CODER HOUSE

Clase 25. Programación Backend

Autorización y Autenticación

OBJETIVOS DE LA CLASE

- Incorporar los conceptos de autenticación y autorización y sus métodos más usados.
- Comprender las diferencias entre ambos conceptos.
- Conocer el módulo Passport de Node y sus mecanismos.
- Comprender en detalle el módulo passport-local.

CRONOGRAMA DEL CURSO

Clase 24

Cookies, Sesiones, storages: Parte II

Clase 25

Autorización y Autenticación

Clase 26

Estrategias de autenticación con redes sociales

CODER HOUSE

AUTORIZACIÓN VS. AUTENTICACIÓN

CODER HOUSE

Autenticación

- Es el proceso de identificación de usuarios para asegurarse su identidad.
- Existen diversos métodos para probar la autenticación, siendo la contraseña el más conocido y utilizado.
- Parte del principio de que si el usuario dispone de las credenciales requeridas (por ejemplo, nombre de usuario y contraseña), el sistema puede validar la identidad del usuario y permitir el acceso a los recursos solicitados.

Autorización

- Define la información, los servicios y recursos del sistema a los que podrá acceder el usuario autenticado.
- Uno de sus usos más comunes es para generar distintos permisos para el usuario común y el administrador, quienes tendrán acceso a distintos tipo de recursos.
- Existen distintos métodos para autorizar usuarios.
  Suele utilizarse el método mediante **middlewares**, donde permitan el acceso según el tipo de usuario autenticado (admin, cliente, etc.).

Resumiendo...

Autorización:
verifica los permisos que corresponden a cada identidad.

Autenticación:
verifica las identidades, por diferentes métodos (algo que sabemos, algo que tenemos, algo que somos).

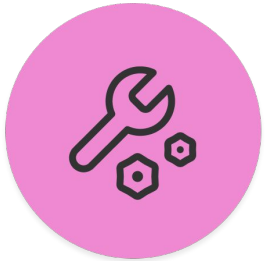
CODER HOUSE

Métodos de autenticación

- **Usuario y contraseña:** Es el método tradicional más utilizado, donde el usuario ingresa *username* o *email* y *password* para autenticarse.
- **Sin contraseña (*passwordless*):** Consiste en que, cada vez que queramos iniciar sesión a un recurso, se nos enviará al email un enlace que nos permitirá acceder sin necesidad de contraseña.
- **Por redes sociales:** Varias aplicaciones nos dan como opción iniciar sesión directamente con alguna red social. La ventaja principal es que se usan directamente los datos de esa cuenta social para hacer el inicio de sesión.
- **Datos biométricos:** Autentica usuarios mediante huellas dactilares.

Métodos de autenticación

- **JWT(*JSON Web Token*):** Este método *open source* permite la transmisión segura de datos entre las distintas partes. Comúnmente se utiliza para la autorización a partir de un par de claves que contiene una clave privada y una pública.
- **OAuth 2.0:** Permite que mediante una API, el usuario se autentique y acceda a los recursos del sistema que necesita.



INICIO DE SESIÓN CON USERNAME Y PASSWORD

Tiempo: 15 minutos

CODER HOUSE

Desafío genérico

Inicio de Sesión con Username y Password

Basado en un proyecto express que almacene sesiones de usuario, realizar un sistema que:

1) Tenga una vista de registro de usuario (nombre, password y dirección) que almacene dicha información en un *array* en memoria.

2) Posea un formulario de *login* (nombre y password) para permitir a los usuarios registrados a acceder a su información.

3) Si accede un usuario no registrado o las credenciales son incorrectas, el servidor enviará un error (puede ser a través de un objeto plano o de una plantilla).

4) Si se quiere registrar un usuario que ya está registrado, el servidor enviará un error (puede ser a través de un objeto plano o de una plantilla).

