



Clase 26. Programación Backend

Estrategias de autenticación con redes sociales



OBJETIVOS DE LA CLASE

- Conocer acerca del inicio de sesión mediante redes sociales.
- Realizar el inicio sesión con Twitter.
- Incorporar el módulo JSON Web Token y realizar el inicio de sesión.

CRONOGRAMA DEL CURSO

Clase 25



**Autorización y
autenticación**

Clase 26



**Estrategias de
autenticación con redes
sociales**

Clase 27



**Proceso principal del
servidor**

PASSPORT: ESTRATEGIAS DE AUTENTICACIÓN CON REDES SOCIALES

PASSPORT-TWITTER



TwitterStrategy



- La estrategia de Twitter permite a los usuarios iniciar sesión en una aplicación web utilizando su cuenta de Twitter.
- Internamente, la autenticación de Twitter funciona con **OAuth 1.0a**.
- El soporte para Twitter se implementa mediante el módulo ***passport-twitter***.
- Es prácticamente igual a lo que vimos con Facebook.



Twitter for Developers

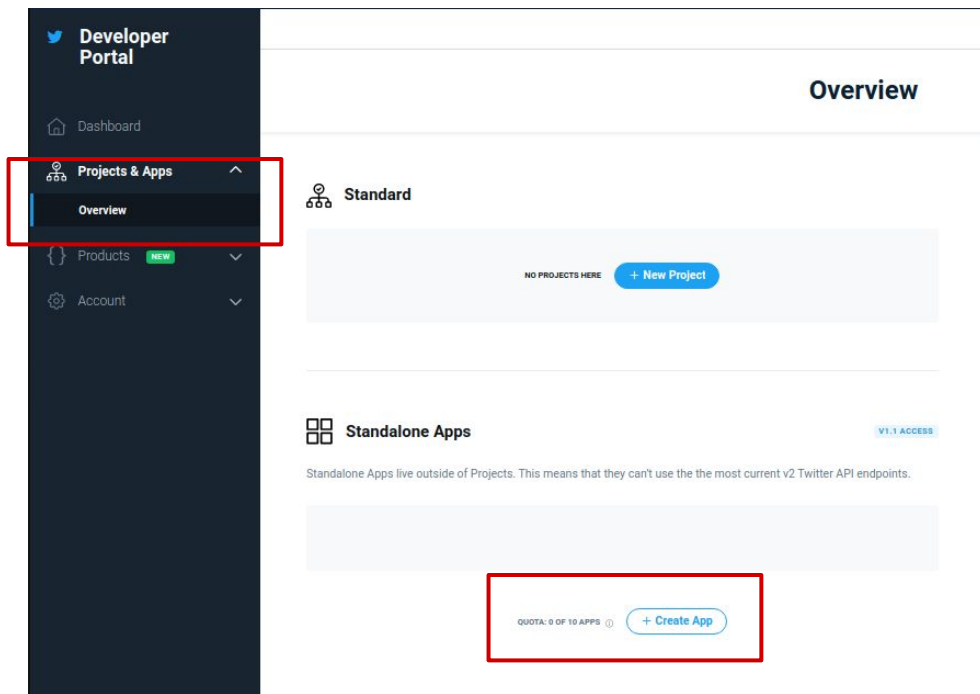


- Para habilitar la autenticación a través de Twitter, primero tenemos que crear una Twitter App utilizando el Twitter Developers.
- Una vez creada la App, necesitamos los datos de App ID y App Secret asignados a la app. Debemos además, especificar una URL para redireccionar al usuario una vez que inicia sesión con Twitter.
(De igual forma que hicimos con Facebook)
- Para empezar a crear esta app, debemos primero ingresar con nuestra cuenta de Twitter al [siguiente link](#)



Crear Twitter App

1. Al ingresar con una cuenta válida de Twitter, nos pedirá seguir unos los pasos, completar unos formularios requeridos y finalmente verificar nuestro email.
2. En la barra lateral elegir *Overview* (dentro de *Projects & Apps*) y clickear el botón *+ Create App*.







