Beginner's Guide to Building a Simple API with Passport.

Introduction

In this tutorial, we will walk through the process of using Passport.js to protect a GET request in an Express.js application. Passport.js is a popular authentication middleware for Node.js and Express.js, making it easy to authenticate users in your web applications.

Prerequisites

Before we get started, make sure you have the following:

- Node.js and npm installed
- Basic knowledge of JavaScript

Step 1: Setting Up Your Project

Create a new directory for your project and initialize it as a Node.js project:

```
mkdir passport-protect-get
cd passport-protect-get
npm init -y
```

Step 2: Setting Up Middleware and Routes

- **Install Dependencies**: Install the necessary packages:

```
npm install express passport passport-local express-session
```

- **Create Your Express Application (app.js)**:

Create an app.js file and set up middleware for your Express application:

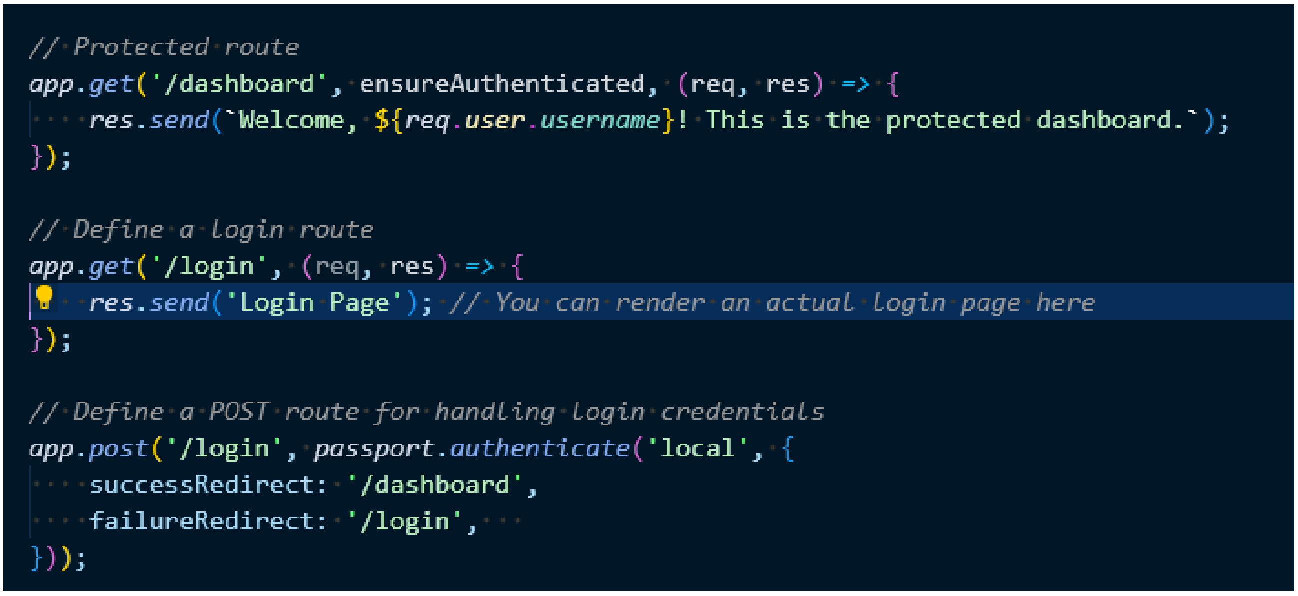
```
const express = require('express');
const session = require('express-session');
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const app = express();
```

```
// Middleware setup
app.use(express.urlencoded({ extended: true }));

// Session middleware setup with options on separate lines
app.use(
  session({
    secret: 'your-secret-key',
    resave: false,
    saveUninitialized: false
  })
);

app.use(passport.initialize());
app.use(passport.session());

// Import Passport.js configuration from passport-config.js
require('./passport-config')(passport);
```

Step 3: Setting Up Routes

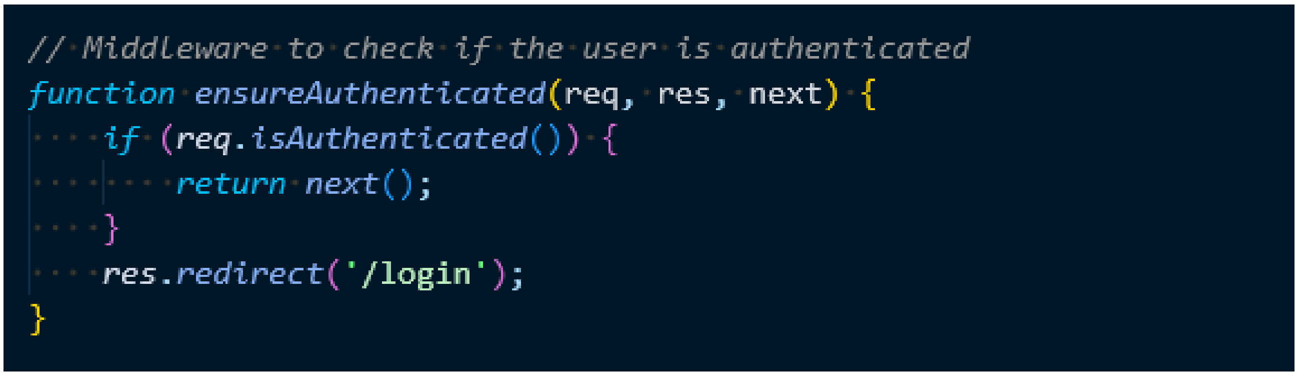
Create Your Express Application (app.js) - Setting Up Routes:

- Set up routes for your Express application:



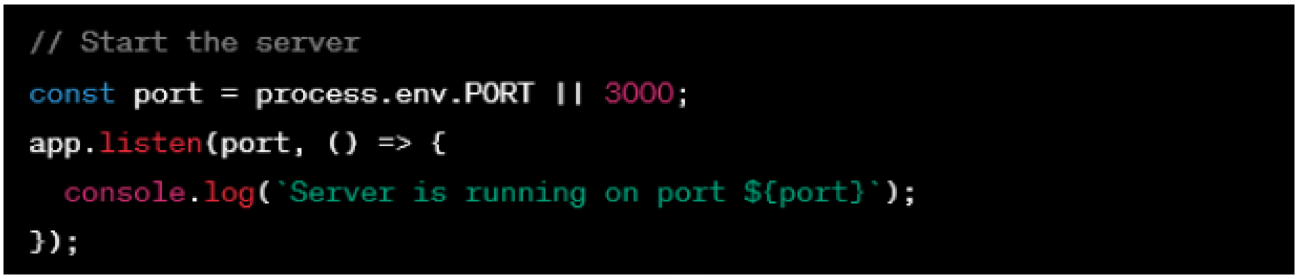
```
// Protected route
app.get('/dashboard', ensureAuthenticated, (req, res) => {
    res.send(`Welcome, ${req.user.username}! This is the protected dashboard.`);
});

// Define a login route
app.get('/login', (req, res) => {
    res.send('Login Page'); // You can render an actual login page here
});

// Define a POST route for handling login credentials
app.post('/login', passport.authenticate('local', {
    successRedirect: '/dashboard',
    failureRedirect: '/login',
}));
```

- Add Middleware to check autheticated route



```
// Middleware to check if the user is authenticated
function ensureAuthenticated(req, res, next) {
    if (req.isAuthenticated()) {
        return next();
    }
    res.redirect('/login');
}
```

Start server



```
// Start the server
const port = process.env.PORT || 3000;
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

Step 4: Creating Passport Configuration (passport-config.js)

Create Passport Configuration: Create a passport-config.js file for Passport.js configuration:

```
const LocalStrategy = require('passport-local').Strategy;

// Dummy user for demonstration (replace with your authentication logic)
const user = {
  id: 1,
  username: 'user',
  password: 'password',
};

module.exports = function (passport) {
  // Configure Passport.js for local authentication
  passport.use(new LocalStrategy((username, password, done) => {
    if (username === user.username && password === user.password) {
      return done(null, user);
    } else {
      return done(null, false, { message: 'Incorrect username or password' })
    }
  }));

  passport.serializeUser((user, done) => {
    done(null, user.id);
  });

  passport.deserializeUser((id, done) => {
    if (id === user.id) {
      done(null, user);
    } else {
      done({ message: 'User not found' }, null);
    }
  });
};
```

Step 5: Test Secure REST Endpoints

Step 5.1 Install and Open Postman

If you don't have Postman installed, you can download it from the official website. Once installed, open Postman.

Step 5.2: Test Authentication Routes

To test authentication routes (e.g., login and logout), follow these steps:

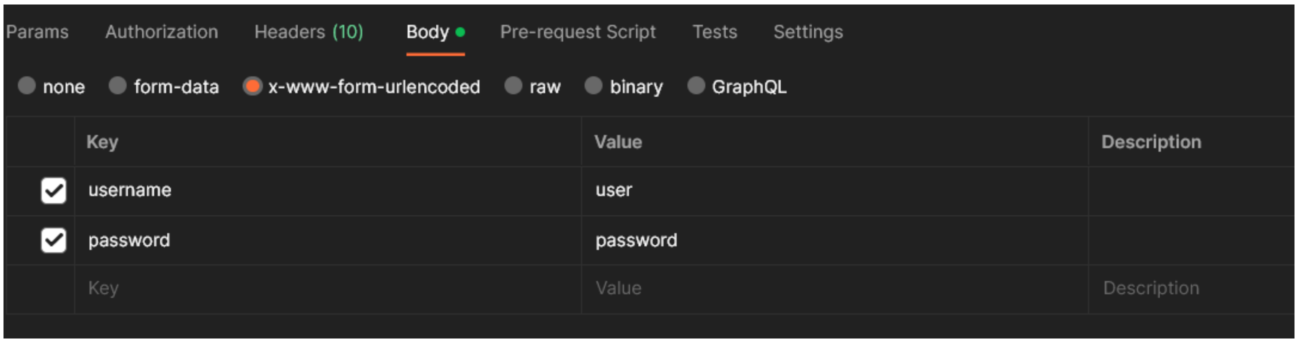
- **Login Route**:

Create a new request in Postman.

Set the request method to POST.

Enter the URL for your login route (e.g., http://localhost:3000/login).

In the request body, select **x-www-form-urlencoded** and add the username and password fields with the appropriate values.



	Key	Value	Description
✓	username	user	
✓	password	password	
	Key	Value	Description

Step 5.3: Test Protected Routes

To test protected routes (e.g., /dashboard), follow these steps:

- **Protected Route**:

Create a new request in Postman.

Set the request method to GET.

Enter the URL for your protected route (e.g., http://localhost:3000/dashboard).

Send the request.

If you have successfully logged in using the login route, you should be able to access the protected route and receive a response indicating that you are authenticated. If you haven't logged in, you should receive an authentication error or a redirection response.

Postman allows you to test various scenarios by sending HTTP requests to your application's routes. Make sure you customize the request URLs and parameters according to your application's routes and authentication logic.

Conclusion:

In this tutorial, we've demonstrated how to protect a GET request using Passport.js in an Express.js application. Passport.js simplifies the authentication process and allows you to secure routes easily. By following these steps, you can implement authentication in your web application and restrict access to specific routes based on user authentication status.