Desafío generico

Inicio de Sesión con Username y Password

5) En el caso de que sea válido el *login*, se iniciará una sesión de usuario y se mostrarán los datos completos del usuario en una ruta específica (/datos).
👉 *Se puede mostrar la información a través de un objeto plano o de una plantilla.*

6) Implementar el cierre de sesión en una ruta '/logout' que puede llamar desde la barra de dirección del browser, o desde un botón en la misma plantilla de datos.

7) Esa ruta '/datos' sólo estará disponible en caso de estar en una sesión de usuario activa. caso contrario, se redireccionará a la vista de login.

8) Como extra podemos implementar un contador de visitas, que se muestre sobre la vista de datos.

NOTA: no utilizar passport.

CODER HOUSE

PASSPORT

CODER HOUSE

¿De qué se trata?

- Passport es un *middleware* de autenticación de NodeJS.
- Cumple únicamente la función de autenticar solicitudes, por lo que delega todas las demás funciones a la aplicación. *Esto mantiene el código limpio y fácil de mantener.*
- Passport reconoce los distintos métodos de login utilizados actualmente, por lo que sus mecanismos de autenticación se empaquetan como módulos individuales. Entonces, no es necesario crear dependencias que no se vayan a utilizar.
- Cada uno de estos mecanismos se llaman ***strategies***.

Strategies

- Cada *strategy* tiene un módulo distinto de NodeJS para instalar.
- Las *strategy* disponibles son:
  - ***passport-local*** para autenticación de usuarios mediante nombre de usuario y contraseña.
  - ***passport-openid*** para autenticación mediante OpenId (estándar abierto para la autenticación federada).
  - ***passport-oauth*** para autenticación mediante API de otros proveedores como de redes sociales.

PASSPORT LOCAL

CODER HOUSE

CONFIGURACIÓN INICIAL

CODER HOUSE

Empezar a utilizar Passport-local

Ejemplo en vivo

En primer lugar debemos instalar el módulo *passport* y el de *passport-local*

```
$ npm install passport
```

```
$ npm install passport-local
```

Además, debemos instalar todas las otras dependencias que se muestran a continuación.

```
"dependencies": {
  "bcrypt": "^5.0.1",
  "express": "^4.17.1",
  "express-handlebars": "^5.3.3",
  "express-session": "^1.17.2",
  "mongoose": "^6.0.5",
  "passport": "^0.4.1",
  "passport-local": "^1.0.0"
}
```

CODER HOUSE

Requerir los módulos

Ejemplo en vivo

- Se requiere el módulo de *passport*, junto con el módulo de *passport-local*, que nos da control para implementar manualmente el mecanismo de autenticación.

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

CODER HOUSE

Configurar LocalStrategy de login

Ejemplo en vivo

```
passport.use('login', new LocalStrategy(
  (username, password, done) => {
    User.findOne({ username }, (err, user) => {
      if (err)
        return done(err);

      if (!user) {
        console.log('User Not Found with username '+ username);
        return done(null, false);
      }

      if (!isValidPassword(user, password)) {
        console.log('Invalid Password');
        return done(null, false);
      }

      return done(null, user);
    });
  })
);
```

- Se define una nueva instancia de LocalStrategy y se la carga mediante el método passport.use( ).
- El primer parámetro es el nombre de la strategy (“login” en este caso) y el segundo es una instancia de la estrategia que se desea usar (LocalStrategy en este caso)
- LocalStrategy espera encontrar por defecto las credenciales de usuario en los parámetros nombre de usuario ‘*username*’ y contraseña ‘*password*’ (si se definen con otros nombres, no los encontrará!)