Crear Twitter App

4. Elegir el nombre para la app.
5. Finalmente, nos muestra el API Key y el API Secret Key que es lo que necesitamos para la configuración.

Here are your keys & tokens

For security, this will be the last time we'll display these. If something happens, you can always regenerate them.

API Key ⓘ
zD9oHrZQMjbilyLVyHBrj4JLR 

API Secret Key ⓘ
urXi6FitafBRlgxHsBt1sn2JzcOLQ0blVjNjAPvZaNeMvfypmN 

Crear Twitter App



Developer Portal

- Dashboard
- Projects & Apps
- Overview
- STANDALONE APPS
- CoderHouseLoginDemo**
- Products NEW
- Account

CoderHouseLoginDemo

Settings Keys and tokens

App Details [Edit](#)

NAME
CoderHouseLoginDemo

APP ICON

APP ID
21960311

DESCRIPTION
This app was created to use the Twitter API.
This information will be visible to people who've authorized your app

App permissions [Edit](#)

Read Only
Read Tweets and profile information

Authentication settings [Edit](#)

3-legged OAuth is **disabled**

Use 3-legged OAuth for Sign in with Twitter, posting Tweets on behalf of other accounts and more. Get more information in the [docs](#).

6. Adicionalmente, debemos configurar la aplicación para que pueda ser usada para autenticación.

Crear Twitter App



Edit authentication settings

Enable 3-legged OAuth

Use 3-legged OAuth for Sign in with Twitter, posting Tweets on behalf of other accounts and more. Get more information in the [docs](#).



Callback URLs (required) ⓘ

<http://localhost:8080/auth/twitter/callback>

[+ Add another](#)

Website URL (required)

<https://twitter.com/drmaquino>

7. También debemos configurar la aplicación para que pueda ser usada para autenticación...

Crear Twitter App



drmaquino

Authorize CoderHouseLoginDemo to access your account?



CoderHouseLoginDemo

twitter.com/drmaquino

This app was created to use the Twitter API.

Authorize app

Cancel

This application will be able to:

- See Tweets from your timeline (including protected Tweets) as well as your Lists and collections.
- See your Twitter profile information and account settings.
- See accounts you follow, mute, and block.

Learn more about third-party app permissions in the [Help Center](#).

We recommend reviewing the app's terms and privacy policy to understand how it will use data from your Twitter account. You can revoke access to any app at any time from the [Apps and sessions](#) section of your Twitter account settings.

By authorizing an app you continue to operate under Twitter's [Terms of Service](#). In particular, some usage information will be shared back with Twitter. For more, see our [Privacy Policy](#).

7. Por último, debemos configurar la aplicación para que pueda ser usada para autenticación...y permitirle a la aplicación acceder a nuestros datos.

CODER HOUSE



Empezar a usar passport-twitter

En primer lugar, instalamos el módulo de passport-twitter.

```
$ npm install passport-twitter
```

Se requiere el módulo de *passport*, y además, se define la *TwitterStrategy*, requiriendo el módulo *passport-twitter* como se muestra en la imagen.

```
const passport = require('passport')  
const TwitterStrategy = require('passport-twitter').Strategy;
```

Configurar passport-twitter



- De igual forma que en el caso de Facebook, utilizamos **passport.use** para configurar el módulo.
- El primer parámetro es el objeto con la Key, Secret Key y el *callbackURL* que es la ruta a la que redirige luego del *login*.
- Luego, está el *callback* de verificación que busca el usuario en la base de datos. En este, el parámetro *profile* contiene la información de usuario provista por Twitter.

```
passport.use(new TwitterStrategy({
  consumerKey: TWITTER_CONSUMER_KEY,
  consumerSecret: TWITTER_CONSUMER_SECRET,
  callbackURL: "http://www.example.com/auth/twitter/callback"
}),
function(token, tokenSecret, profile, done) {
  User.findOrCreate(profile.id, function(err, user) {
    if (err) { return done(err); }
    done(null, user);
  });
});
```



Configurar las rutas

- Se requieren **dos rutas** para la autenticación de Twitter. La primera inicia una transacción OAuth y redirige al usuario a Twitter. La segunda es la URL a la que Twitter redirigirá al usuario después de que haya iniciado sesión.
- En ambas, utilizamos el método **passport.authenticate**, especificando que se trata de Twitter(va como primer parámetro del método).

```
app.get('/auth/twitter', passport.authenticate('twitter'));  
  
app.get('/auth/twitter/callback',  
  passport.authenticate('twitter', { successRedirect: '/',  
                                     failureRedirect: '/login' }));
```

👉 Hay que tener en cuenta que la URL de la ruta de devolución de llamada (segunda ruta) debe coincidir con la de la opción *callbackURL* especificada al configurar *TwitterStrategy* en la diapositiva anterior.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

JWT ***(JSON Web Token)***



¿De qué se trata?

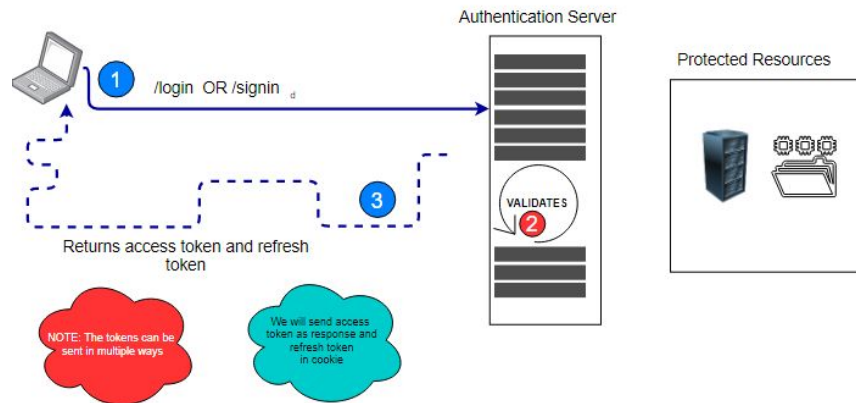
- JSON Web Token es un método estándar y abierto para representar reclamaciones de forma segura entre dos partes.
- [JWT.IO](https://jwt.io) nos permite decodificar, verificar y generar JWT.
- Básicamente, los JWT son cadenas de datos que se pueden utilizar para autenticar e intercambiar información entre un servidor y un cliente.



Flujo de funcionamiento

El flujo de funcionamiento es el siguiente:

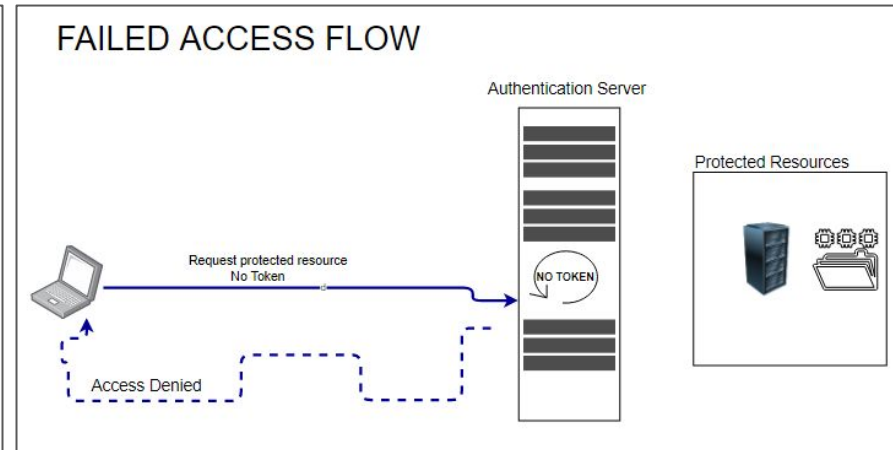
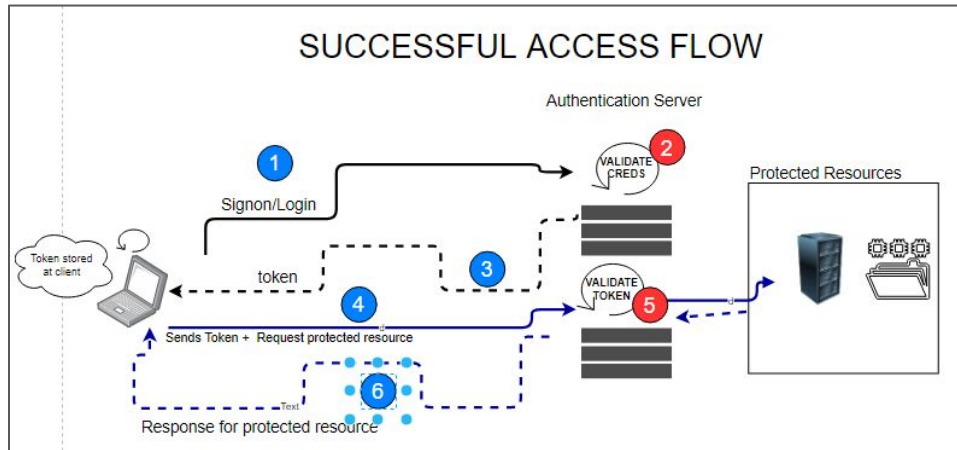
1. El cliente envía credenciales al servidor.
2. El servidor verifica las credenciales, genera un JWT y lo envía como respuesta.
3. Las solicitudes posteriores del cliente tienen un JWT en los *headers* de la solicitud.
4. El servidor valida el token y, si es válido, proporciona la respuesta solicitada.





Flujo de funcionamiento

- Las solicitudes posteriores del cliente tienen un JWT en los *headers* de la solicitud.
- El servidor valida el token y, si es válido, proporciona la respuesta solicitada.
- Si no se valida el token, se niega el acceso.





Empezar a utilizarlo

El módulo para crear y decodificar los tokens JWT es jsonwebtoken.

```
$ npm install jsonwebtoken
```

Luego, requerimos en el o los archivos que lo vamos a utilizar, como muestra el código.

```
const jwt = require("jsonwebtoken");  
const PRIVATE_KEY = "myprivatekey";
```

La constante ***PRIVATE_KEY*** puede tener cualquier *string*. Se usa para encriptar y desencriptar los datos.



Generar el token

- La función ***generateToken*** recibe como parámetro un usuario y devuelve el token. En otras palabras, inicia sesión.
- Se utiliza el método ***jwt.sign***.
- El tercer parámetro es el tiempo hasta que expire ese token, es decir, el tiempo que la sesión va a permanecer iniciada como máximo.

```
function generateToken (user) {  
  const token = jwt.sign({ data: user }, PRIVATE_KEY, { expiresIn: '24h' });  
  return token;  
}
```

Ruta de registro



```
// REGISTER
app.post('/register', (req, res) => {

  const { nombre, password, direccion } = req.body

  const yaExiste = usuarios.find(usuario => usuario.nombre == nombre)
  if (yaExiste) {
    return res.json({ error: 'ya existe ese usuario' });
  }

  const usuario = { nombre, password, direccion }

  usuarios.push(usuario)

  const access_token = generateToken(usuario)

  res.json({ access_token })
})
```

- Primero se chequea que no exista el usuario.
- Si no existe, se crea un usuario nuevo, y se guarda en la BD.
- Luego, si está todo bien, se genera un token, que llama a la función **generateToken** y le pasa el usuario creado como parámetro.

Ruta de login



```
// LOGIN
app.post('/login', (req, res) => {

  const { nombre, password } = req.body

  const usuario = usuarios.find(u => u.nombre == nombre && u.password ==
password)
  if (!usuario) {
    return res.json({ error: 'credenciales invalidas' });
  }

  const access_token = generateToken(usuario)

  res.json({ access_token })
})
```

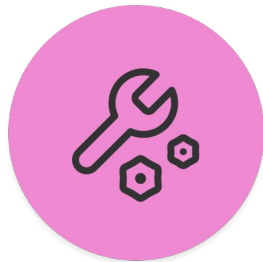
- Busca el usuario en la BD. Si lo encuentra, chequea la contraseña.
- Si coinciden las contraseñas, se genera el token, que llama a la función **generateToken** y le pasa el usuario encontrado como parámetro para iniciar sesión.



Middleware de verificación

- Este middleware verifica que exista un token, y si existe, trae los datos de ese usuario.
- Lo usamos para autorizar ciertas rutas a ciertos usuarios.
- Extrae el token desde el encabezado de la petición (generalmente del campo **authorization**, y generalmente precedido por la palabra 'Bearer' y un espacio).
- Si existe, entonces utiliza **jwt.verify** para poder obtener los datos del usuario, que luego los guarda en **req.user**.

```
function auth(req, res, next) {  
  const authHeader = req.headers.authorization;  
  
  if (!authHeader) {  
    return res.status(401).json({  
      error: 'not authenticated'  
    });  
  }  
  
  const token = authHeader.split(' ')[1];  
  
  jwt.verify(token, PRIVATE_KEY, (err, decoded) => {  
    if (err) {  
      return res.status(403).json({  
        error: 'not authorized'  
      });  
    }  
  
    req.user = decoded.data;  
    next();  
  });  
};
```



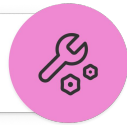
INICIO DE SESIÓN CON JWT

Tiempo: 15 minutos

Inicio de Sesión con JWT

Tiempo: 15 minutos

Desafío
generico



Utilizando la estructura de un servidor Nodejs express, realizar el siguiente desarrollo implementando las sesiones de usuario a través de JWT (JSON Web Token). Dichas sesiones tendrán un tiempo de expiración de 1 minuto.

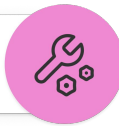
Requerimientos del sistema:

1. Tenga un formulario de registro de usuario (nombre, password y dirección) que almacene dicha información en un array en memoria.
2. Un formulario de login (nombre y password) para permitir a los usuarios registrados iniciar una sesión.
3. Si accede un usuario no registrado ó los credenciales no corresponden, el servidor enviará un error (puede ser a través de un objeto plano o de una plantilla).

Inicio de Sesión con JWT

Tiempo: 15 minutos

Desafío
generico

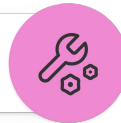


4. Si se quiere registrar un usuario que ya está registrado, el servidor enviará un error (puede ser a través de un objeto plano o de una plantilla).
5. Al cargar la página principal ('/'), si existe una sesión iniciada, se mostrarán los datos del usuario en cuestión (obtenidos mediante una consulta con el token debidamente adjunto en el encabezado de la petición de datos). Caso contrario, se lo redirigirá a una página que le informe que no posee autorización, y le ofrecerá ir a la página de login.
6. Implementar el cierre de sesión con un botón *logout* en la página de datos de usuario, que redirige la vista al formulario de login.
7. Como extra podemos implementar un contador de visitas, que se muestre sobre la vista de datos.

Inicio de Sesión con JWT

Tiempo: 15 minutos

Desafío
generico



Nota:

- No utilizar passport.
- No utilizar session de express.
- Priorizar la funcionalidad del backend (especialmente el manejo de firma y verificación de tokens) antes que el desarrollo de las vistas.
- Se puede usar fetch para hacer la petición a la API de datos (adjuntando la cabecera necesaria para la autenticación)
- También se pueden usar plantillas públicas (handlebars) para la carga dinámica de la página de datos, dependiendo del resultado de la autenticación.



