Manual Authentication

Introduction

In this section, we are going to learn about Authentication. Authentication means establishing your identity. First, we will learn how to establish the identity manually then we will look at the drawbacks { extra code that we have to write }. Then we will be using a library for authentication.

The steps we will be looking at in this module -

- We send the username and password to the server.
- The server will receive the data and verify the identity from the database where all the data is being stored.
- After that, It will create a token that will establish the user identity and send it back to the browser that stores it.
- Whenever the browser sends another request for fetching some data, it will send the stored token alongside every request.
- The server will identify the identity of the user with that request.
- The browser will then serve user-specific data
- The browser will delete the token on sign-out that will lose the user identity.

-Setting up User Schema

- We have to create a user inside the models folder by creating a new file { **user.js** }.
- For creating a schema we have to import mongoose inside the { **user.js** } file.
- Inside the user schema, we need to define different fields. Hence, we will define different properties inside the fields as an object.
- Whenever you create a new object the database should store a field called **created at** and whenever you update that object the database should store a field called **updated at** which gets updated. These two fields are managed by the mongoose itself using the **timestamps** property.
- We have to export the { **user.js** } file.

{ user.js }

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

const usersConrtoller = require('../controllers/users_controller');

router.get('/profile', usersConrtoller.profile);


module.exports = router;
```

NOTE - You have to create two pages as your mini assignment - { Signing up the user and Signing in the user }

Rendering Pages for Sign up and Sign in

- We will create two files inside the views folder - { **user_sign_in.ejs** } and { **user_sign_up.ejs** }.
- To render these pages we need to have some controllers.
- In { **users_controller.js** } we have to create some actions for the sign-up page and sign-in page.
- We have to render the { **user_sign_up.ejs** } and { **user_sign_in.ejs** } files though actions.
- Inside the routes folder in { **users.js** } file, we have to create routes for the two pages.
- The type of request for both pages will be a **GET** request.

Explaining Cookies

- A cookie is a file that is stored by the browser. It is sent with every request to the server and the server sends back the same file to the browser.
- This file can be edited at the browser or the server level.
- It is used for a lot of purposes in terms of storing data related to the user, establishing the user's identity, or storing the product that the user has browsed in.
- A cookie is a file that contains key-value pairs.
- Browser is the environment in which the cookies reside. Thus, if free space is filled by the cookies in the browser, it will automatically clear all the cookies present over there.

- The key-value pair is stored in an encrypted manner.

Creating and Alternating a Cookie

- For reading and writing into cookies, we will be using a library called cookie-parser.
- Install the library using the command { npm install cookie-parser }.
- Require the installed library in the **{ index.js file }**.
- We have to tell the app to use the library in the middleware.
- We have to set up the cookie parser using the app. use method. { app.use(cookieParser() }.

{ index.js file }

```
app.use(express.urlencoded());

app.use(cookieParser());

app.use(express.static('./assets'));

app.use(expressLayouts);
```

User Sign Up

- We will create the user through the signup form that we have created and save that user in the database to establish the identity of the user.
- We will allow that user to be authenticated.
- Once that user is authenticated, the next step would be to show the details of the user on the profile page.
- We will remove the cookie after the user signs out of our webpage.
- There can be two cases-
    - If the user already exists, we do not need to recreate his data.
    - If the user does not exist in the database, we will store all the data we have collected in the database.

{ users_controller.js }

```
// render the sign up page
module.exports.signUp = function(req, res){
    return res.render('user_sign_up', {
        title: "Codeial | Sign Up"
    })
}


// render the sign in page
module.exports.signIn = function(req, res){
    return res.render('user_sign_in', {
        title: "Codeial | Sign In"
    })
}

// get the sign up data
module.exports.create = function(req, res){
    if (req.body.password != req.body.confirm_password){
        return res.redirect('back');
    }

    User.findOne({email: req.body.email}, function(err, user){
        if(err){console.log('error in finding user in signing up'); return}

        if (!user){
            User.create(req.body, function(err, user){
                if(err){console.log('error in creating user while signing up'); return}

                return res.redirect('/users/sign-in');
            })
        }else{
            return res.redirect('back');
        }

    });
}
```

User Sign In

- We have to establish the identity of the user of the system by signing in.
- We have to check whether the user exists.
    - If the user exists, we have to check whether the password entered is correct or not.
- We have to match the password entered by the user to the password present in the database.
- If both the passwords match, then we will store the user's identity in the cookie and send it to the browser.
- If both the passwords don't match we will redirect the user back to the sign-in page.

{ **user_controllers.js** }

```
// sign in and create a session for the user
module.exports.createSession = function(req, res){

    // steps to authenticate
    // find the user
    User.findOne({email: req.body.email}, function(err, user){
        if(err){console.log('error in finding user in signing in'); return}
        // handle user found
        if (user){

            // handle password which doesn't match
            if (user.password != req.body.password){
                return res.redirect('back');
            }

            // handle session creation
            res.cookie('user_id', user.id);
            return res.redirect('/users/profile');

        }else{
            // handle user not found

            return res.redirect('back');
        }


    });
```

Show Details of Signed in User

- We have to show the user's information on the profile page.
- The user should be able to access the profile page only when sign-in is complete.
- If the user is not authenticated, then they should be redirected back to the sign-in page.
- We have to check if the user-id is present inside the cookies.
    - If yes, then we have to find the user.
- If the user is not found, redirect the user to the sign-in page.
- If the user is found, then render the user to the { **user_profile** } page which will contain all the information about the user.

{ user_controller }

```
const User = require('../models/user');


module.exports.profile = function(req, res){
    if (req.cookies.user_id){
        User.findById(req.cookies.user_id, function(err, user){
            if (user){
                return res.render('user_profile', {
                    title: "User Profile",
                    user: user
                })
            }else{
                return res.redirect('/users/sign-in');

            }
        });
    }else{
        return res.redirect('/users/sign-in');

    }


}
```

EXTRA:
Create the Sign-out page as your mini assignment

Summarizing Manual Authentication

- We first created two pages which were the { sign_in } and { sign_out } pages with some definite set of fields
- We have created some actions in the **{ users_controller .js }** to get data from the sign-in and sign-up forms, and also some actions to render the two pages.
- Four actions were created - 1. Two of them are used to render the two pages ({ sign_in } and { sign_out })
  2. The rest two are used to get data from the two pages ({ sign_in } and { sign_out }).

- When we want to sign up the user, we have to check whether the string entered in the field **Password** and **Confirm Password** are the same or not.
- We try to find the use. If the user is not found, we return from the database.
- In case the user is not found, this means we need to create a user. Post creation, we redirect him to the sign-in page
- Once the user is created we need the user to sign in.
- We tried to find the user. If the user is found, we try to match the password.
- If the password matches, we redirect the user to the profile page.
- Before redirecting to the profile page, we set up the user id in the cookie.
- In the action that shows the profile page, we tried to find the user by the user id in the cookie.
    - If the user is found we show the information of the user on the profile page.
    - If the user is not found then redirect the user to the sign-in page.