Configurar LocalStrategy de login

Ejemplo en vivo

```
passport.use('login', new LocalStrategy(
  (username, password, done) => {
    User.findOne({ username }, (err, user) => {
      if (err)
        return done(err);

      if (!user) {
        console.log('User Not Found with username '+ username);
        return done(null, false);
      }

      if (!isValidPassword(user, password)) {
        console.log('Invalid Password');
        return done(null, false);
      }

      return done(null, user);
    });
  })
);
```

- Buscamos el usuario por su *username* en la base de datos mediante User.findOne( ).
- Utilizamos el *callback* de verificación *done* en el *return* para devolver lo que corresponda.
- Si el usuario se encuentra en la base de datos y es válido se devuelve en el *done* : *null* (indicando que no hubo error) y el usuario encontrado *user*.
- La función *isValidPassword* es:

```
function isValidPassword(user, password) {
 return bCrypt.compareSync(password, user.password);
}
```

CODER HOUSE

Ejemplo en vivo

Configurar LocalStrategy de signup

Para crear la instancia de *strategy* para el registro de nuevo usuario, es similar al de login. La diferencia es que primero chequeamos si ya existe o no ese usuario.

- ➔ Si no existe, creamos un usuario nuevo y lo guardamos en la base de datos.
- ➔ Si ya existe, devolvemos un mensaje que lo informe, dando error al registrar.

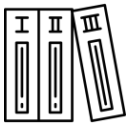
Configurar LocalStrategy de signup

Ejemplo en vivo

```
passport.use('signup', new LocalStrategy({
 passReqToCallback: true
},
 (req, username, password, done) => {
   User.findOne({ 'username': username }, function
(err, user) {

     if (err) {
       console.log('Error in SignUp: ' + err);
       return done(err);
     }

     if (user) {
       console.log('User already exists');
       return done(null, false)
     }

     const newUser = {
       username: username,
       password: createHash(password),
       email: req.body.email,
       firstName: req.body.firstName,
       lastName: req.body.lastName,
     }
```

```
     User.create(newUser, (err, userWithId) => {
       if (err) {
         console.log('Error in Saving user: ' +
err);
         return done(err);
       }
       console.log(user)
       console.log('User Registration succesful');
       return done(null, userWithId);
     });
   });
 })
)

function createHash(password) {
  return bCrypt.hashSync(
           password,
           bCrypt.genSaltSync(10),
           null);
}
```

CODER HOUSE



Serializar y deserializar

- Para restaurar el estado de autenticación a través de solicitudes HTTP, Passport necesita serializar usuarios y deserializarlos fuera de la sesión. Esto se hace de modo que cada solicitud subsiguiente no contenga las credenciales del usuario anterior.
- Se suele implementar proporcionando el ID de usuario al serializar y consultando el registro de usuario por ID de la base de datos al deserializar.
- Los métodos que proporciona Passport para esto son *serializeUser* y *deserializeUser*.

Serializar y deserializar

- El código ejemplo de ambos métodos se muestra a continuación.
- Se puede ver que el método *serializeUser* utiliza el id del usuario y el *deserializeUser* utiliza el objeto de usuario, como lo mencionamos antes.

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

passport.deserializeUser((id, done) => {
  User.findById(id, done);
});
```

BREAK

¡5/10 MINUTOS Y VOLVEMOS!

INICIALIZACIÓN, RUTAS Y CONTROLLERS

CODER HOUSE

Iniciar passport

```
const app = express();

app.use(session({
 secret: 'keyboard cat',
 cookie: {
   httpOnly: false,
   secure: false,
   maxAge: config.TIEMPO_EXPIRACION
 },
 rolling: true,
 resave: true,
 saveUninitialized: false
}));

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

- Debemos inicializar con app.use( ) express y express-session.
- Además, debemos inicializar passport como se muestra en el código.

CODER HOUSE

Definir las rutas

```
//  INDEX
app.get('/', routes.getRoot);

//  LOGIN
app.get('/login', routes.getLogin);
app.post('/login',  passport.authenticate('login', { failureRedirect: '/faillogin' }), routes.postLogin);
app.get('/faillogin', routes.getFaillogin);

//  SIGNUP
app.get('/signup', routes.getSignup);
app.post('/signup', passport.authenticate('signup', { failureRedirect: '/failsignup' }), routes.postSignup);
app.get('/failsignup', routes.getFailsignup);


//  LOGOUT
app.get('/logout', routes.getLogout);

//  FAIL ROUTE
app.get('*', routes.failRoute);
```

Definimos las rutas de *index*, *login*, *singup*, *logout* y *fail route*. En las rutas por *post* de *login* y *signup*, en las que se procesan los datos ingresados en los formularios, utilizamos como middleware el método *authenticate* de *passport*, con el nombre de la LocalStrategy configurada como primer parámetro, y a dónde redirigir en caso de falla como segundo.

