¿Qué es Passport.js?

Passport.js es un middleware de autenticación para Node.js que facilita la implementación de estrategias de autenticación en aplicaciones web. Su diseño modular permite que se integre fácilmente con diferentes métodos de autenticación, como:

- **Autenticación basada en credenciales (usuario/contraseña)**
- **Autenticación con redes sociales (OAuth, como Google, Facebook, Twitter, etc.)**
- **Autenticación mediante JWT (JSON Web Tokens)**
- **Autenticación con OpenID Connect**

Passport.js se centra en proporcionar una estructura mínima y flexible, lo que permite que se pueda personalizar según las necesidades de cada proyecto.

¿Cómo funciona Passport.js?

Passport.js se basa en el concepto de **estrategias** (strategies). Cada estrategia define una forma específica de autenticar a un usuario. El flujo básico es el siguiente:

1. **Configuración de Passport.js:**
   Se inicializa Passport en la aplicación Express utilizando `passport.initialize()` y se define una o más estrategias de autenticación.

2. **Configuración de la estrategia:**
   Se define una estrategia que maneje la autenticación (por ejemplo, `LocalStrategy` para autenticación con usuario/contraseña).

3. **Serialización/Deserialización del usuario:**
   Passport necesita serializar el usuario en una sesión para que las solicitudes posteriores puedan reconocerlo.

4. **Middleware de protección de rutas:**
   Passport proporciona el middleware `passport.authenticate()` para proteger rutas específicas.

Explicación del flujo

✅ `passport.initialize()` → Inicializa Passport.js.
✅ `passport.session()` → Habilita el manejo de sesiones para persistir la autenticación.
✅ `passport.use()` → Define la estrategia de autenticación que se utilizará.
✅ `passport.serializeUser()` → Almacena información mínima del usuario en la sesión.
✅ `passport.deserializeUser()` → Recupera el usuario desde la sesión en cada solicitud.
✅ `passport.authenticate()` → Middleware que gestiona el flujo de autenticación.

Ventajas de usar Passport.js

✅ Flexible y modular, permitiendo múltiples estrategias.
✅ Compatible con Express.js y otros frameworks basados en Node.js.
✅ Permite integrar autenticación social como Google, Facebook, etc.
✅ Cuenta con una amplia variedad de estrategias listas para usar.

Comando para instalar passport y estrategia local

npm install passport passport-local

¿Qué es la serialización y deserialización en Passport.js?

En **Passport.js**, la **serialización** y **deserialización** son procesos clave para gestionar la autenticación persistente, es decir, mantener al usuario autenticado en múltiples solicitudes HTTP.

¿Por qué se necesita la serialización/deserialización?

HTTP es un protocolo **sin estado**, lo que significa que cada solicitud se procesa de forma independiente. Para recordar qué usuario está autenticado entre solicitudes, Passport utiliza sesiones.

- **Serialización:** Guarda información mínima del usuario en la sesión (normalmente el `id`).
- **Deserialización:** Recupera la información completa del usuario a partir de los datos almacenados en la sesión.

`passport.serializeUser()`

- Se ejecuta inmediatamente después de que la autenticación sea exitosa.
- Recibe el objeto `user` autenticado y decide qué dato almacenar en la sesión.
- Normalmente se guarda solo el `id` del usuario para minimizar el tamaño de la sesión.

`passport.deserializeUser()`

- Se ejecuta en cada solicitud posterior, siempre que el usuario esté autenticado.
- Toma el `id` almacenado en la sesión y recupera el objeto completo del usuario desde la base de datos u otra fuente.

Buenas prácticas

✅ En `serializeUser()`, guarda solo información mínima (como el `id`) para que las sesiones sean ligeras.
✅ En `deserializeUser()`, recupera solo los datos necesarios del usuario para cada solicitud.
✅ Considera manejar errores en ambas funciones para evitar fallos silenciosos.