms and consider using libraries like Passport.js for more extensive functionality.