Métodos definidos en las rutas

```
//  INDEX
function getRoot(req, res) {
  res.send("Bienvenido")
}

//  LOGIN
function getLogin(req, res) {
  if (req.isAuthenticated()) {
      let user = req.user;
      console.log('user logueado');
      res.render('profileUser', {user});
  }
  else {
    console.log('user NO logueado');
    res.render('login')
  }
}

// SIGNUP
function getSignup(req, res) {
    res.render('signup')
}
```

A continuación, está el código de ejemplo para el controller de los métodos de las rutas que definimos en la diapositiva anterior.

- Observamos que las rutas por get muestran una vista o un mensaje.
- En *getLogin* primero verifica si ya está logueado, mediante el método ***isAuthenticated*** del *request* ***req*** que nos da *passport*.

Continúa en la siguiente diapositiva.

Métodos definidos en las rutas

```
// PROCESS LOGIN
function postLogin (req, res) {
  var user = req.user;
  res.render('profileUser')
}

// PROCESS SIGNUP
function postSignup (req, res) {
  var user = req.user;
  res.render('profileUser')
}

function getFaillogin (req, res) {
  console.log('error en login');
  res.render('login-error', {});
}

function getFailsignup (req, res) {
  console.log('error en signup');
  res.render('signup-error', {});
}
```

```
// LOGOUT
function getLogout (req, res) {
  req.logout();
  res.render('index')
}

function failRoute(req, res){
  res.status(404).render('routing-error', {});
}
```

- Las rutas por *post* solo muestran una vista ya que el inicio de sesión en sí lo realiza directo *passport* con el middleware *passport.authenticate*.
- Para el *getLogout* se utiliza el método ***logout*** del *request* ***req*** que nos da *passport*.

RUTAS PROTEGIDAS

CODER HOUSE

Autorizar rutas protegidas

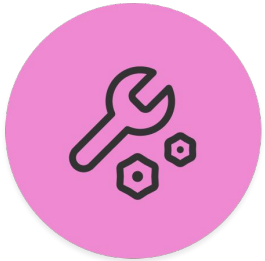
- Mediante *middlewares*, podemos proteger distintas rutas, de modo que solo se pueda acceder si hay un usuario logueado.
- Para esto, usamos nuevamente req.isAuthenticated( ). Si existe, entonces podemos continuar mediante *next( )*. Si no existe, redirigimos al *login*.

```
function checkAuthentication(req,res,next){
  if(req.isAuthenticated()){
      //req.isAuthenticated() will return true if user is logged in
      next();
  } else{
      res.redirect("/login");
  }
}
```

Autorizar rutas protegidas

En la o las ruta/s que queremos proteger, se agrega el middleware que vimos en la diapositiva anterior. Queda entonces, como se muestra en el siguiente código.

```
app.get('/ruta-protegida',checkAuthentication, (req,res) => {
  //do something only if user is authenticated
  var user = req.user;
  console.log(user);
  res.send('<h1>Ruta OK!</h1>');
});
```



INICIO DE SESIÓN CON PASSPORT-LOCAL

Tiempo: 15 minutos

CODER HOUSE

Desafío
generico

Inicio De Sesión Con Passport-local

Tiempo: 15 minutos

Realizar el lo visto en el anterior segmento, en esta ocasión utilizando passport con *LocalStrategy* para realizar todas las funciones que se piden.

No hace falta encriptar las contraseñas ni usar base de datos, todo puede residir en memoria del servidor: usuarios y sesiones.

CODER HOUSE

¿PREGUNTAS? 

¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Conceptos y diferencias de Autenticación y Autorización.
- Passport y sus mecanismos.
- Passport-local en detalle.

OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE