Hacé click acá para dejar tu feedback sobre esta clase.

Hacé click acá completar el quiz teórico de esta lecture.

Authentication

Autenticación VS Autorización

Es normal pensar que autenticación y autorizacion son sinónimos. De hecho, ambos son procesos de seguridad, aunque tienen propósitos diferentes.

README.md

La autenticación es el proceso de identificar a los usuarios y garantizar que los mismos sean quienes dicen ser. Cuál podría ser una prueba de autenticación? La más utilizada es la contraseña. A la hora de iniciar sesión, si conocen sus credenciales (nombre de usuario y contraseña), el sistema entenderá que su identidad es válida. En consecuencia,van a poder acceder al recurso o conjunto de recursos que quieran.

Por otro lado, la autorización es lo que define a qué recursos dentro de ese sistema autenticado van a poder acceder. Que hayan logrado pasar la instancia de la autenticación, no significa que van a poder utilizar el sistema por completo. Un ejemplo sería autenticarse en la página de un banco. Una autenticación exitosa no les va a otorgar la capacidad de ver las cuentas de otros clientes ni de retirar dinero de las mismas. En cambio, para un ejecutivo de cuentas, que inicia sesión como administrador, es habitual acceder a muchas cuentas y realizar acciones adicionales que un usuario, como cliente, no podría.

Autenticación

Durante este tiempo, estuvimos trabajando con protocolos HTTP. Estos protocolos tienen una particularidad, y es que son stateless, osea que no tienen conocimiento sobre estados. Y eso qué significa? que cada vez que enviemos una request, esta se va a ejecutar, pero si refrescamos la página todos esos datos se van a perder. Es por esto que vamos a necesitar herramientas que **guarden** la sesión, es decir, que nos permitan como usuarios permanecer logueados una vez que enviemos nuestros datos.

Cookies

Una de las herramientas que nos van a permitir poder guardar esa información son las cookies. Una cookie (o galleta informática) es un pequeño archivo de datos creado por el sitio web visitado y almacenado en el navegador o dispositivo utilizado por los usuarios. Las cookies contienen pequeñas cantidades de información que se envían entre un emisor (generalmente el servidor de la web) y un receptor (el navegador del usuario) almacenando esta información en la memoria del navegador. El objetivo de este proceso es que la web visitada pueda comprobar esa cookie en una futura conexión del usuario y utilizarla.

- **Cookies de sesión (Session Cookies)**

Comprenden el tiempo que transcurre entre el inicio y el cierre de una sesión. Al iniciarla, el servidor genera un "ID de sesión" que se transmite al cliente. Este ID, también denominado identificador de sesión, es un número generado al azar que las cookies almacenan de manera temporal. Su único fin es asignar al usuario una sesión en particular. Este identificador tiene una gran ventaja: si nosotros como usuarios abrimos varias ventanas en el mismo sitio web, estas se van a asignar a una sola sesión. Esto nos va a permitir que iniciemos varias consultas de forma simultánea sin que se pierda información personal importante. Una vez cerrada esta sesión de navegación, tanto el identificador, como el resto de datos almacenados, se borran.

- **Cookies persistentes**

Opuestas a las cookies de sesión, las cookies persistentes son aquellas que siguen almacenadas en el navegador durante más tiempo incluso después de la sesión (pueden ser horas, meses o años).

Tokens de Autenticación

Por otro lado, la autenticación basada en tokens es un protocolo que nos va a permitir verificar nuestra identidad y, a cambio, recibir un token de acceso único. Durante la vida del token, los usuarios accedemos al sitio web o la aplicación para la que se emitió, en lugar de tener que volver a ingresar las credenciales cada vez que volvemos a ingresar a la misma página web, aplicación o cualquier recurso protegido con ese mismo token.

Lo pueden pensar como un boleto sellado. Básicamente, vamos a conservar el acceso mientras el token siga siendo válido. Una vez que cerramos sesión o salimos de una aplicación, el token se invalida.