INICIO DE SESIÓN

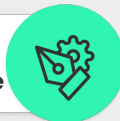
Retomemos nuestro trabajo para agregar inicio de sesión a nuestro sitio.

INICIO DE SESIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



>> Consigna:

Implementar sobre el entregable que venimos realizando un mecanismo de autenticación. Para ello:

Se incluirá una vista de registro, en donde se pidan email y contraseña. Estos datos se persistirán usando MongoDB, en una (nueva) colección de usuarios, cuidando que la contraseña quede encriptada (sugerencia: usar la librería *bcrypt*).

Una vista de login, donde se pida email y contraseña, y que realice la autenticación del lado del servidor a través de una estrategia de passport local.

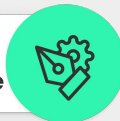
Cada una de las vistas (logueo - registro) deberá tener un botón para ser redirigido a la otra.

INICIO DE SESIÓN

Formato: link a un repositorio en Github con el proyecto cargado.

Sugerencia: no incluir los node_modules

Desafío
entregable



Una vez logueado el usuario, se lo redirigirá al inicio, el cual ahora mostrará también su email, y un botón para desolguearse.

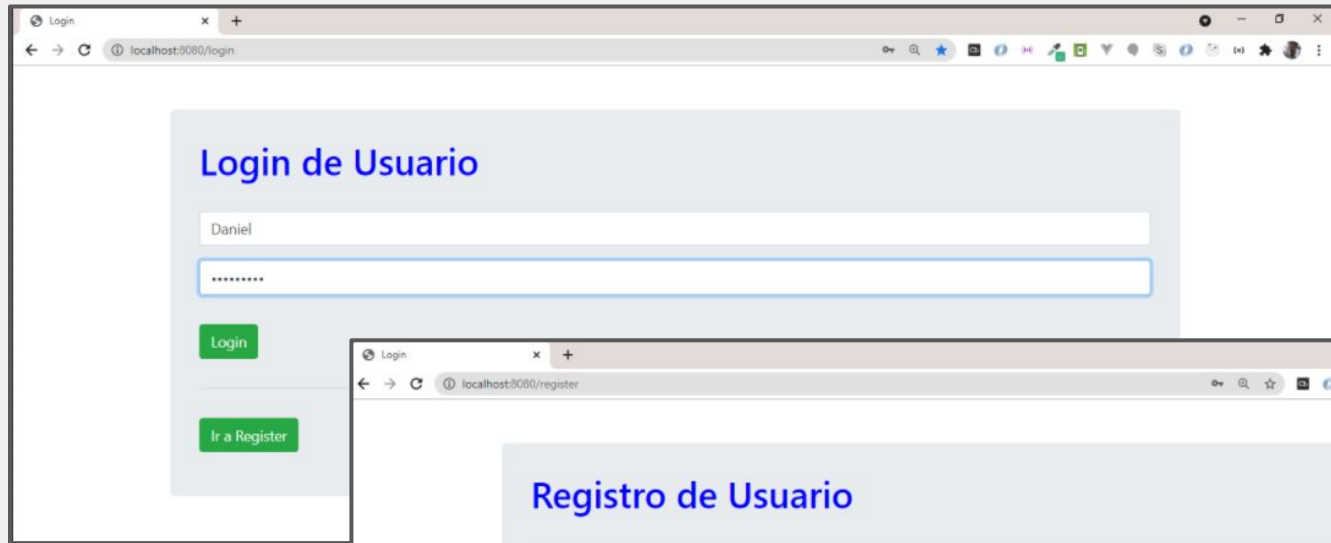
Además, se activará un espacio de sesión controlado por la sesión de passport. Esta estará activa por 10 minutos y en cada acceso se recargará este tiempo.

