



[Nueva guía] Descargue la Guía 2017 de Microservicios: Rom...

[Descargar la guía](#) ▶

# Cree un SPA seguro con Spring Boot y OAuth

por **Brian Demers** MVB · 28 y 17 de noviembre · Zona de Java

Pruebe [Okta](#) para agregar el inicio de sesión social, el MFA y el soporte de OpenID Connect a su aplicación Java en minutos. Cree una [cuenta de desarrollador gratuita](#) hoy y nunca vuelva a crear auth.

---

Incluso la aplicación de una página (JavaScript) más básica de JavaScript necesita acceder de forma segura a los recursos de una aplicación de origen, y si eres un desarrollador de Java como yo, es probable que sea una aplicación Spring Boot, y probablemente quieras usar OAuth 2.0 flujo implícito. Con este flujo, su cliente enviará un token de portador con cada solicitud y su aplicación del lado del servidor verificará el token con un Proveedor de identidad (IdP). Para una explicación más detallada de los diversos flujos de OAuth, eche un vistazo a nuestra publicación [What the Heck is OAuth](#) .

En este tutorial, aprenderá más sobre el flujo implícito creando dos pequeñas aplicaciones que demuestren estos principios en acción: una aplicación de cliente SPA simple con un poco de JQuery y un servicio de fondo con Spring Boot. Comenzarás utilizando los bits estándar de Spring OAuth y luego cambiarás al arrancador de arranque Okta Spring y verás sus características adicionales. Las primeras secciones serán independientes del proveedor, pero como no soy totalmente imparcial, te mostraré cómo usar Okta como tu IdP.

## Crear una aplicación Spring

# Boot

Si no ha probado start.spring.io vaya a verlo ahora mismo ... con un par de clics, obtendrá una aplicación Spring Boot básica y ejecutable.

```
1 curl https://start.spring.io/starter.tgz \  
2 -d artifactId = oauth-implicit-example \  
3 -d dependencias = seguridad, web \  
4 -d language = java \  
5 -d type = maven-project \  
6 -d baseDir = oauth-implicit-example \  
7 | tar -xzf -
```

Si desea descargar el proyecto desde su navegador, vaya a: start.spring.io busque y seleccione las dependencias de "seguridad" y luego haga clic en el botón verde grande "Generar proyecto".

Una vez que haya descomprimido su proyecto, debería poder iniciarlo en la línea de comando: con `./mvnw spring-boot:run`. Esta aplicación no hará nada todavía, pero este es un buen cheque 'hasta ahora tan bueno'. ¡Mata el proceso `^c` y empecemos a escribir código!

## ¡Escribe algún código!

Bueno, casi. Primero, agregue la dependencia Spring OAuth 2.0 a su `pom.xml`

```
1 < dependencia >  
    < groupId > org . springframework . seguridad  
2 < artifactId > spring - security - oauth2 </ a  
3 < versión > 2.2 . 0. LIBERACIÓN </ version >  
4  
5 </ dependency >
```

Abre `DemoApplication.java`, si sigues (¿y tienes razón?) Entonces debería ubicarse en `src/main/java/com/example/oauthimplicitexample`. No debería ser difícil de encontrar, el proyecto solo contiene dos clases de Java y una de ellas es una prueba.

Anote la clase con `@EnableResourceServer`, esto le indicará a Spring Security que agregue los filtros y la lógica necesarios para manejar las solicitudes implícitas

de OAuth.

Luego, agrega un Controlador:

```
1 @RestController
2     clase pública MessageOfDayController {
3         @GetMapping ( "/" mod" )
4         public String getMessageOfDay ( Principal
5             return "El mensaje del día es aburrido pa
6     }
7 }
```

¡Eso es! Básicamente, hola mundo con una anotación adicional. Comience la copia de seguridad de su aplicación con `./mvnw spring-boot:run`. Deberías poder golpear `http://localhost:8080/mod`:

```
1 curl -v http: // localhost: 8080 / mod
2
3 HTTP / 1.1 401
4 Tipo de contenido: application / json ; charset =
5 WWW-Authenticate: Bearer realm = "oauth2-resource
6
7 {
8     "error" : "no autorizado" ,
9     "error_description" : "Se requiere autenticaci
10 }
```

A 401 ? Sí, seguro por defecto! Además, no hemos suministrado ninguno de los detalles de configuración para nuestro OAuth IdP. Detenga el servidor `^c` y pase a la siguiente sección.

## Obtenga su OAuth Info Ready

Como mencioné anteriormente, usarás Okta en el futuro. Puede registrarse para obtener una cuenta gratuita (siempre) en <https://developer.okta.com/>. Simplemente haga clic en el botón de registro y complete el formulario. Cuando haya terminado, tendrá dos cosas, su URL base Okta, que se parece a algo así como: `dev-`

123456.oktapreview.com un correo electrónico con instrucciones sobre cómo activar su cuenta.

Active su cuenta, y mientras se encuentre todavía en la Consola de Desarrollador Okta tiene un último paso: crear una aplicación Okta SPA. En la barra de menú superior, haga clic en **Aplicaciones** y luego en **Agregar aplicación** . Seleccione **SPA** y haga clic en **Siguiente** .

Complete el formulario con los siguientes valores:

- Nombre: Tutorial implícito de OAuth
- URI base: http: // localhost: 8080 /
- URI de redireccionamiento de inicio de sesión: http: // localhost: 8080 /

Deje todo lo demás como predeterminado y haga clic en **Listo** . En la parte inferior de la página siguiente está tu `Client ID` que necesitarás en el siguiente paso.

## Configurar OAuth para Spring

La aplicación de muestra generada usa un `application.properties` archivo. Prefiero YAML así que voy a cambiarle el nombre al archivo `application.yml` .

Un servidor de recursos de aplicaciones solo necesita saber cómo validar un token de acceso. Dado que el formato del token de acceso no está definido por las especificaciones OAuth 2.0 u OIDC, los tokens se validan de forma remota.

```
1 seguridad :
2   oauth2 :
3     recurso :
4       userInfoUri : https : //dev-123456.oktapreview
```

¡En este punto, puede iniciar su aplicación y comenzar a validar los tokens de acceso! Pero, por supuesto, necesitaría un token de acceso para validar ...

## Crear una página de inicio de sesión

Para mantener las cosas simples, vas a reutilizar tu aplicación Spring Boot existente para alojar tu SPA. Por

aplicación Spring Boot existente para alojar tu SPA. Por

lo general, estos activos pueden alojarse en otro lugar: una aplicación diferente, un CDN, etc. Simplemente parece exagerado alojar un archivo index.html solitario en una aplicación diferente para los propósitos de este tutorial.

### Crea un nuevo archivo

src/main/resources/static/index.html y complétalo con lo siguiente:

```

1  <!doctype html>
2
3  <html lang = "en" >
4  <head >
5      <meta charset = "utf-8" >
6      <meta name = "viewport" content = "width =
7      <meta name = "description" content = "" >
8      <meta name = "author" content = "" >
9      <title > Okta Implicit Spring-Boot </title >
10     <base href = "/" >
11     <script src = "https://ok1static.oktacdn.com
12     <link href = "https://ok1static.oktacdn.com/
13     <link href = "https://ok1static.oktacdn.com/
14     <link rel = "stylesheet" href = "https://ma
15     <script src = "https://ajax.googleapis.com/a
16     <script src = "https://maxcdn.bootstrapcdn.c
17 </head >
18 <body >
19     <!-- Renderiza el widget de inicio de sesión aquí -->
20     <div id = "okta-login-container" > </div >
21     <!-- Representa la respuesta REST aquí -->
22     <div id = "cool-stuff-here" > </div >
23     <!-- Y un botón de cierre de sesión, oculto por d -->
24     <button id = "logout" type = "button" class =
25     <script >
26
27     $.ajax ({

```

```

28     url : "/ sign-in-widget-config" ,
29 }). luego ( función ( datos ) {
30
31     // estamos preparando nuestro objeto config co
32     // Podrías definir estáticamente tu configurac
33     / *
34     const data = {
35         baseUrl: 'https://dev-123456.oktapreview.com
36         ID de cliente: '00icu81200icu812w0h7',
37         redirectUri: 'http: // localhost: 8080',
38         authParams: {
39             emisor: 'https://dev-123456.oktapreview.co
40             responseType: ['id_token', 'token']
41         }
42     }; * /
43
44     // queremos el token de acceso, así que incluy
45     datos . authParams . responseType = [ 'id_tok
46     datos . authParams . ámbitos = [ 'openid' , '
47     datos . redirectUri = ventana . ubicación .
48     // configurar el widget
49     ventana . oktaSignIn = nuevo OktaSignIn ( d
50
51     // manejar el resto de la página
52     doInit ();
53 });
54
55 / **
56  * Realiza una solicitud a un recurso REST y mue
57  * @param accessToken El token de acceso utiliza
58  * /
59 function doAllTheThings ( accessToken ) {
60
61     // incluye el token de portador en la sollicitu
62     $ . ajax ({
63         url : "/ mod" ,
64         encabezados : {

```

```

        'Autorizacion' : "Portador" + accesoloke
65     <
66     },
67     }). luego ( función ( datos ) {
68         // Renderiza el mensaje del día
69         $ ( '# cool-stuff-here' ). append ( "<strong
70     })
71     . fallar ( función ( datos ) {
72         // manejar cualquier error
73         consola . error ( "ERROR !!" );
74         consola . log ( error de datos . responseJSO
75     consola . log ( data . responseJSON . error_
76     });
77
78     // muestra el botón de cerrar sesión
79     $ ( "#logout" ) [ 0 ]. estilo . display = 'b
80 }
81
82 function doInit () {
83
84     $ ( "#logout" ). click ( función () {
85         oktaSignIn . signOut (() => {
86             oktaSignIn . tokenManager . clear ();
87             ubicación . recargar ();
88         });
89     });
90
91     // Verifica si ya tenemos un token de acceso
92     const token = oktaSignIn . tokenManager . g
93
94     // si lo hacemos bien, ¡simplemente ve con él!
95     if ( token ) {
96         doAllTheThings ( token . accessToken )
97     } else {
98
99         // de lo contrario, mostrar el widget de ini
100
101         oktaSignIn . renderEl (
102             { el : '# okta-login-container' },
103             función ( respuesta ) {
104

```

```

3
10
4      // verificar si el éxito
10      if ( response . status === 'SUCCESS' )
5      ◀────────────────────────────────▶
10
6
10      // para nuestro ejemplo, tenemos el to
7      ◀────────────────────────────────▶
10      oktaSignIn . tokenManager . add ( 'my_
8      ◀────────────────────────────────▶
10      oktaSignIn . tokenManager . add ( 'my_
9      ◀────────────────────────────────▶
11
10
11      // esconder el widget
11
12      oktaSignIn . hide ();
11
13
11      // ¡Ahora viene la parte divertida!
4      ◀────────────────────────────────▶
11      doAllTheThings ( respuesta [ 1 ]. acce
5      ◀────────────────────────────────▶
11
12      }
6
11      },
7
11      función ( err ) {
8
11      // manejar cualquier error
9
12      consola . log ( err );
10
12      }
11
12      );
12
13      }
12
14      }
12
15
12      </ script >
16
12      </ body >
17
12      </ html >
18

```

Esta página hace algunas cosas:

- Muestra el widget Okta de inicio de sesión y obtiene un token de acceso



- Llama al `/sign-in-widget-config` controlador para configurar dicho widget (estamos pretendiendo que este archivo es servido por otro servicio)
- Una vez que el usuario inicia sesión, la página llama al `/mod` controlador (con el token de acceso) y muestra el resultado

Para admitir nuestro HTML, necesitamos crear una nueva Controller para el `/sign-in-widget-config` punto final.

En el mismo paquete que la clase de la aplicación Spring Boot, crea una nueva `SignInWidgetConfigController` clase:

```

1  @RestController
   class pública SignInWidgetConfigController {
2
3
4      private final String issuerUrl ;
5      private final String clientId ;
6
7      público SignInWidgetConfigController ( @Value
8
9
10         Afirmar . notNull ( clientId , "Propieda
11
12         Afirmar . notNull ( issuerUrl , "Propiedad
13
14         esta . clientId = clientId ;
15         esta . issuerUrl = issuerUrl ;
16     }
17
18     @GetMapping ( "/ sign-in-widget-config" )
19     public WidgetConfig getWidgetConfig () {
20
21         devolver el nuevo WidgetConfig ( issuerU
22
23     }
24
25     clase pública estática WidgetConfig {
26         public String baseUrl ;
27         public String clientId ;
28         Mapa público < String , Object > authPar
29
30         WidgetConfig ( Cadena emisor , Cadena cl

```

```

27         esta . clientId = clientId ;
28         esta . authParams . put ( "emisor" , e
29         esta . baseUrl = emisor . replaceAll
30     }
31 }
32 }

```

Agregue la configuración correspondiente a su `application.yml` archivo:

```

1  okta :
2    oauth2 :
3      # ID de cliente del paso anterior
4      ID de cliente : 00 ICU81200ICU812
5      emisor : https : //dev-123456.oktapreview.com/

```

Lo último es permitir el acceso público a la `index.html` página y `/sign-in-widget-config`

Defina un `ResourceServerConfigurerAdapter` Bean en su Aplicación para permitir el acceso a esos recursos.

```

1  @Frijol
2  protected ResourceServerConfigurerAdapter resou
3  return new ResourceServerConfigurerAdapter (
4      @Anular
5      public void configure ( HttpSecurity ht
6          http . authorizeRequests ()
7              . antMatchers ( "/" , "/index.
8              . anyRequest (). autenticado (
9      }
10 };
11 }

```

## ¡Préndelo!

Inicie su aplicación nuevamente con `./mvnw spring-boot:run`, y navegue hasta `http://localhost:8080/`.

Debería poder iniciar sesión con su nueva cuenta Okta y

## Pruebe el arrancador de arranque Okta Spring

Hasta ahora (con la excepción de la página de inicio de sesión) ha estado utilizando el soporte de Spring Security OAuth 2.0. Esto solo funciona porque: ¡estándares! Hay algunos problemas con este enfoque:

- Cada solicitud a nuestra aplicación requiere un viaje de ida y vuelta innecesario al OAuth IdP
- No sabemos qué ámbitos se usaron cuando se creó el token de acceso
- Los grupos / roles del usuario no están disponibles en este contexto

Estos pueden o no ser problemas para su aplicación, pero resolverlos es tan simple como agregar otra dependencia a su archivo POM:

```
1 < dependencia >
    < ID de grupo > com.okta.spring </ groupId >
2 < artifactId > okta-spring-boot-starter </ art
3 < versión > 0.2.0 </ version >
4 </ dependencia >
```

Incluso puede recortar su `application.yml` archivo si lo desea, cualquiera de las `security.*` propiedades tendrá prioridad sobre `okta.*` las siguientes:

```
1 okta :
2 oauth2 :
3     ID de cliente : 00ICU81200ICU812
4     emisor : https : //dev-123456.oktapreview.com/
```

¡Reinicie su aplicación y las dos primeras preocupaciones se han solucionado!

El último requiere un paso adicional, tendrá que agregar datos adicionales al token de acceso de Okta. Tenemos una publicación completa sobre este tema , pero las notas del acantilado son las siguientes:

Regrese a Okta Developer Console, en la barra de menú, haga clic en **API > Authorization Server**. En este ejemplo, hemos estado utilizando el servidor de autorización 'predeterminado', así que haga clic en editar, luego seleccione la pestaña 'Reclamos'. Haga clic en 'Agregar reclamo' y complete el formulario con los siguientes valores:

- Nombre: grupos
- Incluir en tipo de token: token de acceso
- Tipo de valor: Grupos
- Filtro: Regex - .\*

Deje el resto como predeterminado y haga clic en 'Crear'.

El `okta-spring-boot-starter` mapa automáticamente los valores en el `groups` reclamo a las autoridades de seguridad de primavera; en la moda estándar de Spring Security, podemos anotar nuestros métodos para configurar los niveles de acceso.

Para habilitar el uso de la `@PreAuthorize` anotación, debe agregarla `@EnableGlobalMethodSecurity` a su aplicación Spring Boot. Si también desea validar los ámbitos de OAuth, deberá agregar un `OAuth2MethodSecurityExpressionHandler`. Simplemente suelte el siguiente fragmento en su aplicación Spring Boot.

```

1  @EnableGlobalMethodSecurity ( prePostEnabled =
2  clase estática protegida GlobalSecurityConfigur
3  @Anular
4  protected MethodSecurityExpressionHandler cr
5  devuelve el nuevo OAuth2MethodSecurityEx
6  }
7  }

```

Finalmente, actualice `MessageOfDayController` con a `@PreAuthorize` (en este caso, está permitiendo que los miembros del grupo 'Todos' o cualquiera con el alcance de 'correo electrónico').

```

1  @RestController
2  ~  clase pública MessageOfDayController {

```

```

2     public MessageOfTheDayController (
3         @GetMapping ( "/" mod" )
4         @PreAuthorize ( "hasAuthority ('Everyone') ||
5         public String getMessageOfTheDay ( Principal
6         return "El mensaje del día es aburrido pa
7     }
8 }

```

## ¡Aprende más!

En esta publicación, creamos una aplicación estándar Spring Boot + Spring Security OAuth 2.0 que utiliza un flujo implícito de OAuth, y luego la enriqueció con el `okta-spring-boot-starter` soporte agregado (sin ningún código) para: validación de token de acceso del lado del cliente, compatibilidad con OAuth scope y Mapeo del grupo Okta a la autoridad. La próxima vez, crearé una aplicación de muestra que utiliza un flujo de código OAuth.

¿Quieres saber más sobre OAuth?

- Qué diablos es OAuth
- Un manual de OpenID Connect
- OAuth.net

¿Preguntas? ¿Comentarios? Buenas historias sobre OAuth? Sígueme en Twitter [@briandemers](#) y asegúrate de seguir a mi equipo [@oktadev](#) .


Secure Your SPA with OAuth and Spring Boot fue publicado originalmente en el blog de desarrolladores de Okta el 27 de octubre de 2017.

---

Cree y ejecute más rápido con la API de administración de usuarios de Okta. ¡Regístrese hoy para la edición gratuita de desarrolladores para siempre !

---

Temas: JAVA, SPA, ARRANQUE DE PRIMAVERA, OAUTH, TUTORIAL

Publicado en DZone con el permiso de Brian Demers ,  
DZone MVB . [Vea el artículo original aquí.](#) 

Las opiniones expresadas son las propias y no representan a DZone.

Las opiniones expresadas por los contribuidores de DZone son suyas.

# Obtenga lo mejor de Java en su bandeja de entrada.

Manténgase actualizado con DZone's Bi-weekly Java Newsletter. [VER UN EJEMPLO](#)

[SUSCRIBIR](#)

## Recursos para socios de Java

Migrar a bases de datos de Microservicio

Programa de desarrollo de Red Hat



Play Framework: La ruta del arquitecto de JVM hacia aplicaciones web súper rápidas

Lightbend



Sube de nivel tu código con un IDE Pro

JetBrains



Cómo resolver particiones de red en segundos con Lightbend

Enterprise Suite

Lightbend



## Node.js Crash Course

by [Jesse Warden](#) MVB · Nov 30, 17 · [Web Dev Zone](#)

Tips, tricks and tools for creating your own data-driven app, brought to you in partnership with Qlik.

### Introduction

I've been doing Node full-time at work and noticed a lot of other people lacking a centralized resource to get up and running quickly. There are a lot of wonderful resources out there for Node that are only a Google

search away, but, hopefully, this document should get you coding quickly and help you to communicate effectively with other Node developers.

I've tried to write this list in order of the most important things you need to know. Feel free to skip around.

## Node Version Manager (NVM)

Node changes often. To quickly change which version you're using, install NVM. I've put `nvm use stable` in my `.bash_profile` so whenever I open a terminal, it uses the latest version. If this is confusing or doesn't work, simply download the latest installable from [nodejs.org](https://nodejs.org).

## Running and Testing Node

Open a command line and type `node .` To get out, on your keyboard press Control + C. While in the Node terminal, you can write JavaScript, and import modules to test them.

To run a JavaScript file, simply `cd` to the directory of the code, and in your command line, type `node yourfile.js`.

## Modules

To share code, you use modules. There are 2 types of modules: CommonJS and ES6 (ignore AMD for now). CommonJS is what Node started with, and then browsers adopted ES6. Node 9 will officially adopt ES6 modules, but, for now, ignore that and focus on CommonJS modules since they work with Node, both old and new.

The easiest way is to define functions/variables, then at the very bottom, put them in a list.

## Simple

```
1 // a function and a variable
2 const cow = () => 'cow';
3 const AGE = 38;
4 // every Node.js file gets this module.exports gl
5 // There are a variety of ways to use it, but the
```

```

        // is to define an object, and put your variables
6      <
7      // at the bottom of the file
8      module.exports = {
9          cow,
10         AGE
11     };
12     // use it in index.js
13     const {
14         cow,
15         AGE
16     } = require('./cow');
17     console.log("cow:", cow());

```

## Big One

When you have a higher module that imports a bunch of child ones, you can enforce the user to require subfolders like this:

```

1     const Maybe = require('./some/deep/folder/thing.j

```

This is fine. But, another way to suggest what the user should use is to only expose that in a higher module:

```

1     // your library/index.js
2     const Maybe = require('./library/some/deep/folder
3     const CHICKEN = 'Sooo Good, mannnn....';
4     const {
5         cow,
6         AGE
7     } = require('./cow');
8     // we don't expose AGE, but everything else is ok
9     // we also organize things
10    module.exports = {
11        functional: {
12            Maybe
13        },
14        animals: {
15            CHICKEN,
16            cow
17        }
18    };
19    // use it in index.js
20    const yourLib = require('./library');
21    const result = yourLib.animals.cow();
22    console.log("result:", result); // cow

```



## What Are These Weird, Empty Functions at the Top of My File?

If you see things like:

```
1  (function() {  
2    ...  
3  })();
```

Where the majority of the code is in the `...` part, that's called an Immediately Invoked Function Expression. It's an old pattern used in the client-side/browser and is not needed or used in Node. Simple remove the top/bottom parts and manually expose the functions/variables you wish to use.

## Modify Dependency Injection

If you see this version:

```
1  (function(window, undefined) {
```

That's the browser's way of performing dependency injection, specifically to help remove global variables. While things like `window` and `document` are globals, global variables make things hard to test, so this allows you to pass those values in during testing and runtime. Refactor the functions that use those globals as function parameters.

## Functions vs. Arrow Functions

There are a variety of ways to define functions in JavaScript. The 2 most common ways in Node are old school function declarations and arrow functions.

### Old Functions

The old way of defining functions is:

```
1  function nameOfIt() {  
2    ...  
3  }
```

The powers that these named function declarations have are:

1. You can forward reference them, meaning you can call them from higher up in the code before they're

actually defined in the file if you write imperative code.

2. They have a built-in `arguments` property that is an array of the arguments passed into the function.
3. They have a `this` keyword which allows various forms of Object Orientated Programming.
4. Older browsers provide more informative stack traces because the function name is included in the stack trace vs. an anonymous function (Node doesn't have this problem).

## Arrow Functions

None of those things are needed anymore, especially in Node where stateful OOP doesn't really exist in stateless server applications. That said, many developers still use classes.

Arrow functions have the following differences:

1. No `this`, instead they adopt whatever scope they are in. If you never use `this` or `scope`, then you have none of those problems.
2. No `arguments` property. If you wish to use something like that, you can define a function by using rest parameters, like `addNumbers(...numbers)`. This makes the `numbers` property an Array of arguments; otherwise, you'll have an empty Array.
3. They are treated like anonymous JavaScript functions, which means they are normal variables and you cannot forward reference them. That problem only occurs if you write imperative code. Calling an one arrow function from another arrow function works fine.
4. They automatically return values unless you add `{}` to the function block. This removal of the need to manually write `return` combined with the removal of the need to write the word `function` leads to much smaller functions.

You should use Arrow Functions unless you know why you should be using older functions.

## Arrow Functions With One Parameter

Typically, you write an Arrow function with a parameter like this:

```
1  const sayName = (name) => console.log("Hello " +
```

However, if you just have one argument, the () are optional:

```
1  const sayName = name => console.log("Hello " + na
```

## Arrow Function Line Length

While smaller, two new problems are created using lots of arrow functions. The first is, the line length can still get pretty long as you try to put everything on one line. Some ESLint rules written by jerks yell about this. The second is you'll start having functions return functions, especially with Promises, and it gets unwieldy to read.

You cannot break them into multiple lines after the equal:

```
1  // wrong
2  const sayName =
3      name => console.log("Hello " + name);
```

But you can line break after the fat arrow:

```
1  // correct
2  const sayName = name =>
3      console.log("Hello " + name);
```

This helps with nested functions since we don't have pipe operators like Elm or Elixir.

## Common Pitfall

Debugging one-line arrow functions that are composed together can be a bit challenging. You have three options here.

```
1  const add = (a, b) => a + b;
```

The first is to break it out into a multi-line, imperative style function:

```
1  const add = (a, b) => {
2      console.log("a:", a);
3      console.log("b:", b);
4      const result = a + b;
5      console.log("result:", result);
```

```
6     return result;  
7 };
```

The challenge is to remember to use the `return` keyword to return the result once you go back to multiple line arrow functions.

The second option is to use an `||` (or) statement to log your information first. Since `console` is a noop (a function that returns no value), it'll return undefined, and trigger the code to the right of the `||` operator.

```
1 const add = (a, b) => console.log("a:", a) || a +
```

The third option is to use a modern IDE like Visual Studio Code that supports adding breakpoints on columns.

## Truthy/Falsey

### if (thing)

JavaScript has lax operators for Boolean evaluation. In short, they suck, hence we call them “truthy” and “falsey.” They aren’t very exact.

You have a few options:

1. Learn them and look smart, yet have to continually remind your coworkers.
2. Ignore them and don’t go down that path and use Lodash.

For example, this prints out “it’s true, homey”:

```
1 const cow = true;  
2 if (cow) {  
3     console.log("it's true, homey");  
4 }
```

So does this:

```
1 const cow = 'false';  
2 if (cow) {  
3     console.log("it's true, homey");  
4 }
```

The same holds true for nothing using equality vs strict equality in JavaScript:

```
1 null == undefined; // true  
  null === undefined; // false
```

```
2 null === undefined, // false
```

Ignore it and all the weird edge cases. Create predicate functions using Lodash, and your problems go away, and your code works in all browsers and in Node versions.

## Notes

You'll occasionally see people do `if(!thing) {`. It basically means `!==true`.

## Equality

Comparison operators in JavaScript are broken. You can learn the differences if you care to, but they are too hard to remember and don't really help you write better code. Don't use two equals, use three:

```
1 // wrong
2 if(thing == false)
3 // correct
4 if(thing === false)
```

## Asynchronous Programming

JavaScript, unlike other programming languages, is asynchronous by default. Learn more to understand how to avoid creating race conditions as well as helpful tips if you come from C#. If you're from Scala, JavaScript's Promises are like Scala's Futures.

In short, JavaScript does not stop or "block" on a line of code while an asynchronous operation such as an HTTP request, file read, or database call is run. Instead, you can give it a function to call later when it's done, and your code keeps running in the current call stack. I've written an article that hopefully gives you clear examples of asynchronous programming.

## Callbacks vs. Promises

The old way to code in Node is using callbacks. The new way is Promises. Since it is an opinion, Node continues to support callback APIs and create new APIs using callbacks. While callbacks can result in callback hell, so can Promises.

Either way, callbacks sadly are noops, meaning they don't return a value. We don't do that in functional

can't return a value. We don't do that in functional

programming, and neither should you. While newer versions of node support promisify, you should be using Promises because:

1. They always return a value.
2. They have built-in `try/catch`
3. They are a native, finite state machine.
4. They use Left/Right functional programming error handling fall through.

This leads to easier unit testing, more composable functions, and easier to debug code.

Best article to learn Promises is to learn how Promises are used wrong.

That said, if callbacks are easier for you, and you're stuck with Promise based code, Node 9 has a way to convert them back to callbacks.

## Command Line

To build command line Node apps, check out Commander.

## Object-Oriented Programming

The basics of classes with inheritance work in the latest browser and Node without the need of a transpiler/compiler using ES6. However, transpilers offer a lot of nice features that, if you're from an OOP background, it's worth your time to check out.

For the basics, check out the Babel compiler. For a language, compiler, and simpler parallelism functionality, with runtime exceptions for non-prod code, check out Google's Dart. For another great typed language with a helpful compiler, check out Microsoft's TypeScript. Facebook has Flow in much the same vein.

Be aware, a lot of the marketing of the above tools target browser developers, but many work fine for Node. The beauty of Node is you "can just write code and run it" without waiting for a recompile, but for many, this isn't a problem.

# Functional Programming

You have two options: use libraries or a transpiler.

For libraries, Folktale v2 follows the Fantasy Land spec. Lodash has both functional methods as well as array comprehensions, and low-level JavaScript predicates.

If the mutable state and impurity of JavaScript is too much, you can use Facebook's OCAML influenced Reason, or Haskell influenced PureScript.

## Unit and Integration Testing


To unit test, the four main test runners are Tape, Jasmine, Mocha, and Jest. I like Mocha. Mocha has an assertion library, Chai. For code coverage, use Istanbul. To prevent unit tests from accidentally becoming integration tests that make HTTP calls and other HTTP exceptions, use Nock. For mocking and spies (you poor thing) use Sinon. For integration testing, check out Supertest.

---

Explore data-driven apps with less coding and query writing, brought to you in partnership with Qlik.

---

Topics: FUNCTIONS, NODE.JS, JAVASCRIPT, WEB DEV

Published at DZone with permission of Jesse Warden, DZone MVB. [See the original article here.](#)   
Opinions expressed by DZone contributors are their own.