La autenticación basada en tokens es diferente de las técnicas de autenticación tradicionales basadas en contraseña o en servidor. Los tokens ofrecen una segunda capa de seguridad y los administradores tienen un control detallado sobre cada acción y transacción.

JWT

Un JSON Web Token es un token de acceso estandarizado en el RFC 7519 que permite el intercambio seguro de datos entre dos partes. Contiene toda la información importante sobre una entidad, lo que implica que no hace falta consultar una base de datos ni que la sesión tenga que guardarse en el servidor (sesión sin estado).

Por este motivo, los JWT son especialmente populares en los procesos de autentificación. Con este estándar es posible cifrar mensajes cortos, enviarles información sobre el remitente y demostrar si este cuenta con los derechos de acceso requeridos. Los propios usuarios solo entran en contacto con el token de manera indirecta: por ejemplo, al introducir el nombre de usuario y la contraseña en una interfaz. La comunicación como tal entre las diferentes aplicaciones se lleva a cabo en el lado del cliente y del servidor.

Token JWT

En la práctica, se trata de una cadena de texto que tiene tres partes codificadas en Base64, cada una de ellas separadas por un punto, como en el siguiente ejemplo:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IlNveSBIZW5y
```

Podemos utilizar un Debugger Online para decodificar ese token y visualizar su contenido. Si accedemos al mismo y pegamos dentro el texto completo, se nos mostrará lo que contiene:

Como dijimos, un token tiene tres partes:

- **Header**: encabezado dónde se indica, al menos, el algoritmo y el tipo de token, que en el caso del ejemplo anterior era el algoritmo HS256 y un token JWT.
- **Payload**: donde aparecen los datos de usuario y privilegios, así como toda la información que queramos añadir, todos los datos que creamos convenientes.
- **Signature**: una firma que nos permite verificar si el token es válido. La firma se construye de tal forma que vamos a poder verificar que el remitente es quien dice ser, y que el mensaje no se ha modificado en el camino.

Passport

Y cómo vamos a implementar entonces todo esto? Ahí es donde entra Passport! Passport es una librería de Node que nos va a dar la capacidad de autenticarnos. El único propósito de Passport es autenticar solicitudes, lo que hace a través de un conjunto extensible de complementos conocidos como estrategias. Y qué son las estrategias? Son diferentes mecanismos de autenticación que nos ofrece Passport para no tener que acudir a dependencias innecesarias. Por ejemplo, pueden autenticarse en una instancia de base de datos local/remota o utilizar el inicio de sesión único de proveedores de OAuth para Facebook, Twitter, Google, etc. La API es simple: le proporcionamos una solicitud de autenticación y Passport proporciona enlaces para controlar lo que ocurre cuando esta tiene éxito o falla.

Modo de Uso

Primero que nada vamos a instalar Passport.

```
npm install passport
```

Lo seteamos en nuestro archivo raíz. Acá vamos a requerir la librería y la inicializamos junto con su middleware de autenticación de sesión.

```
const passport = require('passport');

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

Antes de autenticar las solicitudes, configuramos la estrategia (o estrategias) que vamos a usar.

```
passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) {
        return done(null, false, { message: 'Incorrect username.' });
      }
      if (!user.validPassword(password)) {
        return done(null, false, { message: 'Incorrect password.' });
      }
      return done(null, user);
    });
  }
));
```

Para autenticar solicitudes vamos a usar passport.authenticate () y especificar qué estrategia queremos implementar. De forma predeterminada, si la autenticación falla, Passport nos va a responder con un estado 401 No autorizado y no va a invocar a ningún controlador de ruta adicional. Si la autenticación es exitosa, se invocará el siguiente controlador y la propiedad req.user se va a establecer en el usuario autenticado.

```
app.post('/login',
  passport.authenticate('local'),
  function(req, res) {
    // If this function gets called, authentication was successful.
    // `req.user` contains the authenticated user.
    res.redirect('/users/' + req.user.username);
  });
```

Pueden encontrar más información sobre Passport y sus estrategias en passportjs.org

Homework

Completa la tarea descrita en el archivo README