Agregar también vistas de error para login (credenciales no válidas) y registro (usuario ya registrado).

El resto de la funciones, deben quedar tal cual estaban el proyecto original.

>> Ejemplos de vistas de acceso a continuación.

>> Ejemplo

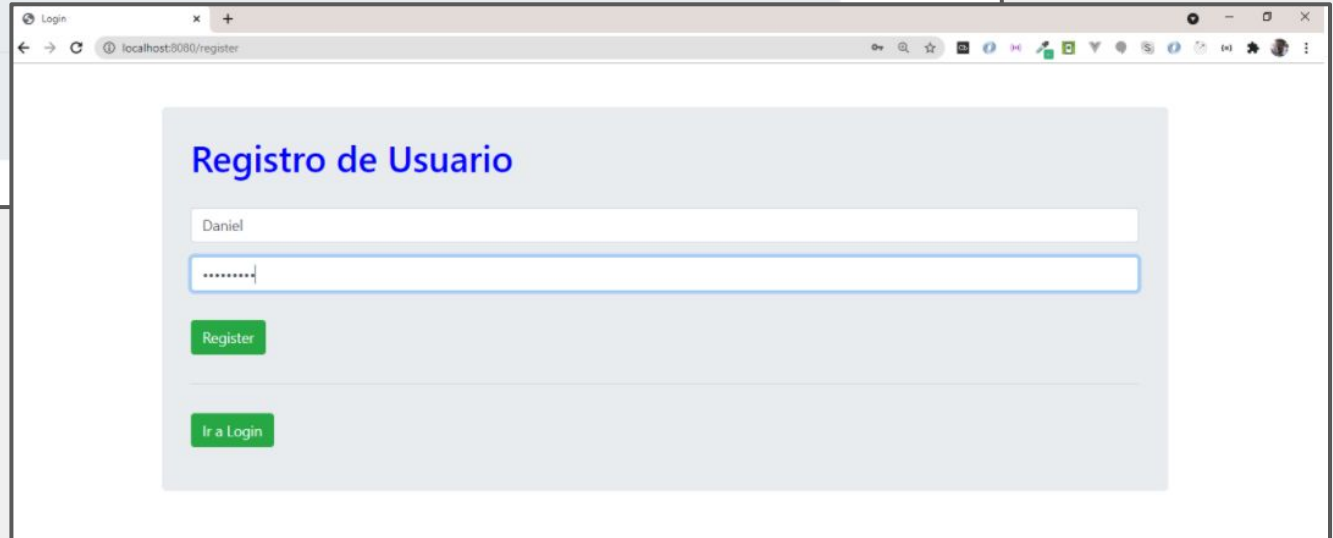


Login de Usuario

Daniel

Login

Ir a Register



Registro de Usuario

Daniel

Register

Ir a Login

Vista de Productos x +

localhost:8080/login

Bienvenido Daniel [Desloguear](#)

Ingrese Producto

Nombre

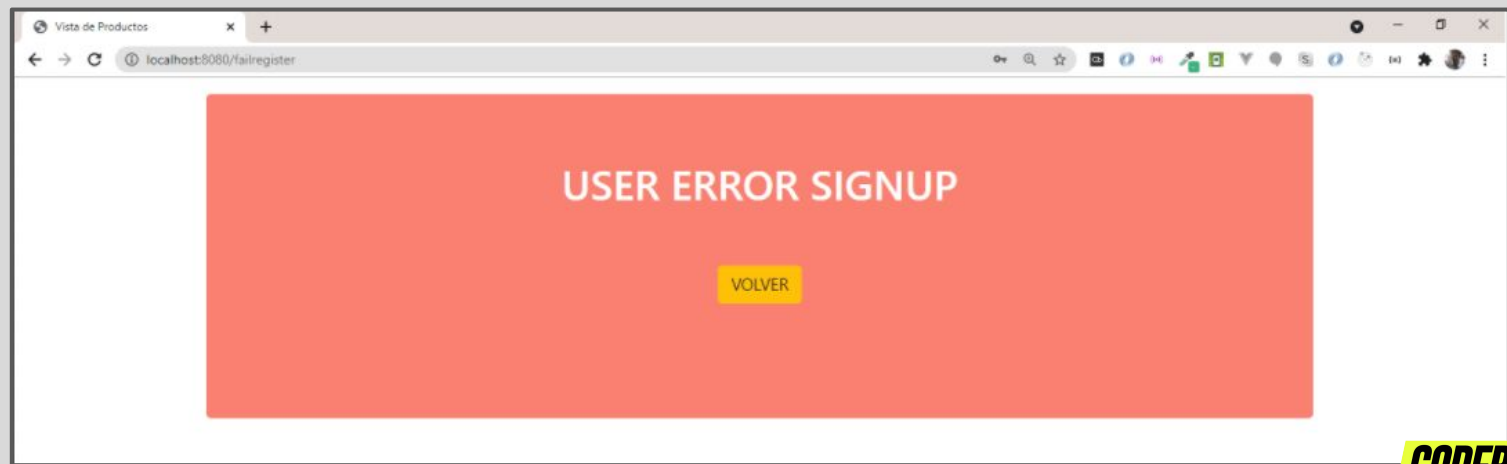
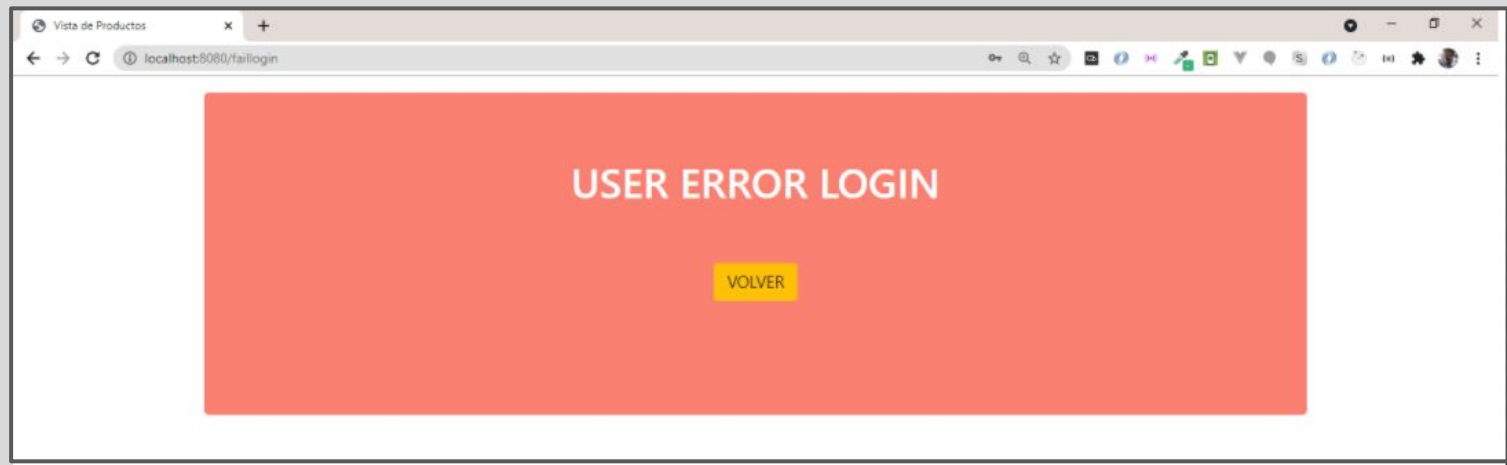
Precio

Foto URL

[Enviar](#)

Lista de Productos

Nombre	Precio	Foto
--------	--------	------



¿PREGUNTAS?



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Passport-Facebook.
- Passport-Twitter.
- JWT.



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE