

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Tabla de contenido** |  |  |
| [Introducción](#page4) | | 1.1 | |
|  |  |  |  |
| [Preparar](#page7) | | 1.2 | |
|  |  |  |  |
| [JavaScript funcional](#page8) | | 1.3 | |
|  |  |  |  |
| [construcciones ES6](#page11) | | 1.4 | |
|  |  |  |  |
|  | [Parámetros por defecto](#page12) | 1.4.1 | |
|  |  |  | |
|  | [Los literales plantilla](#page14) | 1.4.2 | |
|  |  |  | |
|  | [desestructuración](#page16) | 1.4.3 | |
|  |  |  | |
|  | [Funciones de flecha](#page19) | 1.4.4 | |
|  |  |  | |
|  | [promesas](#page21) | 1.4.5 | |
|  |  |  | |
|  | [Dejar y const](#page24) | 1.4.6 | |
|  |  |  | |
|  | [módulos](#page26) | 1.4.7 | |
|  |  |  | |
| [Pensando en Componentes](#page27) | | 1.5 | |
|  |  |  |  |
|  | [Diseño atómica](#page28) | 1.5.1 | |
|  |  |  | |
|  | [Principios de componentes atómicos](#page29) | 1.5.2 | |
|  |  |  | |
|  | [Beneficios de este enfoque](#page33) | 1.5.3 | |
|  |  |  | |
|  | [El proceso](#page34) | 1.5.4 | |
|  |  |  | |
|  | [Tarea 1](#page36) | 1.5.5 | |
|  |  |  | |
| [reaccionar componentes](#page38) | | 1.6 | |
|  |  |  |  |
|  | [componentes sin Estado](#page39) | 1.6.1 | |
|  |  |  | |
|  | [Los componentes con estado](#page40) | 1.6.2 | |
|  |  |  | |
|  | [Stateful vs Componentes sin estado](#page42) | 1.6.3 | |
|  |  |  | |
|  | [Composición](#page43) | 1.6.4 | |
|  |  |  | |
|  | [Tarea 2](#page45) | 1.6.5 | |
|  |  |  | |
|  | [Tarea # 3](#page46) | 1.6.6 | |
|  |  |  | |
|  | [Tarea # 4](#page47) | 1.6.7 | |
|  |  |  | |
|  | [Tarea # 5](#page48) | 1.6.8 | |
|  |  |  | |
| [Inmutable](#page49) | | 1.7 | |
|  |  |  |  |
|  | [¿Qué es la inmutabilidad?](#page50) | 1.7.1 | |
|  |  |  | |
|  | [El caso de la inmutabilidad](#page52) | 1.7.2 | |
|  |  |  | |
|  | [Soluciones de JavaScript](#page53) | 1.7.3 | |
|  |  |  |  |



1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | [Object.assign](#page54) | | | 1.7.3.1 | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | [Object.freeze](#page55) | | | 1.7.3.2 | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | [Fundamentos Immutable.js](#page56) | | | | 1.7.4 | | | | | | |
|  |  |  |  |  |  |  | |  |  |  |  |  |
|  |  |  | [Immutable.Map](#page57) | | | 1.7.4.1 | | | | | | |
|  |  |  |  |  |  |  |  |  |  | |  |  |
|  |  |  |  |  | [Map.merge](#page58) | 1.7.4.1.1 | | | | | | |
|  |  |  |  | |  |  | |  |  | |  |  |
|  |  |  | [Los objetos anidados](#page59) | | | 1.7.4.2 | | | | | | |
|  |  |  |  |  |  |  |  |  |  | |  |  |
|  |  |  |  |  | [Claves Eliminación](#page62) | 1.7.4.2.1 | | | | | | |
|  |  |  |  |  |  |  | |  |  | |  |  |
|  |  |  |  |  | [Los mapas son Iterable](#page63) | 1.7.4.2.2 | | | | | | |
|  |  |  |  | |  |  | |  |  | |  |  |
|  |  |  | [Immutable.List](#page64) | | | 1.7.4.3 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Actuación](#page66) | | | 1.7.4.4 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Estructuras de datos persistentes y transitorias](#page67) | | | 1.7.4.5 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Documentación oficial](#page68) | | | 1.7.4.6 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  | [Ceremonias](#page69) | | | | 1.7.5 | | | | | | |
|  |  |  |  | | |  | |  |  |  |  |  |
|  |  |  | [Tarea 1](#page75) | | | 1.7.5.1 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea 2](#page76) | | | 1.7.5.2 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea # 3](#page77) | | | 1.7.5.3 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea # 4](#page78) | | | 1.7.5.4 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea # 5](#page80) | | | 1.7.5.5 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea # 6](#page81) | | | 1.7.5.6 | | | | | | |
|  |  |  |  | | |  |  |  |  | |  |  |
|  |  |  | [Tarea # 7](#page82) | | | 1.7.5.7 | | | | | | |
|  |  |  | |  | |  |  |  |  | |  |  |
| [Redux](#page83) | | | | | | 1.8 | | | | | | |
|  |  |  | | | |  | |  |  | |  |  |
|  |  | [Revisión de Reductores y funciones puras](#page85) | | | | 1.8.1 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [Redux Reductores](#page86) | | | | 1.8.2 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [acciones Redux](#page88) | | | | 1.8.3 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [Configuración de la aplicación para utilizar Redux](#page90) | | | | 1.8.4 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [Usando Redux con Componentes](#page93) | | | | 1.8.5 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [Redux y arquitectura de componentes](#page96) | | | | 1.8.6 | | | | | | |
|  | |  | | | |  | |  |  |  |  | |
| [enrutamiento](#page97) | | | | | | 1.9 | | | | | | |
|  |  |  | | | |  | |  |  | |  |  |
|  |  | [reaccionar Router](#page98) | | | | 1.9.1 | | | | | | |
|  |  |  | | | |  | |  |  |  |  | |
|  |  | [Router Redux](#page101) | | | | 1.9.2 | | | | | | |
|  | |  | | | |  | |  |  |  |  | |
| [formas](#page103) | | | | | | 1.10 | | | | | | |
|  |  |  | | | |  | |  |  | |  | |
|  |  | [Forma Redux](#page104) | | | | 1.10.1 | | | | | | |
|  | |  | | | |  | |  |  | |  | |
| [Pruebas](#page107) | | | | | | 1.11 | | | | | | |
|  |  |  | | | |  | | |  | |  | |
|  |  | [Preparar](#page108) | | | | 1.11.1 | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |



2

|  |  |
| --- | --- |
| [componentes](#page109) | 1.11.2 |
|  |  |
| [reductores](#page111) | 1.11.3 |
|  |  |
| [Comportamiento](#page113) | 1.11.4 |
|  |  |

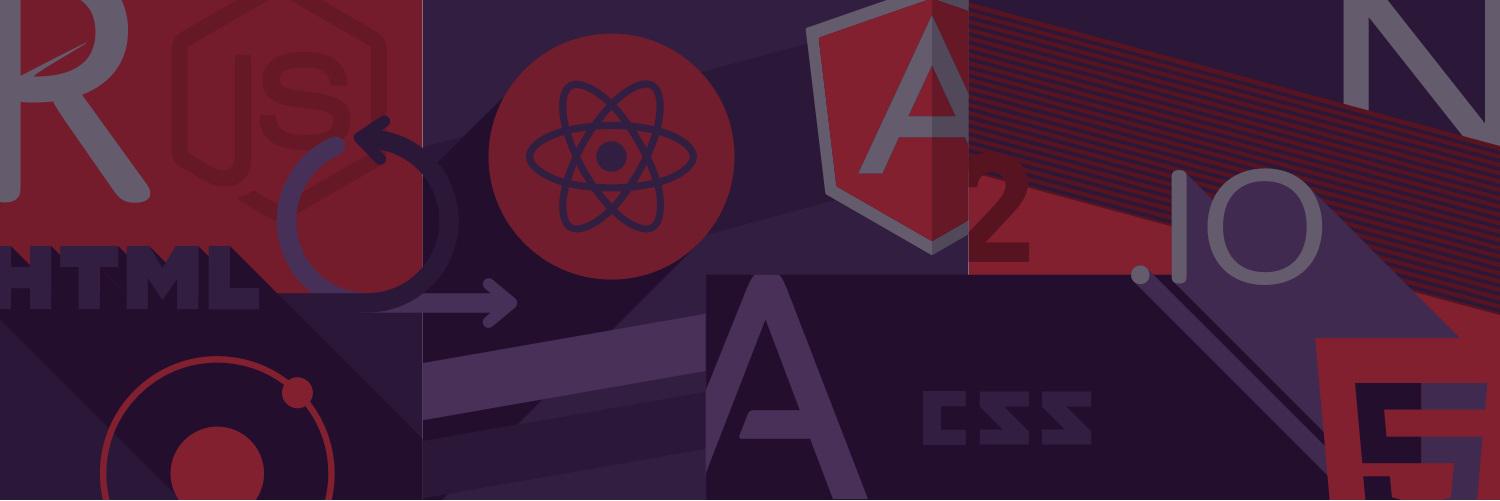


3

Introducción



**Formación React.js**



Como el diseño y desarrollo firma dedicada JavaScript, trabajamos ampliamente con los clientes en React proyectos. Hemos desarrollado esta formación de manera que ahora se puede poner en marcha sus proyectos Reaccionar utilizando las mejores prácticas que hemos desarrollado.

Este libro guía al lector a través de todo lo que necesitan saber de cómo empezar con la cadena de herramientas Reaccionar a escribir aplicaciones con arquitecturas escalables de front-end.

**Lo que aprenderá**

¿Cómo construir una aplicación web con ReactJS, ImmutableJS y Redux.



El proceso utilizado para construir una aplicación usando una arquitectura basada en componentes.



Los beneficios de tener un único estado de la aplicación global y cómo aprovechar el poder de la arquitectura de flujo de inspiración de Redux.



**Requisitos previos**

Los estudiantes deben tener instaladas las siguientes herramientas:

**Línea de comando**

La cadena de herramientas Reaccionar es bastante orientado a la línea de comandos, por lo que sugerimos instalar un buen programa de terminal de línea de comandos.

Para Mac OS X, se recomienda [iTerm2](https://www.iterm2.com/downloads.html).

Para Windows, se recomienda utilizar [Git Bash](http://git-scm.com/download/) (También viene con git - véase más adelante).



4

Introducción



**Git**

Vamos a utilizar Git desde la línea de comandos: <http://git-scm.com/download/>

**Consejos para usuarios de Mac OS X**

Si usted tiene [homebrew](http://brew.sh/) instalado, puede instalar Git por la simple emisión: brebaje instalar git .



**Consejos para los usuarios de Windows**

En caso de duda, seleccione las opciones predeterminadas del asistente de instalación.



El resto de esta formación supone que está utilizando el terminal que viene con Git (Git Bash) para ejecutar todos los ejemplos de línea de comandos.



*Notas: El Git oficial de descarga ofrece varias herramientas gráficas, tanto para OS X y Windows. Si usted es hábil con estas interfaces, no dude en usarlos. Esta formación, sin embargo, sólo proporcionará instrucciones de la línea de comandos.*

**Node.js**

<http://nodejs.org/download/>

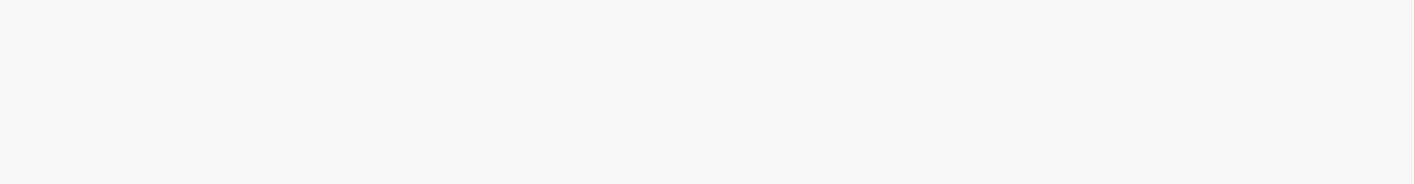
La descarga anterior debe instalar dos comandos: nodo y NPM .



NPM puede ser necesaria una configuración adicional para establecer los permisos correctamente en su sistema.



En Mac OS X, haga lo siguiente:

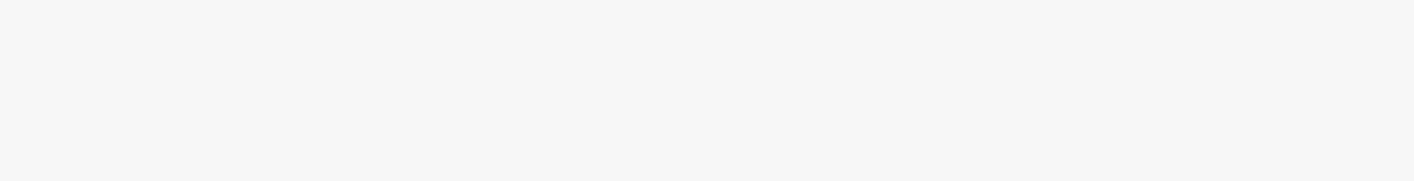


config NPM conjunto prefijo ~ / .npm

eco 'Export PATH = "$ PATH: ~ / .npm / bin"' >> ~ / .bashrc

~ / .bashrc

En Windows, el fuego de Git Bash y escriba el siguiente:

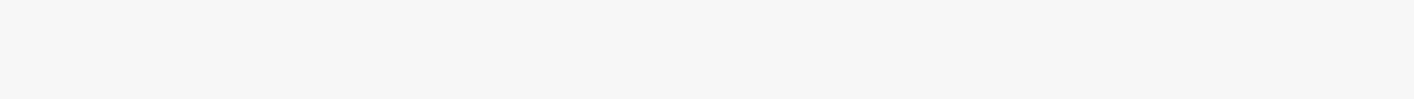


mkdir / c / Usuarios /$ USUARIO/ AppData / Roaming / NPM

eco 'Export PATH = "$ PATH: / c / Archivos de programa / nodejs"' >> ~ / .bashrc

~ / .bashrc

**Nota:** si ha instalado la versión de 32 bits de Windows de nodo, la segunda línea anterior en lugar debe decir:



echo 'export PATH = "$ PATH: / c / Archivos de programa (x86) / nodejs"' >> ~ / .bashrc

**Un editor de código**



5

Introducción



Cualquier editor de texto va a funcionar. En Rangle.io, los más populares editores / IDE son:

[Empuje](http://www.vim.org/download.php)



[Texto sublime](http://www.sublimetext.com/)



[Átomo](https://atom.io/)



[WebStorm](https://www.jetbrains.com/webstorm/)



**Google Chrome**

<https://www.google.com/chrome/browser/desktop/index.html>

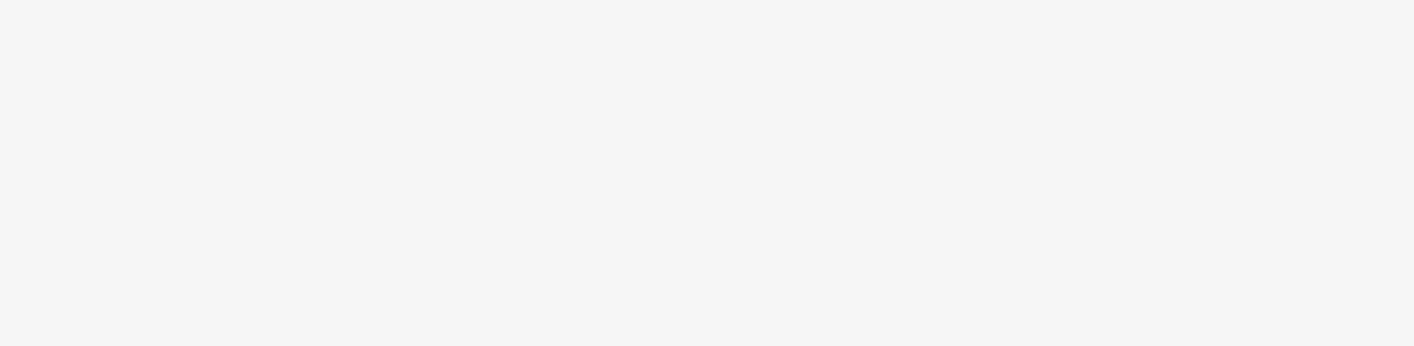


6

Preparar



**Preparar**



git clone https://github.com/rangle/react-training

NPM instalar -g jspm

NPM instalar

jspm instalar

NPM plazo dev

**Solución de problemas**

**se queja de que jspm límite de velocidad de github alcanzó**



1. Ir [github.com](https://github.com), Iniciar sesión y hacer clic ajustes
2. Haga clic en tokens de acceso personal y luego Generar nuevo token (Asegúrese de activar todas las opciones que desee)
3. Copiar el token y empezar línea de comandos dentro de la carpeta del proyecto
4. correr jspm github configuración del registro



7

JavaScript funcional



**JavaScript funcional**



La programación funcional se aproxima a los programas de ordenador como un conjunto de transformaciones de datos, al igual que en las funciones matemáticas. Su programa es una colección de funciones. Cada función tiene algunas entradas y produce algunas salidas. funciones simples se utilizan como bloques de construcción para las más complejas. Este es uno de los primeros enfoques de programación, que data de la década de 1960 y se basa en las matemáticas desarrollada en la década de 1930.

JavaScript permite diferentes estilos de programación: orientado a objetos, imperativo, y funcional.

Reaccionar trae estos conceptos de programación funcional a la interfaz de usuario. Proporciona abstracciones tales como componentes que son (en su mayoría) funciones puras. Un estado determinado (como accesorios) regresan algunos DOM. Esto nos permite alejarnos de tocar imperativamente el DOM.

El estilo funcional es más flexible, y hace más fácil probar y depurar. Dos conceptos básicos de programación funcional son la inmutabilidad y sin estado, ambos de los cuales están cubiertos en las secciones posteriores.



8

JavaScript funcional



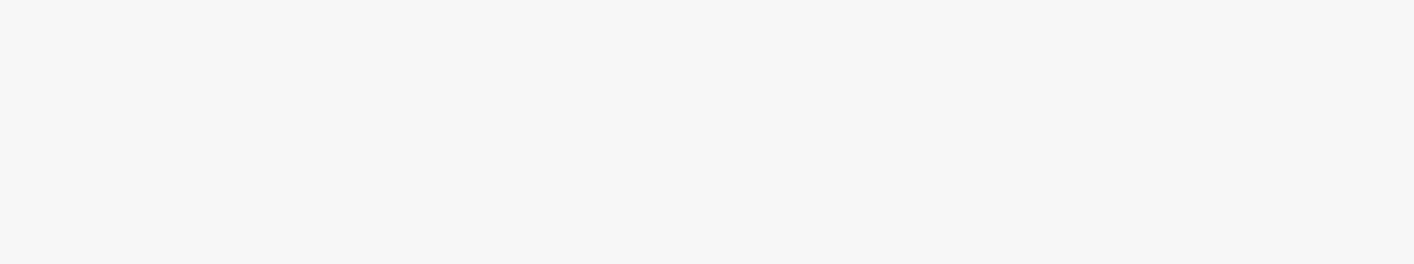
**Conceptos funcionales**

**Funciones puras (Slice vs empalme)**



A pura función es una función donde el valor de retorno solamente se determina por sus valores de entrada, sin efectos secundarios observables.

Por ejemplo, rebanada y empalme completar la misma funcionalidad, sin embargo, empalme tiene el impacto no deseado de la mutación de la entrada de origen.



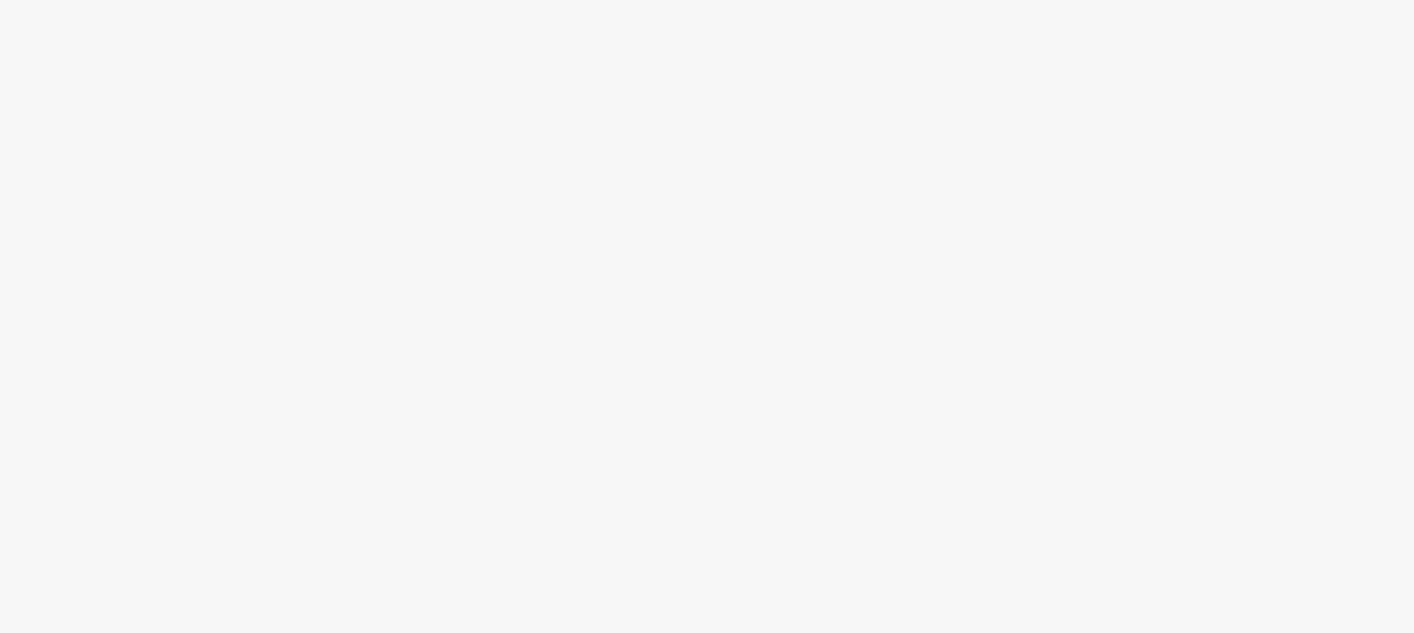
dejar ltrs = ["un", "segundo", "do"]

ltrs.slice (1) // devuelve [ "b", "c"], donde ltrs sigue siendo [ "a", "b", "c"]

ltrs.splice (1) // devuelve [ "b", "c"], donde ltrs es ahora [ "a"]

**Mapa, filtro, Reducir**

métodos Asistente de Arreglo mapa , filtrar y reducir son ejemplos de la programación funcional, que toman en funciones y no mutan la matriz original.



var list = [1, 2, 3, 4, 5];

var doble = list.map (función (X) {

regreso X \* 2;

});

var GT3 = list.filter (función (X) {

regreso x> 3;

});

var = suma list.reduce ((resultado, x) => {

consola.Iniciar sesión(`Resultado de: $ {} consecuencia, x: $ {X}, como resultado cabo: $ {x} + resultado`); regreso resultado + x;

}, 0);

**Zurra**

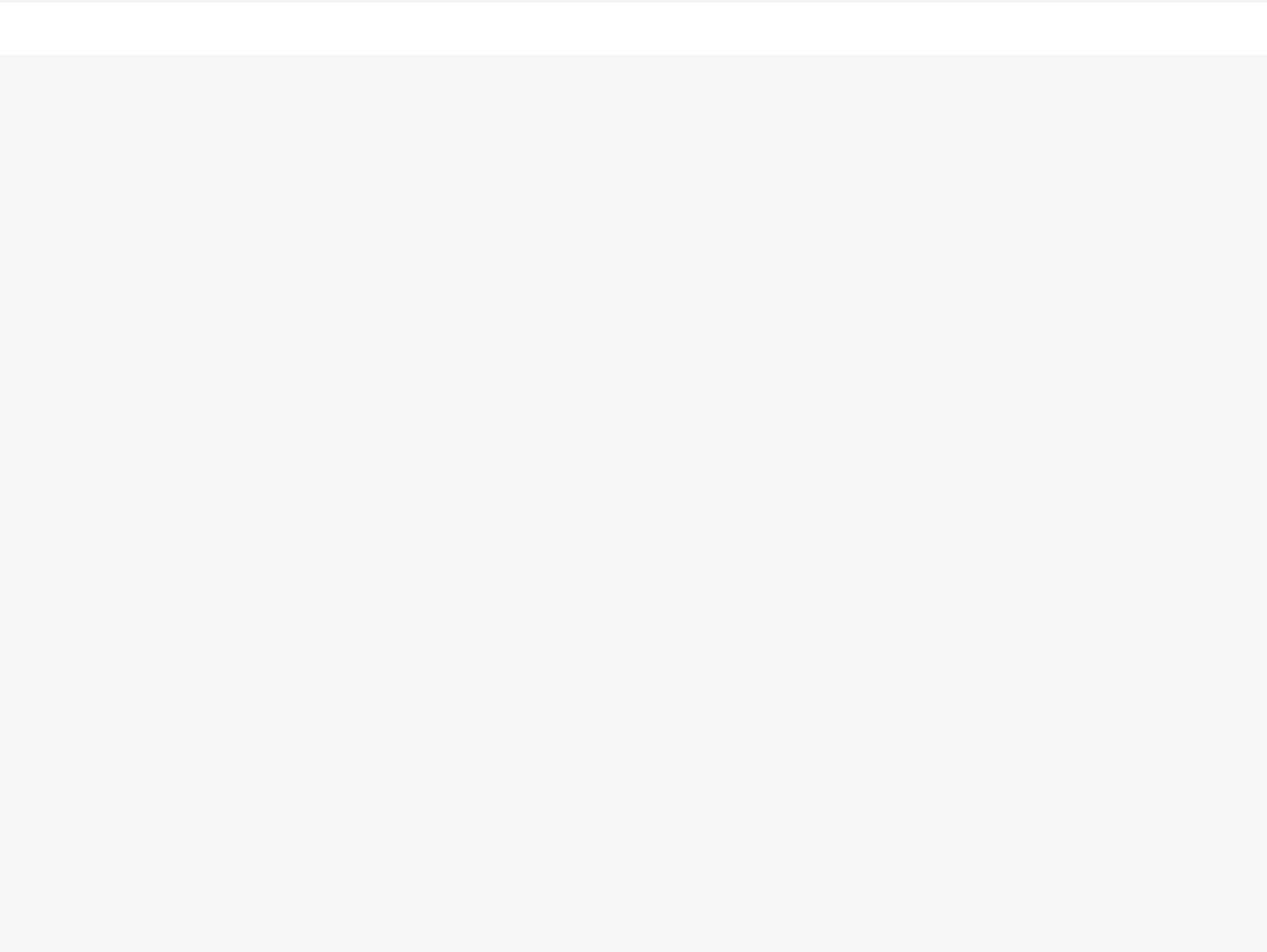


Currying es la técnica de la traducción de la evaluación de una función que toma varios argumentos (o un tupla de argumentos) en la evaluación de una secuencia de funciones, cada uno con un solo argumento.



9

JavaScript funcional



función añadir (a B C) {

regreso a + b + c;

}

añadir(1, 2, 3); // devuelve 6

* Currying add () función añadir (un) {

función de retorno(segundo) { función de retorno(do) {

regreso a + b + c;

}

}

}

add1 = add (1); // devuelve una función donde a = 1

ADD2 = add1 (2); // devuelve una función donde a = 1, b = 2 add3 = ADD2 (3); // devuelve 6

* A continuación se muestra una forma corta de currificación usando funciones de matriz ES6 añadir => a => b => c => a + b + c

resultado = add (1) (2) (3); // devuelve 6

**recursos**

Esta sección cubre los conceptos básicos, echa un vistazo a este [gitbook](https://drboolean.gitbooks.io/mostly-adequate-guide/content/) para un análisis en profundidad de la programación funcional



10

construcciones ES6



**ES6**

JavaScript fue escrito por Brendan Eich durante una sección de diez días en la década de ninties. Más de veinte años después, el lenguaje está prosperando. Hay subconjuntos, superseries, las versiones actuales, y una próxima versión. ES6 es que la próxima versión, y trae un montón de nuevas características. Algunos de los aspectos más destacados:

clases



Parámetros por defecto



Los literales plantilla



desestructuración



Funciones de flecha



promesas



Dejar y const



módulos



11

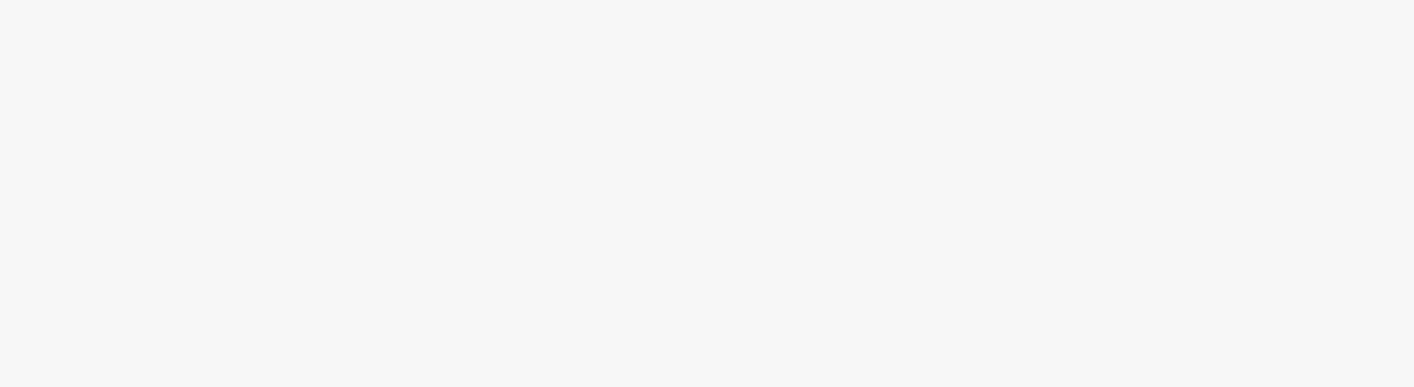
Parámetros por defecto



**Parámetros por defecto**

**ES5 vieja manera de establecer parametros por defecto**

Enfoque 1: forma fácil.



* Esto es lo que la gente normalmente lo hacen en la ES5 para configurar parametros por defecto función enlazar(altura, color, callbackFn) {

var height = altura || 50; var color = color de || 'rojo';

var callbackFn = callbackFn || función() {};

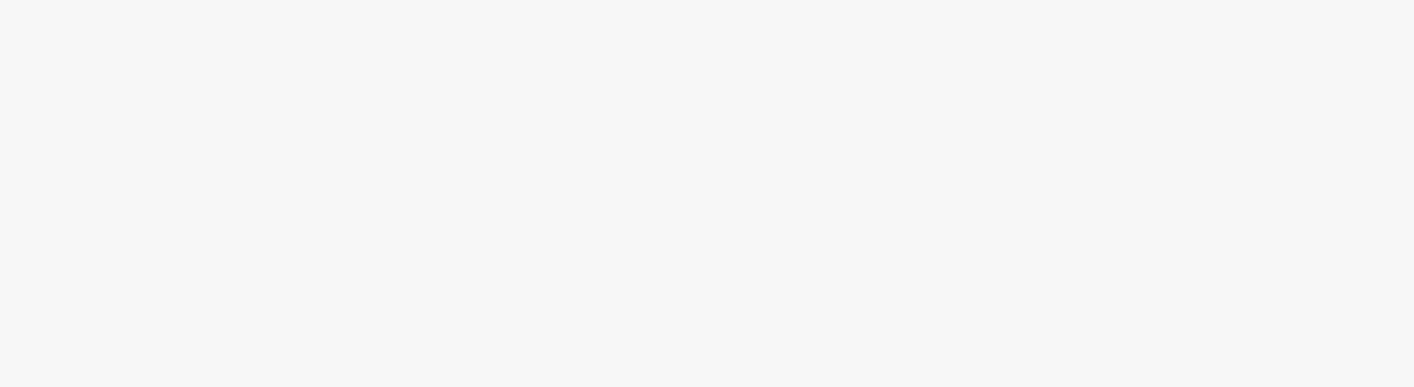
* + función de contenido ...

}

Funciona bien, pero en la aplicación anterior que no tuvo en cuenta los valores Falsy. Por ejemplo:0 '', null, NaN, falsa son [Los valores Falsy](https://dorey.github.io/JavaScript-Equality-Table).



Enfoque 2: mejor manera.



* Así que hay una mejor manera de hacer esto, se comprueba parámetro es en realidad no definido o no: función enlazar(altura, color, callbackFn) {

var height = tipo de altura! == 'Indefinido' ? altura: 50;

var color = tipo de color! == 'Indefinido' ? color : 'rojo';

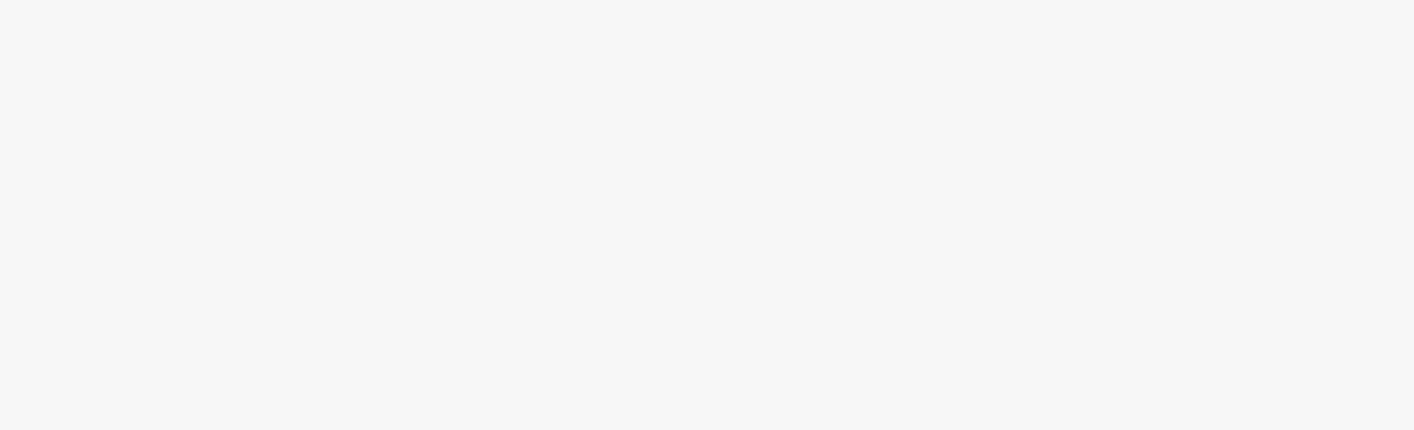
var callbackFn = tipo de callbackFn! == 'Indefinido' ? callbackFn: función() {};

* + función de contenido ...

}

**ES6 manera de escribir parametros por defecto**

Enfoque 3: ES6 manera, se pone tan mucho mejor.



función enlazar(height = 50, color = 'rojo', callbackFn = () => {}) {

// contenido de la función ...

}

* o el uso de ES6 const y dejar const noop = () => {};

const link = (altura = 50, Color = 'rojo', CallbackFn = NOOP) => {

* + función de contenido ...

};



12

Parámetros por defecto



Otras lecturas:

[https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/default\_para](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/default_parameters) [metros](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/default_parameters)



<http://tc39wiki.calculist.org/es6/default-parameter-values/> <http://exploringjs.com/es6/ch_parameter-handling.html> <https://dorey.github.io/JavaScript-Equality-Table>



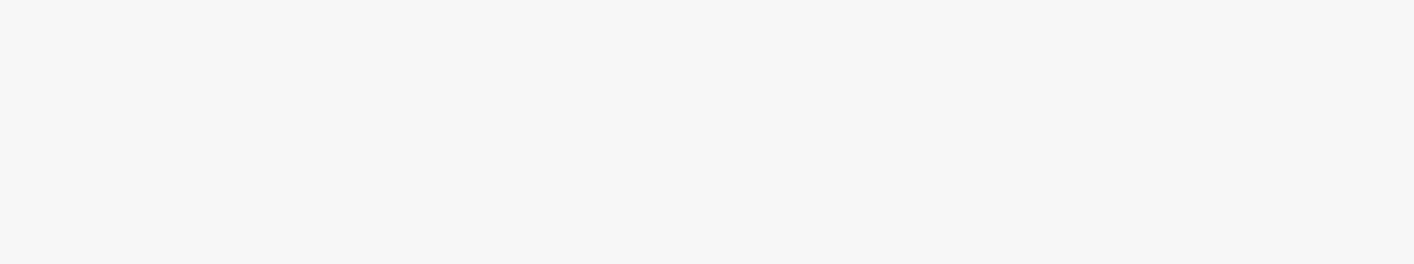
13

Los literales plantilla



**Los literales plantilla**

En JavaScript tradicional, texto que se encierra dentro de coincidencia " marcas o 'marcas se considera una cadena. El texto entre comillas simples o dobles sólo puede estar en una línea. Tampoco hubo manera de insertar datos en estas cadenas. Esto dio lugar a una gran cantidad de código concatenación fea que parecía:

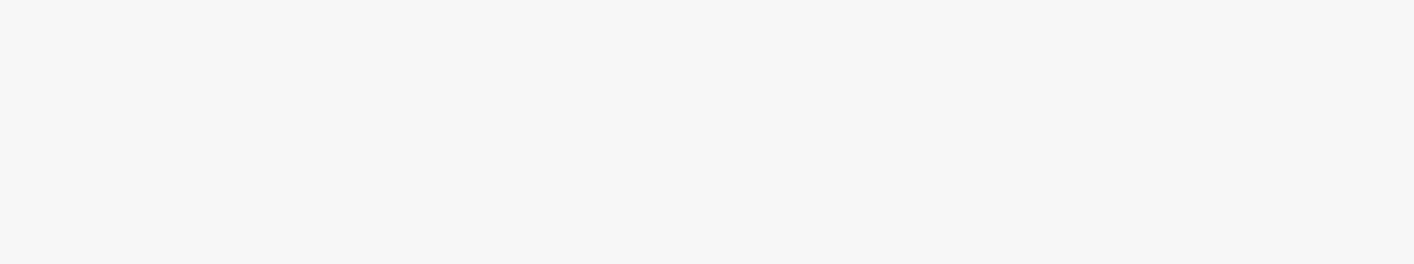


var name = 'Sam';

var edad = 42;

consola.Iniciar sesión('Hola, mi nombre es ' + Nombre + ' Yo soy ' + Edad + 'Años'); // = 'hola mi nombre es Sam Tengo 42 años de edad'

ES6 introduce un nuevo tipo de cadena literal que está marcado con la espalda garrapatas ( `). Estas cadenas literales pueden incluir saltos de línea, y hay un nuevo mecanismo para la inserción de variables en cadenas:

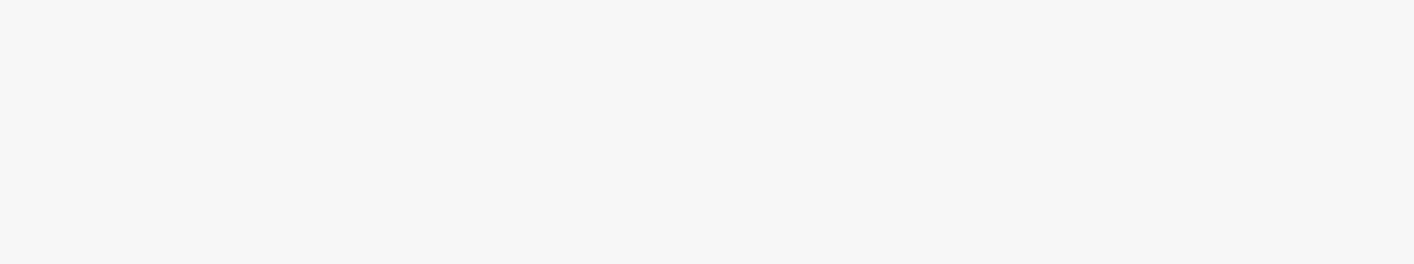


var name = 'Sam';

var edad = 42;

consola.Iniciar sesión(`Hola mi nombre es $ {nombre}, y yo soy $ {} años de edad old`); // = 'hola mi nombre es Sam, y yo soy 42 años'

los ps funciona bien con cualquier tipo de expresión, incluyendo expresiones miembros y las llamadas a métodos.



var name = 'Sam';

var edad = 42;

consola.Iniciar sesión(`Hola mi nombre es $ {name.toUpperCase ()}, y yo soy de edad $ {} / 2 años old`); // = 'hola mi nombre es Sam, y tengo 21 años de edad'

Hay todo tipo de lugares en los que este tipo de cadenas puede ser útil, y desarrollo web front-end es una de ellas.

**Otras lecturas**

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals> <https://developers.google.com/web/updates/2015/01/ES6-Template-Strings?hl=en> <https://ponyfoo.com/articles/es6-template-strings-in-depth>



14

Los literales plantilla



<http://exploringjs.com/es6/ch_template-literals.html>



15

desestructuración

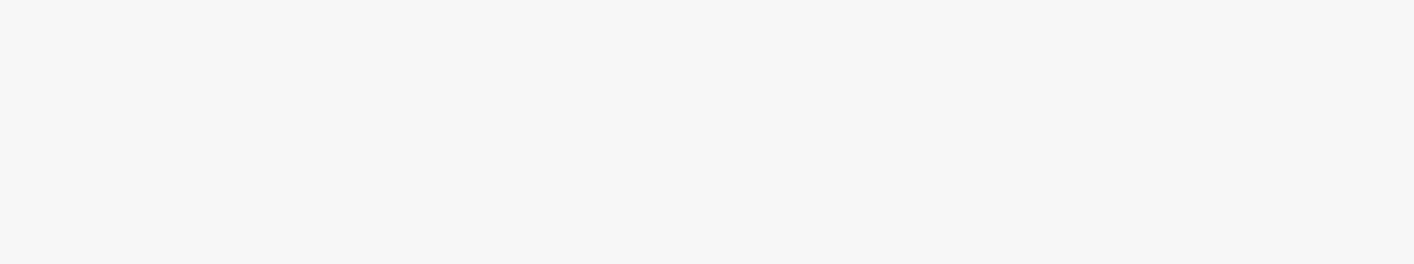


**desestructuración**

Desestructuración es una manera de extraer rápidamente los datos de cada una {} o [] sin tener que escribir mucho código.



Para [prestado de la MDN] [mdnDest], desestructuración se puede utilizar para girar el siguiente:

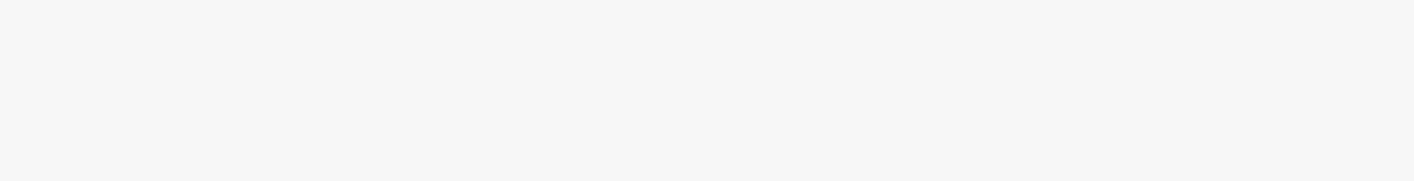


dejar foo = ['uno', 'dos', 'Tres'];

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | dejar | uno | = | foo [0]; |  |
|  |  |  |  |  |  |
| dejar | dos | = | foo [1]; |  |

dejar tres = foo [2];

dentro



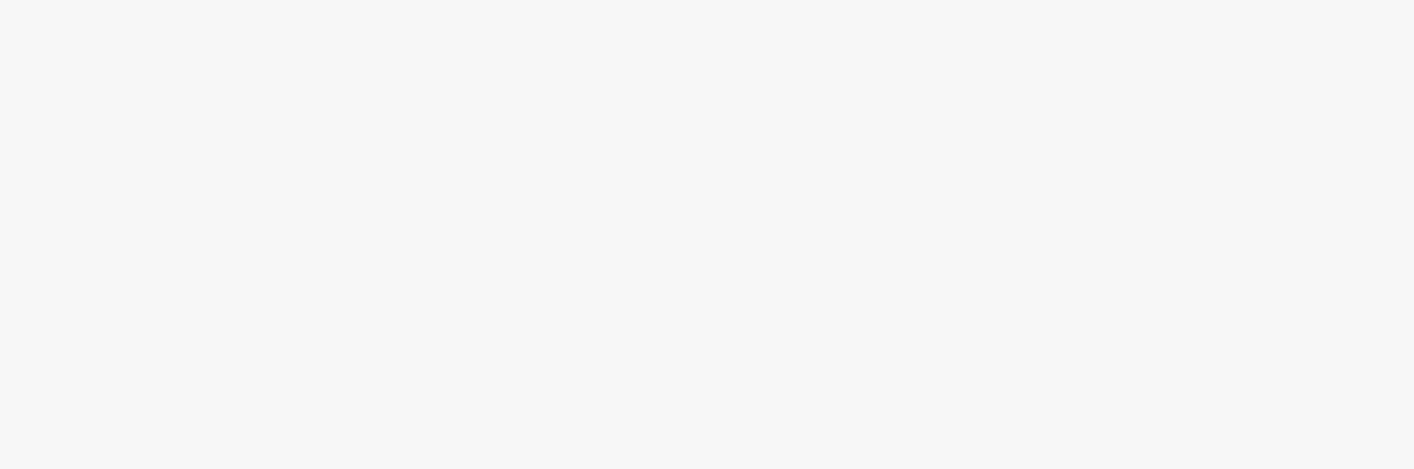
dejar foo = ['uno', 'dos', 'Tres'];

dejar [Uno, dos, tres] = foo;

consola.log (uno); // 'uno'

Esto es muy interesante, pero al principio puede ser difícil de ver el caso de uso. ES6 también es compatible

desestructuración del objeto, lo que podría hacer que utiliza más obvia:



dejar mi\_módulo = {

drawSquare: función drawSquare(longitud) { / \* Aplicación \* / },

drawCircle: función drawCircle(radio) { / \* Aplicación \* / },

drawText: función drawText(texto) { / \* Aplicación \* / },};

dejar {DrawSquare, drawText} = mi\_módulo;

drawSquare (5);

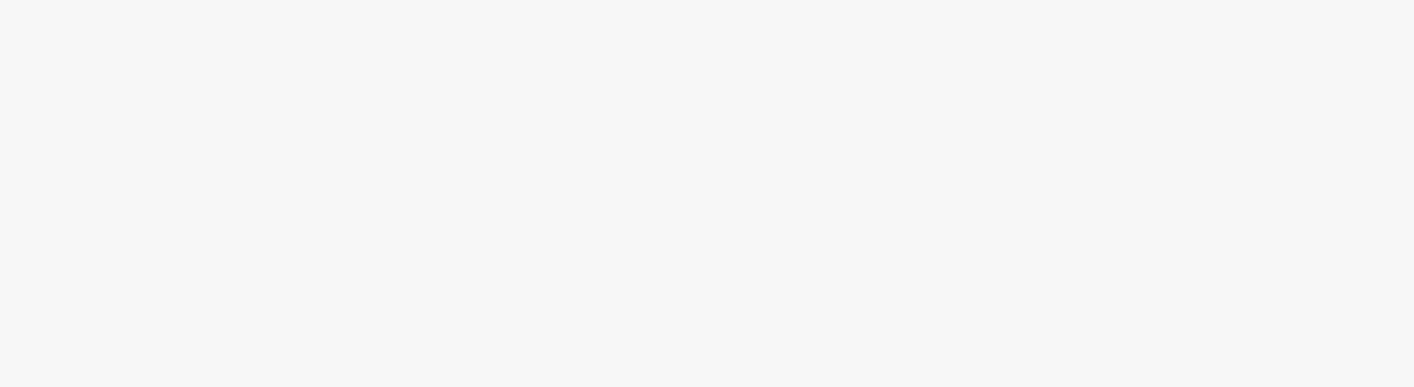
drawText ('Hola');

La desestructuración también se puede utilizar para el paso de objetos en una función, lo que permite tirar de propiedades específicas de un objeto de una manera concisa. También es posible asignar valores por defecto a los argumentos desestructuradas, que puede ser un patrón útil si pasa en un objeto de configuración.



dieciséis

desestructuración



dejar Jane = {firstName: 'Jane', apellido: 'Gama'};

dejar john = {firstName: 'John', apellido: 'Gama', segundo nombre: 'Herrero' } función sayName({Nombre, apellido, MiddleName = 'N / A'}) {

consola.Iniciar sesión(`Hola $ {NOMBRE} $ {} MiddleName $ {}` lastName)

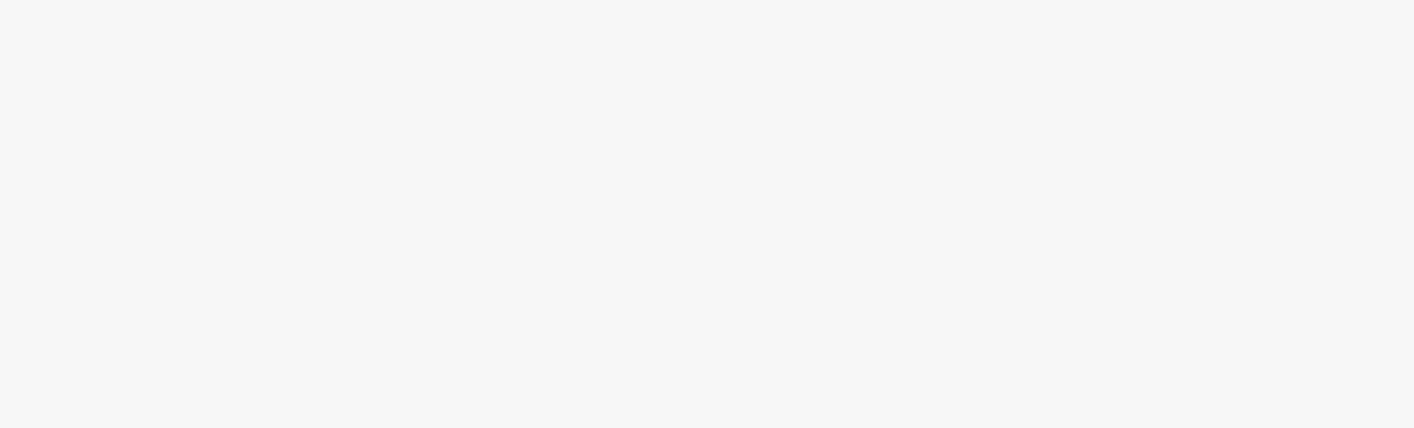
}

sayName (Jane) // -> Hola Jane N / A Doe

sayName (John) // -> Helo John Doe Smith

El operador propagación permite una expresión a ser expandido en lugares donde se espera que múltiples argumentos (para llamadas de función) o elementos múltiples (por literales de matriz) o múltiples variables (por desestructuración asignación).

Para matrices:



const frutas = ['manzana', 'plátano'];

const verduras = ['Pepino', 'patata'];

const comida = ['uvas', ... frutas, verduras ...];

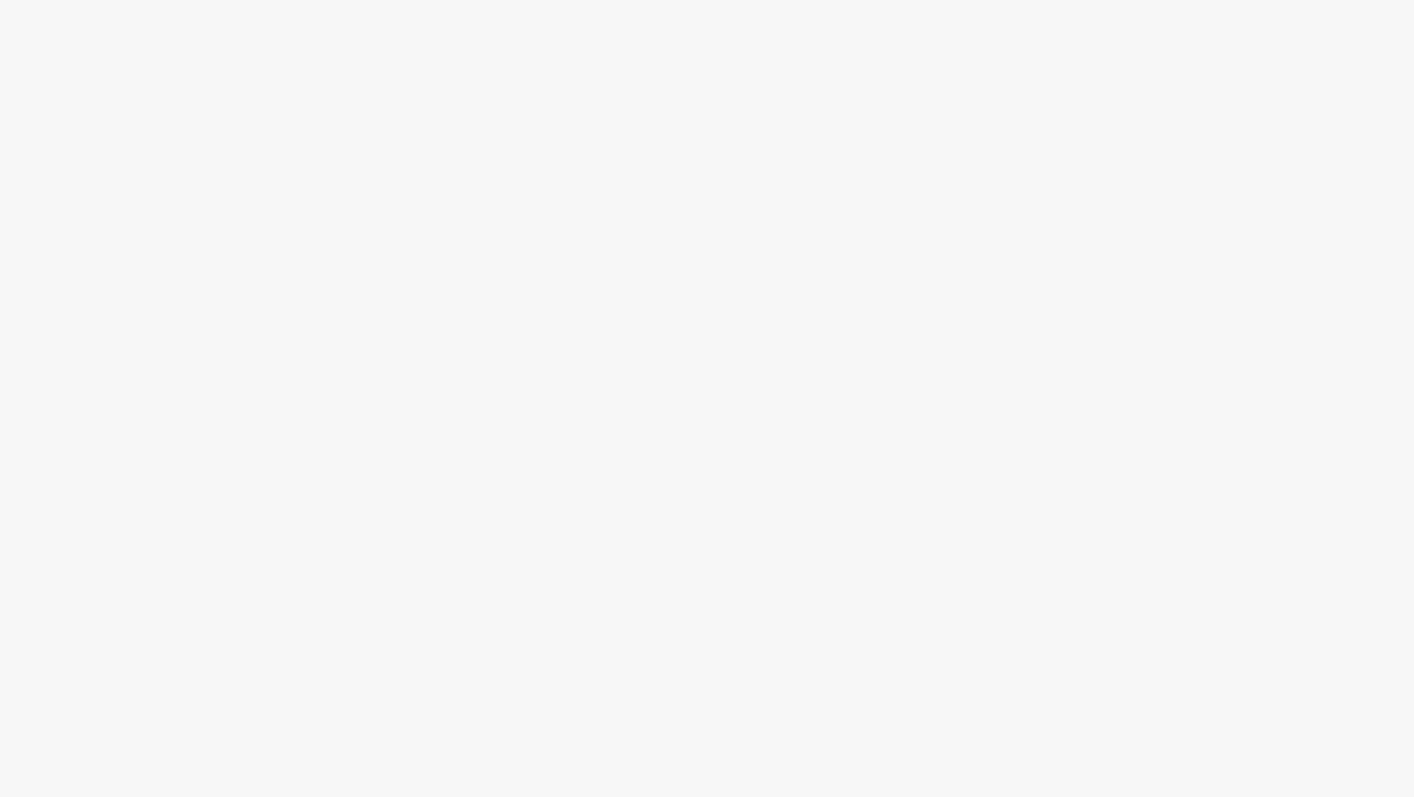
// -> [ "uvas", "manzana", "banana", "pepino", "papa"]

const [FAV, ... otros] = alimentos;

consola.log (FAV); // -> "uvas"

consola.log (otros); // -> [ "manzana", "banana", "pepino", "papa"]

Para los objetos:



const miembro = {

nombre: 'Ben',

título: 'desarrollador de software',

habilidades: ['JavaScrip: t', 'reaccionar', Redux ''],

};

const memberWithMetadata = {

...miembro,

previousProjects: ['Cielo azul', 'RedOcean'];

};

// detrás de las escenas:

const memberWithMetadata = Objeto.assign (miembros, {previousProjects: ['Cielo azul', 'RedOc ean']});

consola.log (memberWithMetadata.name); // -> "Ben"

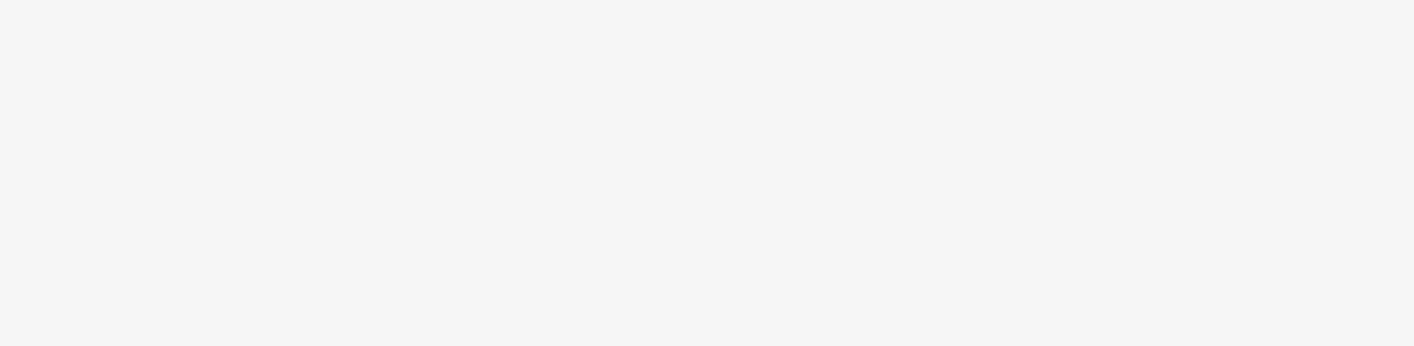
consola.Iniciar sesión(Objeto.keys (memberWithMetadata)); // -> [ "manzana", "banana", "pepino", "p APA "]

Para las llamadas a funciones:



17

desestructuración



const comida = ['uvas', 'manzana', 'plátano', 'Pepino', 'patata']; función comer() {

consola.log (... argumentos);

}

comer alimentos)

// -> uvas manzana pepino plátano patata

**Otras lecturas**

[MDN desestructuración](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)



[ES6 En Profundidad: desestructuración](https://hacks.mozilla.org/2015/05/es6-in-depth-destructuring/)



[La asignación desestructurada en ECMAScript 6](http://fitzgeraldnick.com/weblog/50/)



[Object.assign ()](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)



18

Funciones de flecha



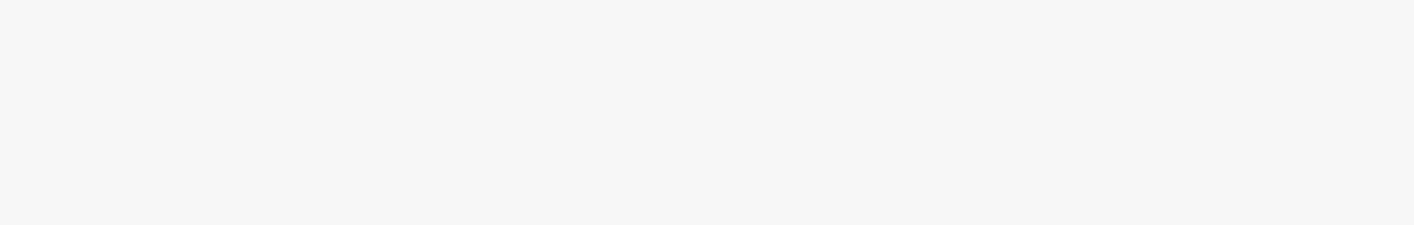
**Funciones de flecha**

ES6 ofrece una nueva sintaxis para hacer frente a esta: "Flecha funciones". Flecha función también hacen el trabajo con funciones de "orden superior" (funciones que tienen funciones como parámetros) mucho más fácil trabajar con ellos.



La nueva notación "flecha de grasa" se puede utilizar para definir funciones anónimas de una manera más simple.

Consideremos el siguiente ejemplo:



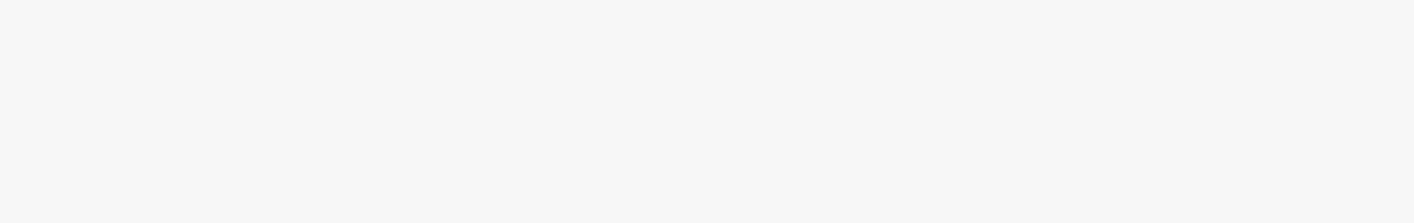
items.forEach (función(X) {

consola.log (x);

incrementedItems.push (x +1);

});

Esto puede ser reescrita como una "función de la flecha" con la siguiente sintaxis:



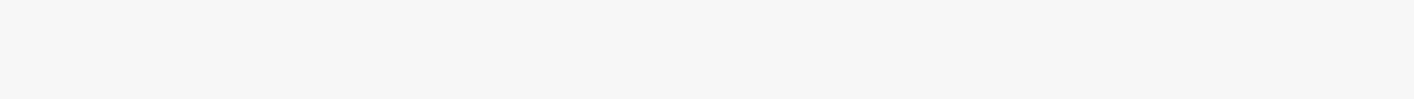
items.forEach ((x) => {

consola.log (x);

incrementedItems.push (x +1);

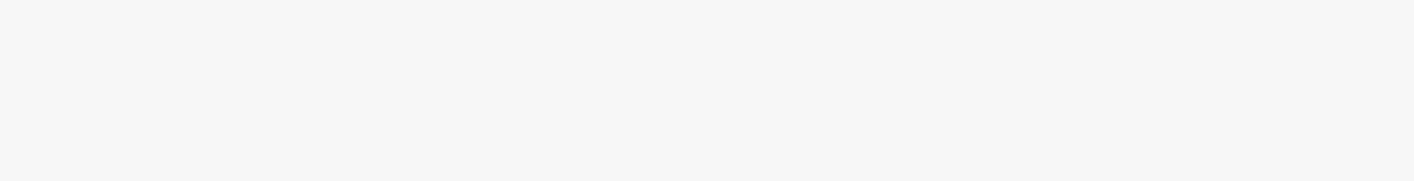
});

Funciones que calculan una sola expresión y regresan a sus valores se pueden definir aún más simple:



incrementedItems = items.map ((x) => x +1);

Este último es casi equivalente a la siguiente:



incrementedItems = items.map (función (X) {

regreso x +1;

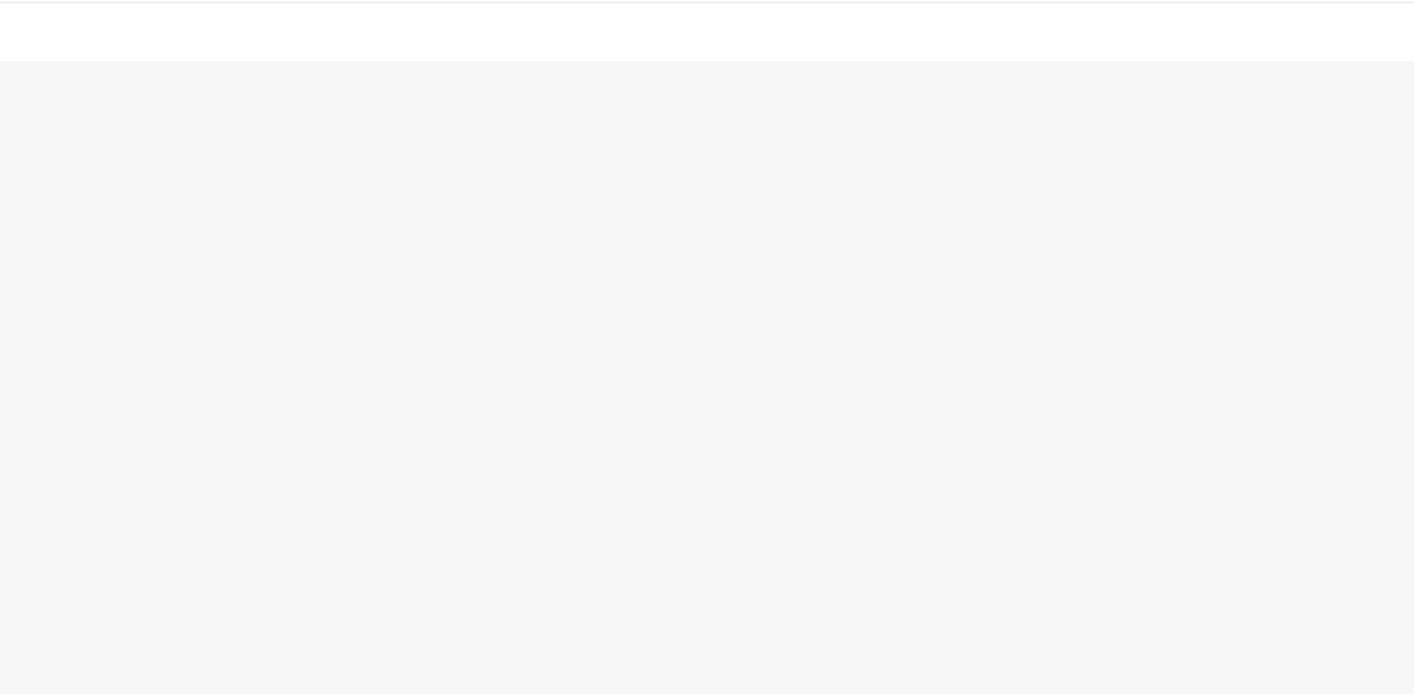
});

Hay una diferencia importante, sin embargo: funciones de dirección no establecen una copia local de esta , argumentos , súper o new.target . Cuando esta se utiliza dentro de una función de flecha la utiliza JavaScript estadel ámbito de aplicación externa. Consideremos el siguiente ejemplo:



19

Funciones de flecha



clase toppings {

constructor(coberturas) {

this.toppings = Formación.isArray (ingredientes)? ingredientes: [];

}

outputlist () {

this.toppings.forEach (función(superando, i) {

consola.log (superando, i + '/' + this.toppings.length); // no este })

}

}

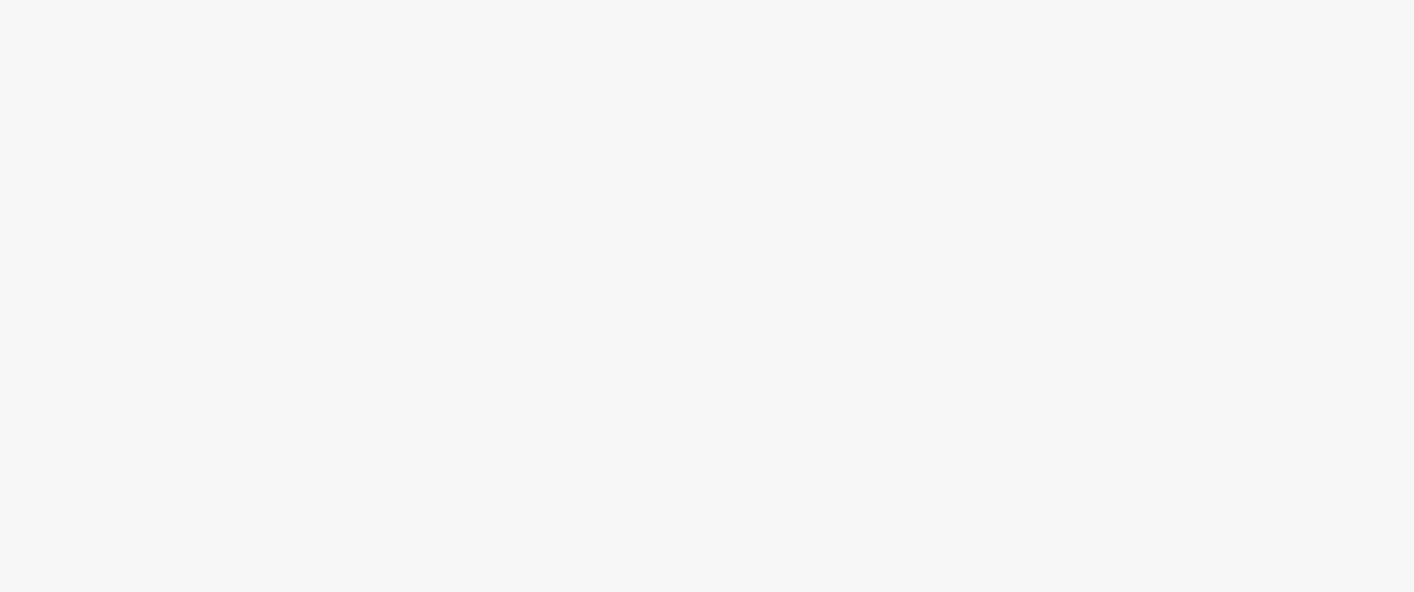
var ctrl = nuevo Ingredientes (['queso', 'lechuga']);

ctrl.outputList ();

Vamos a probar este código en ES6 violín (<http://www.es6fiddle.net/>). Como vemos, esto nos da un error, yathis no está definido dentro de la función anónima.



Ahora, vamos a cambiar el método para utilizar la función de flecha:



class Toppings {

constructor(toppings) {

this.toppings = Array.isArray(toppings) ? toppings : [];

}

outputList() {

this.toppings

.forEach ((superando, i) => console

.log (superando, i + '/' + this.toppings.length); // `this` funciona!

)

}

}

var ctrl = new Toppings(['cheese', 'lettuce']);

aquí this dentro de la función de flecha se refiere a la variable de instancia.



*Advertencia* funciones de dirección no tienen su propio argumentosveteranos programadores de JavaScript. súper y new.target recinto.



variables, esto puede ser confuso para también están en el ámbito de la externa

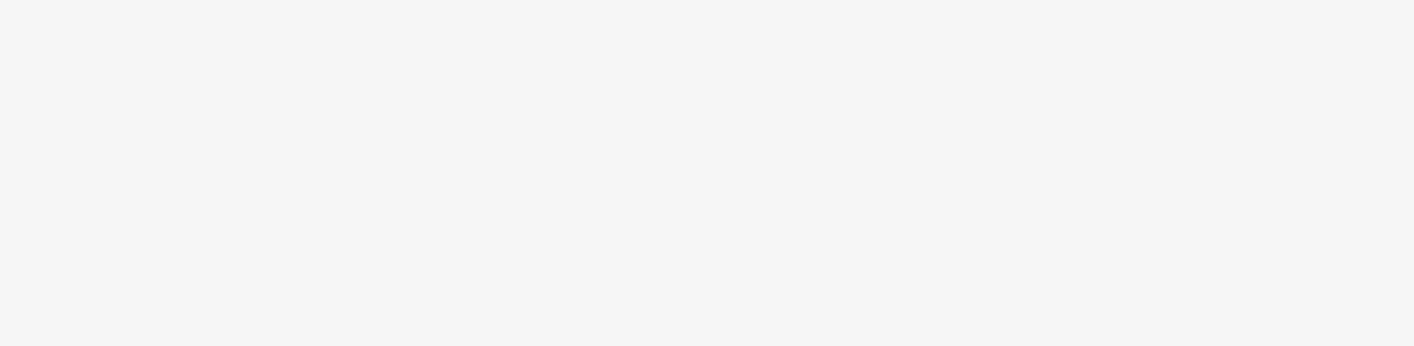
20

promesas



**promesas**

Las promesas están incorporados ES6.



const Wait (ms) => {

return new Promesa((Resolver, rechazar) => {

setTimeout (resolución, ms);

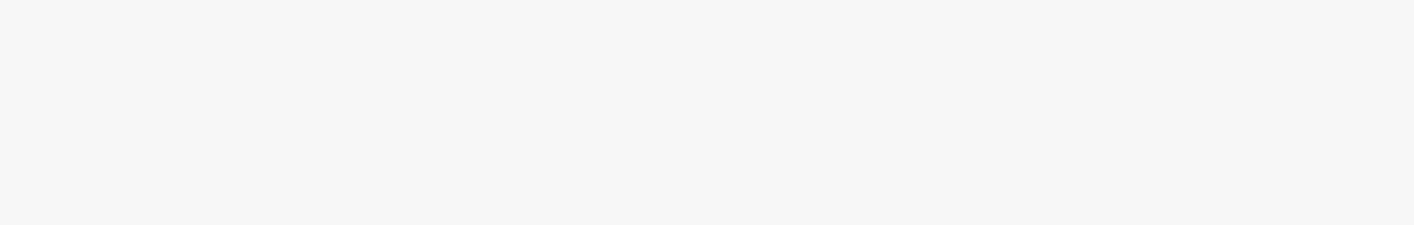
});

}

Espere(1000) .then (() => consola.Iniciar sesión('garrapata'));

**Promesas frente a las devoluciones de llamada**

Para las solicitudes HTTP, nuestra solución existente es utilizar devoluciones de llamada:



solicitud (url, (error, la respuesta) => {

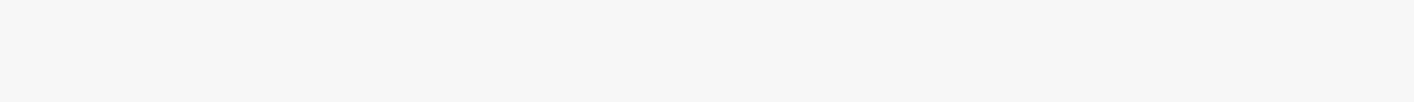
* manejar el éxito o error.

});

hacer algo más();

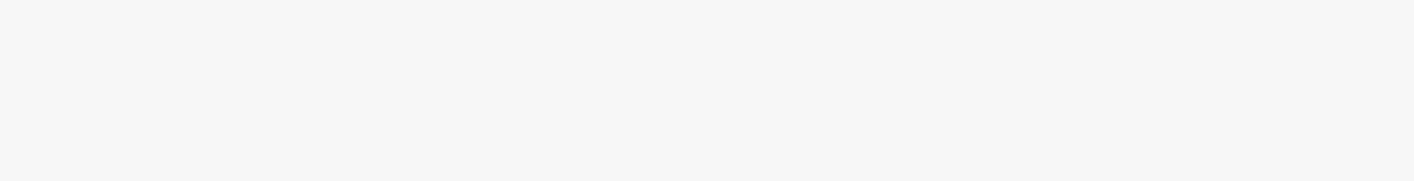
Existen algunos problemas con las devoluciones de llamada. Uno es conocido como["Devolución de llamada Hell"](http://callbackhell.com/). Un problema mayor es la descomposición.

El patrón de devolución de llamada requiere que especifique la tarea y la devolución de llamada al mismo tiempo. Por el contrario, las promesas que nos permiten especificar y despachar la solicitud en un solo lugar:



prometer = fetch (url); // ir a buscar es un reemplazo para XMLHttpRequest

y luego añadir la devolución de llamada más tarde, y en un lugar diferente:



promise.then (respuesta => {

* manejar la respuesta.

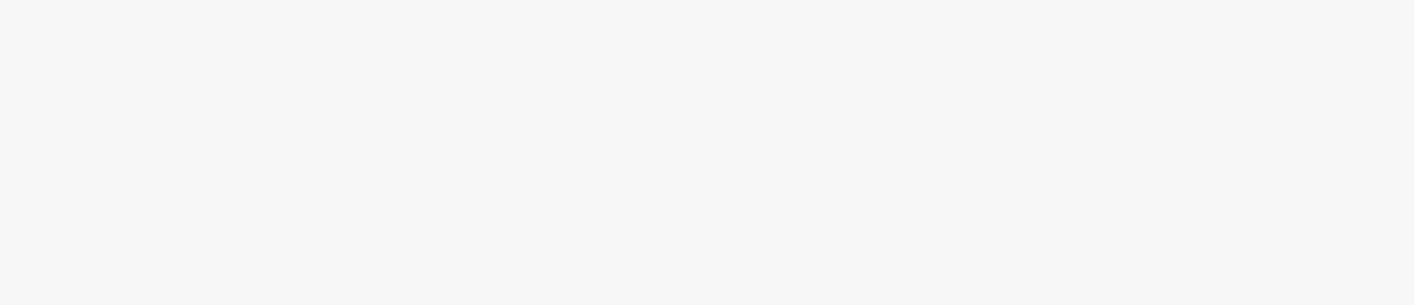
});

Esto también nos permite asociar varios controladores a la misma tarea:



21

promesas



promise.then (respuesta => {

* manejar la respuesta.

});

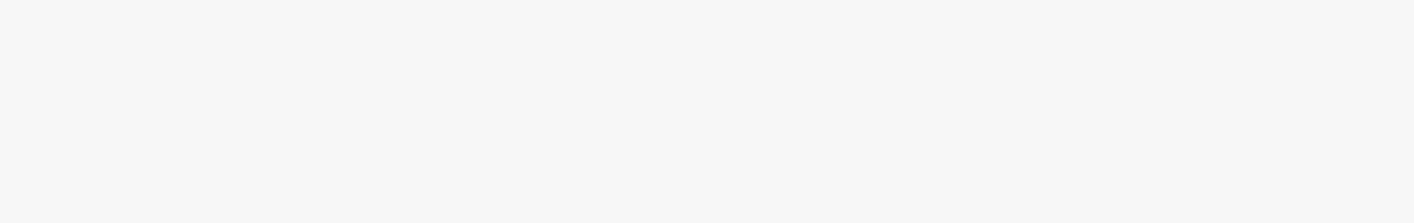
promise.then (respuesta => {

* hacer otra cosa con la respuesta.

});

**Más en promesas**

.entonces() siempre devuelve una promesa. Siempre.



p1 = getDataAsync (consulta);

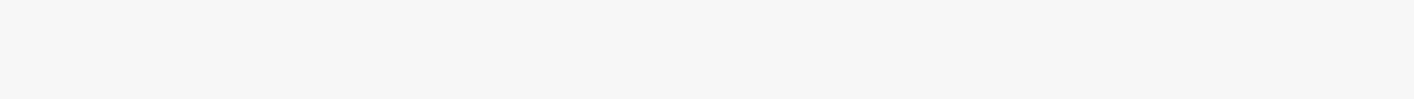
p2 = p1.then (

Resultados => transformData (resultados));

p2 ahora es una promesa, independientemente de lo transformData () devuelto. Incluso si algo falla.



Si la función de devolución de llamada devuelve un valor, la promesa se resuelve a ese valor:

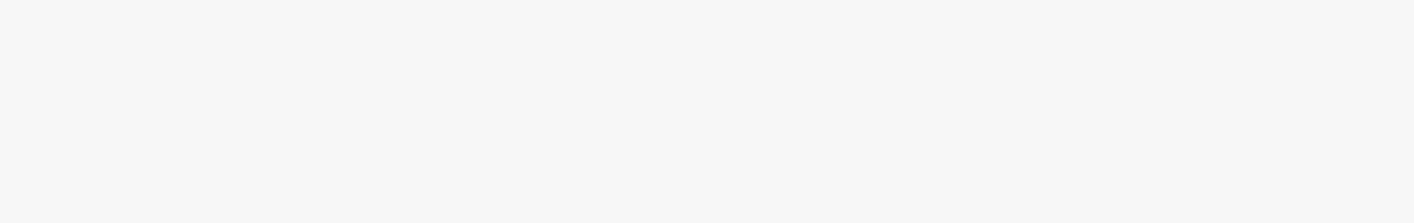


p2 = p1.then (resultados => 1);

p2 resolverá a “1”.



Si la función de devolución de llamada devuelve una promesa, la promesa se resuelve en una promesa funcionalmente equivalente:



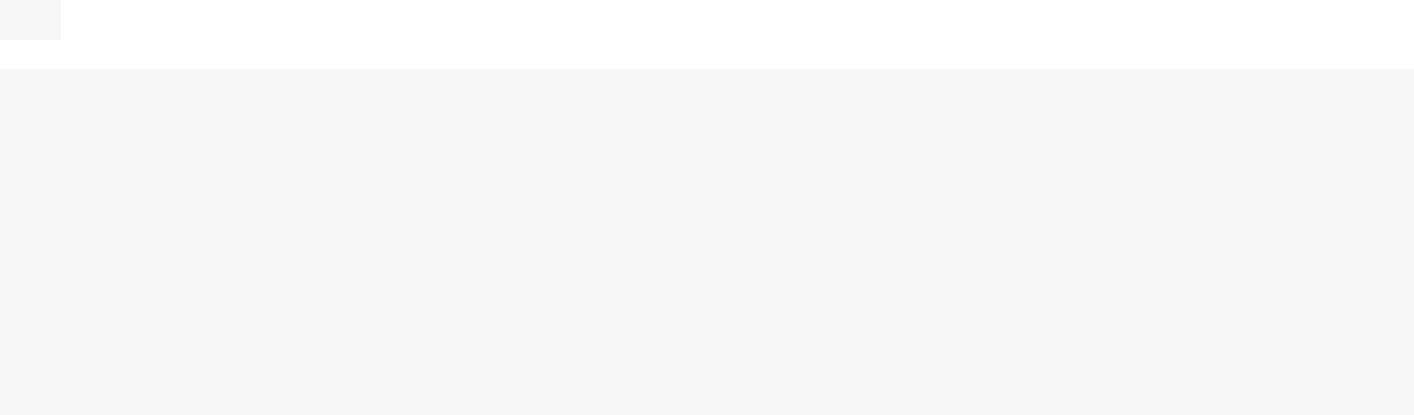
p2 = p1.then (resultados => {

dejar newPromise = getSomePromise ();

regreso newPromise;

});

p2 ahora es funcionalmente equivalente a newPromise.



p2 = p1.then (

Resultados => lanzar Error('Vaya'));

p2.then (resultados => {

* Se le pregunta por qué esto nunca es
* llamado.

});

p2 sigue siendo una promesa, pero ahora será rechazada con el error arrojado.



22

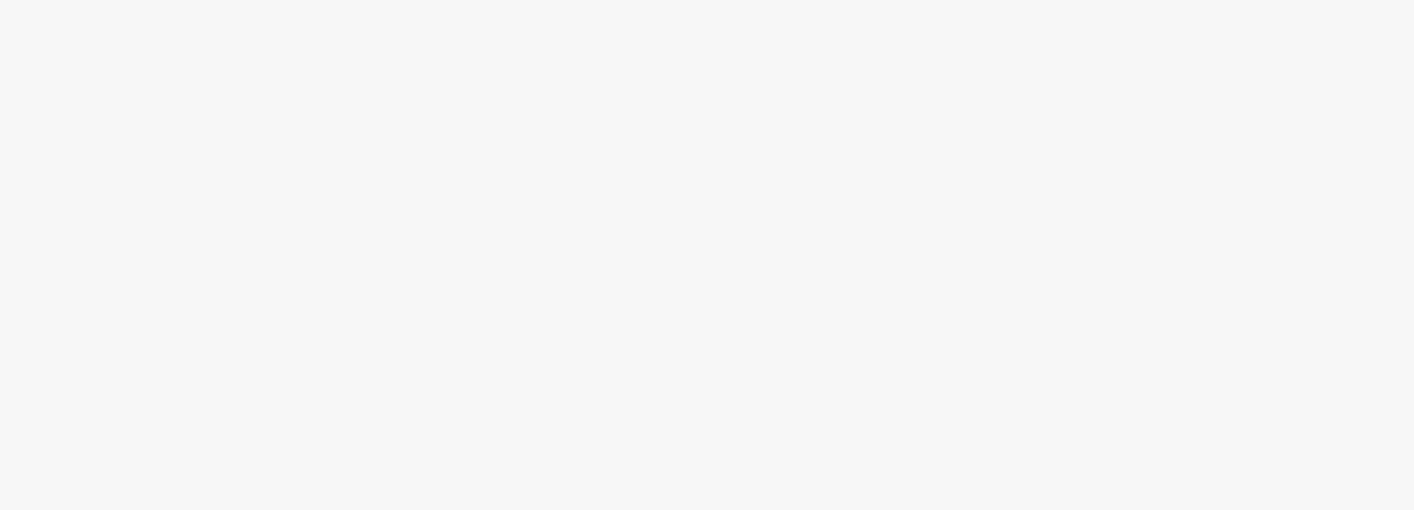
promesas



¿Por qué no va a la segunda devolución de llamada siempre se llamará?

**La captura de Rechazos**

La función pasa a entonces toma un segundo argumento, es decir, error , Que representa la captura de error dentro de la cadena promesa.



ha podido recuperar('Http://ngcourse.herokuapp.com/api/v1/tasks')

.then (respuesta => response.data)

.A continuación (tareas => filterTasksAsynchronously (tareas))

.A continuación (tareas => {

$ Log.info (tareas);

vm.tasks = tareas;

})

.entonces(

nulo,

error => log.error (error)

);

Tenga en cuenta que una pega al final es a menudo suficiente.

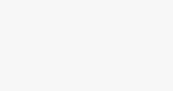


23

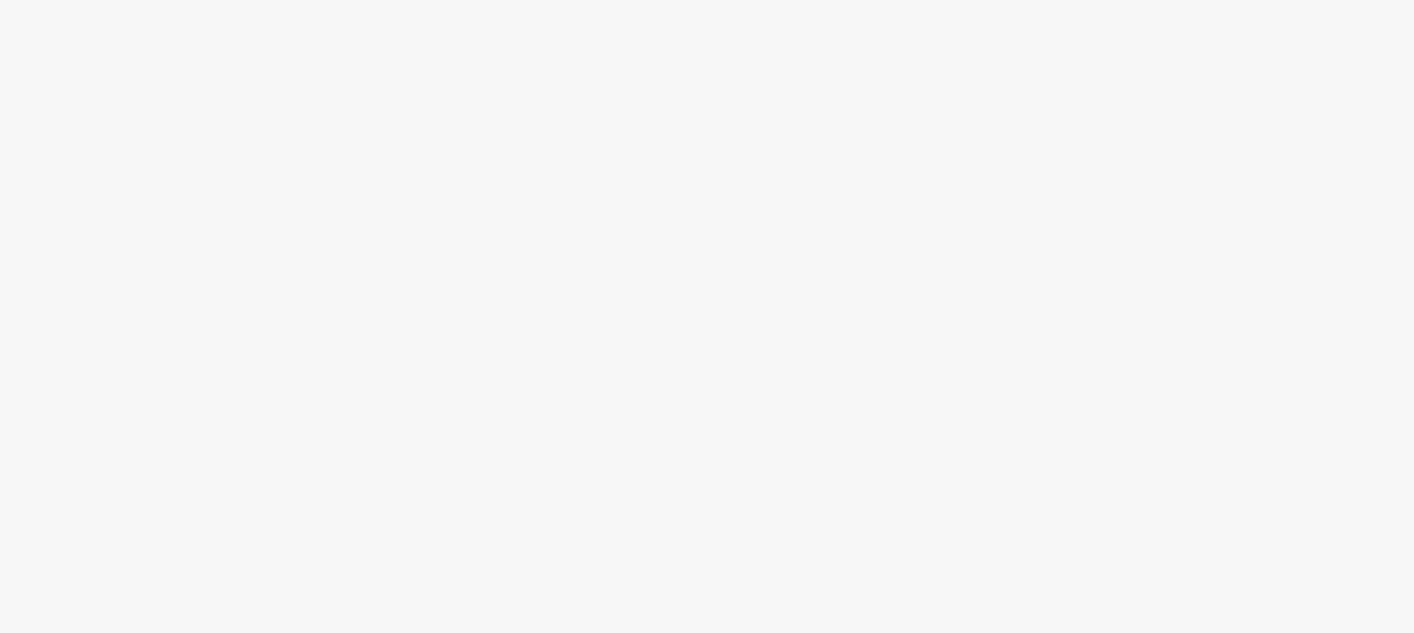
Dejar y const



**dejar** **y** **const**



ES6 introduce el concepto de bloque de alcance. Bloque de alcance será familiar para los programadores de otros lenguajes como C, Java, o incluso PHP. En ES5 JavaScript, y anteriores,var s están en el ámbito de función s, y pueden "ver" fuera de sus funciones al contexto externo.



var cinco = 5;

var threeAlso = tres; // error

función Ámbito de aplicación1() {

var tres = 3;

var fiveAlso = cinco; // == 5

var sevenALso = siete; // error

}

función scopt2() {

var siete = 7;

var fiveAlso = cinco; // == 5

var threeAlso = tres; // error

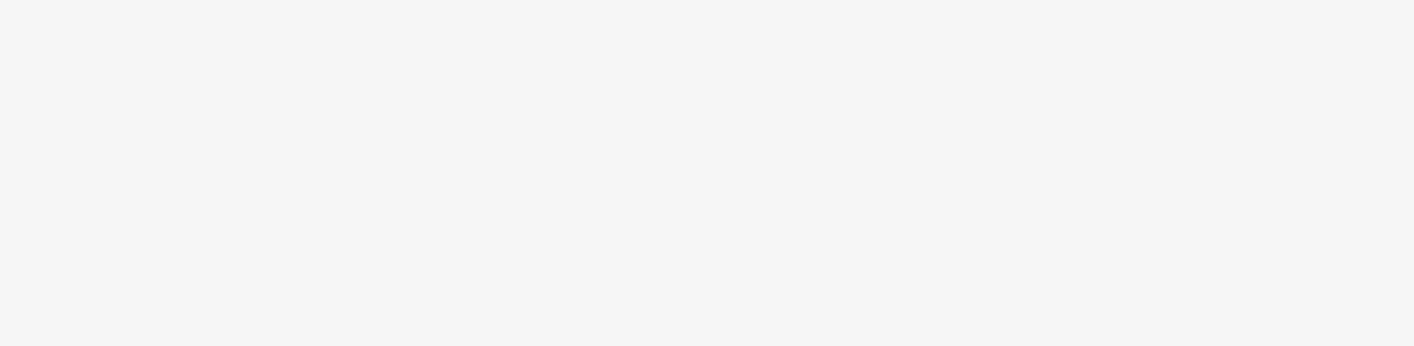
}

En ES5 funciones eran esencialmente contenedores que podrían ser "visto" fuera de, pero no en.

en ES6 var todavía funciona de esa manera, el uso de funciones como contenedores, pero hay dos nuevas formas de declarar las variables: const y dejar . const y dejar utilizar { y } bloques como contenedores, por lo tanto "ámbito de bloque".



Bloque de alcance es más útil durante bucles. Considera lo siguiente:



var yo;

para (I = 0; i < 10; i + = 1) {

var j = i;

dejar k = i;

}

consola.log (j); // 9

consola.log (k); // indefinido

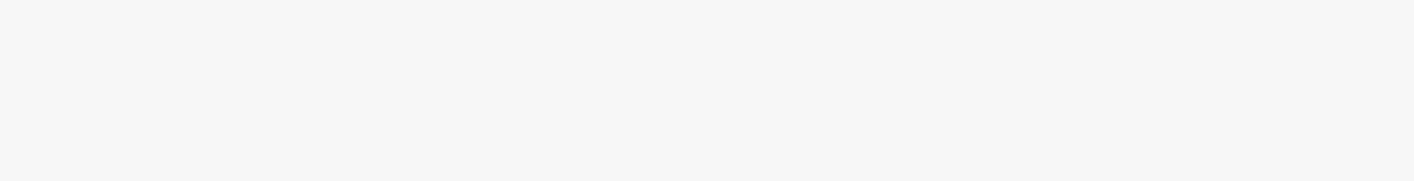
A pesar de la introducción de bloques de alcance, las funciones son todavía el mecanismo preferido para hacer frente a la mayoría de los bucles.

dejar funciona como var en el sentido de que sus datos es de lectura / escritura. dejar también es útil cuando se usa en un bucle. Por ejemplo, sin dejar que:



24

Dejar y const

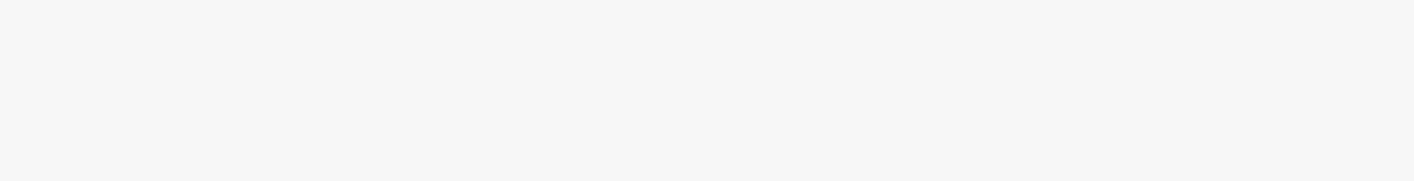


para(var x =0; X <5; X ++) {

setTimeout (() =>consola.log (x), 0)

}

emitiría 5,5,5,5,5. Sin embargo, cuando se usadejar en lugar de var , El valor sería de ámbito de una manera que la gente espera.



para(dejar x =0; X <5; X ++) {

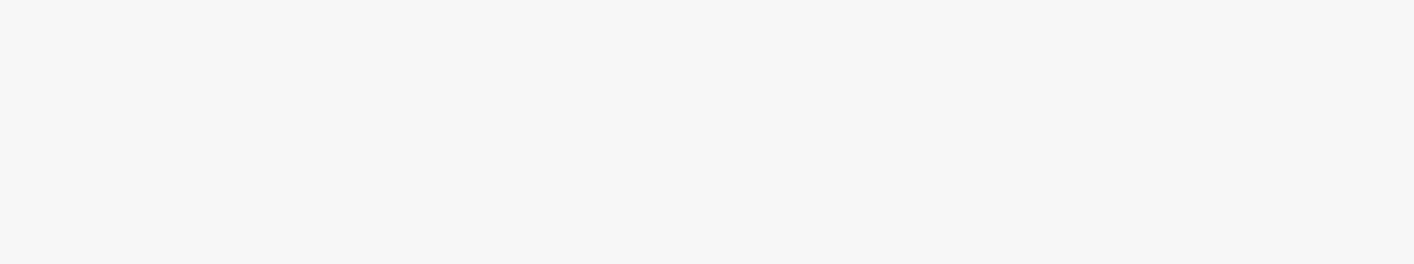
setTimeout (() =>consola.log (x), 0)

}

Alternativamente, constes de sólo lectura. Una vezconst se le ha asignado, el identificador no puede ser re-asignado, el valor es [no es inmutable](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const).



Por ejemplo:

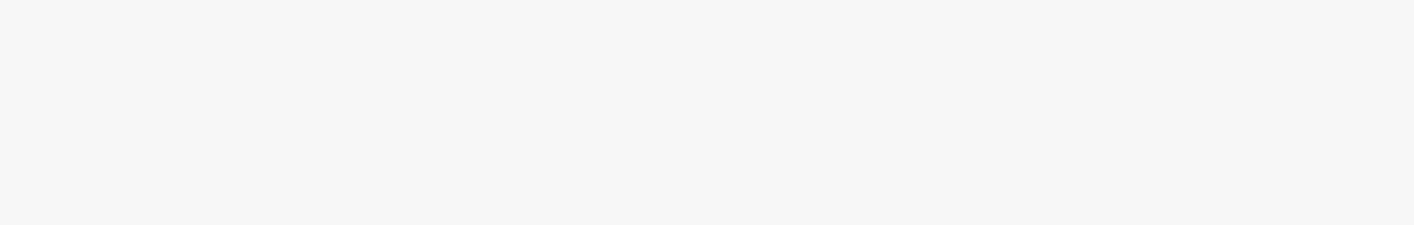


const myName = 'palmadita';

dejar yourname = 'Jo';

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | tu nombre | = 'Sam'; // | cesionarios | |  |
|  |  |  |  |  |  |
| myName = | 'ene'; // | error |  |

La única naturaleza lectura se puede demostrar con cualquier objeto:



const literal = {};

literal.attribute = 'prueba'; // fina

literal = []; // error;



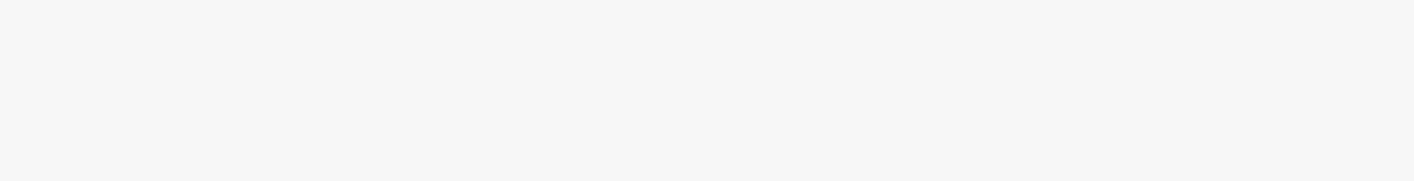
25

módulos



**módulos**

ES6 también introduce el concepto de un módulo, que funciona de manera similar a otros idiomas. La definición de un módulo ES6 es muy fácil: cada archivo se supone que definir un módulo y vamos a especificar los valores que está utilizando la palabra clave de exportación exporta.



importar Analizar gramaticalmente de 'analizar gramaticalmente';

importar {} MyFunc de './myModule';

importar \* como mi\_módulo de './myOtherModule';

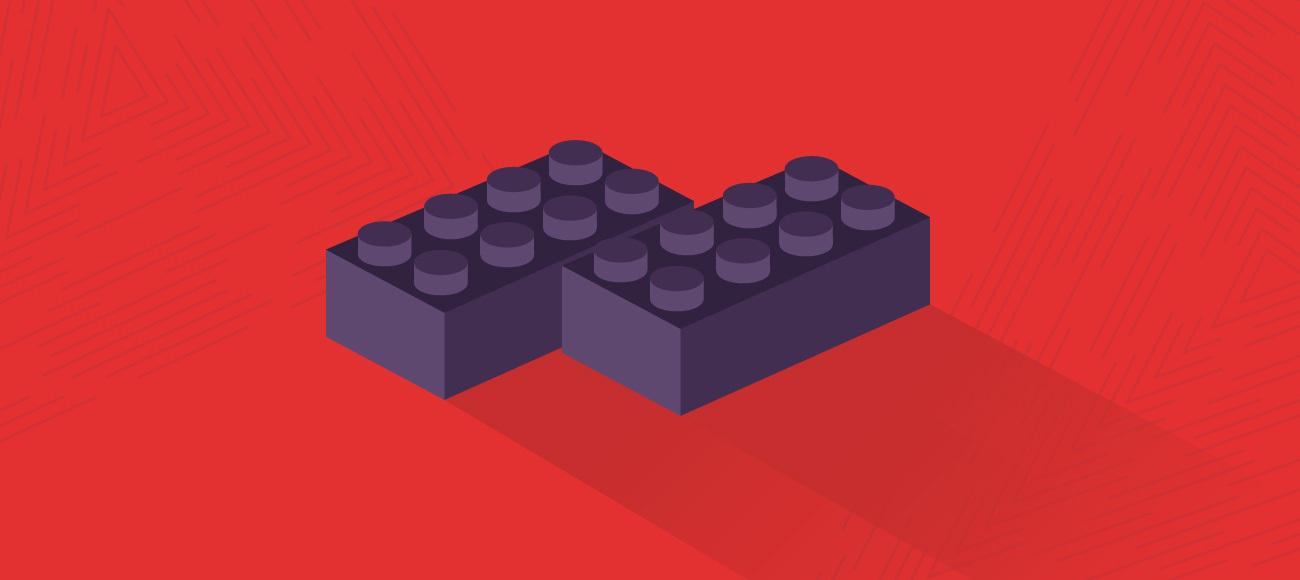


26

Pensando en Componentes



**Pensando en Componentes**



27

Diseño atómica



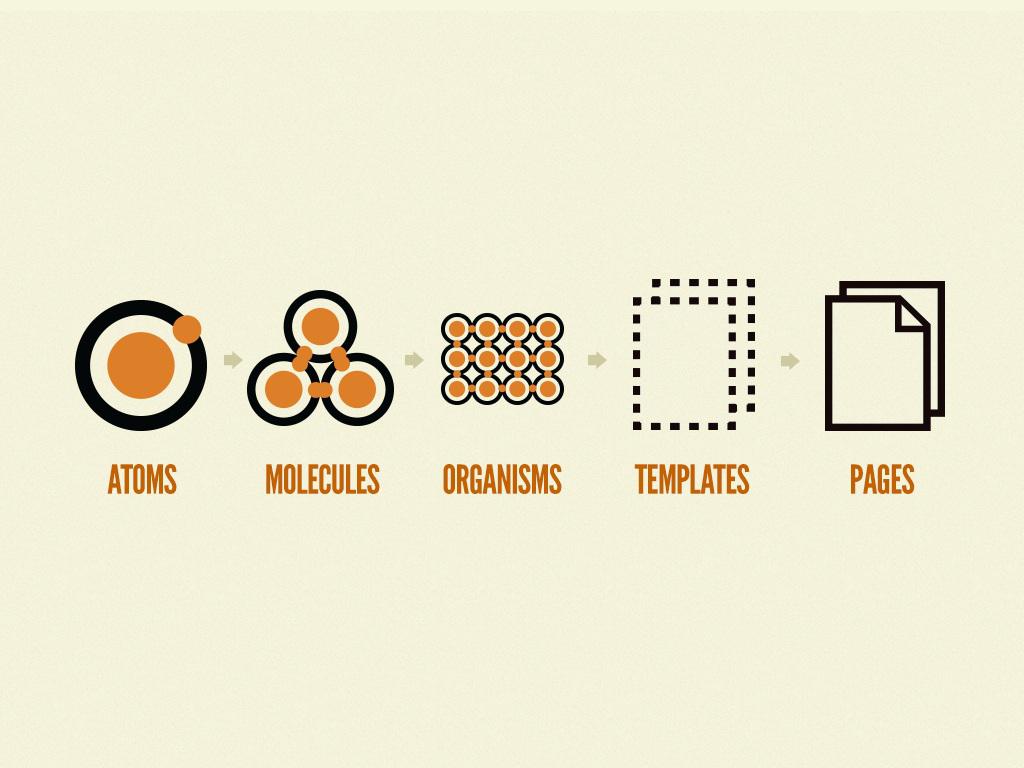
**Diseño atómica**

<http://bradfrost.com/blog/post/atomic-web-design/>



*No estamos diseñando páginas, estamos sistemas de componentes de diseño* - Stephen heno

**Principios de diseño atómicas**



Pero estamos diseñando una aplicación de una sola página. Nunca debemos pensar en nuestras interfaces como páginas.

En cambio, debemos pensar en términos de abstracciones dinámicas, sin estado, que se representan en función del estado proporcionado

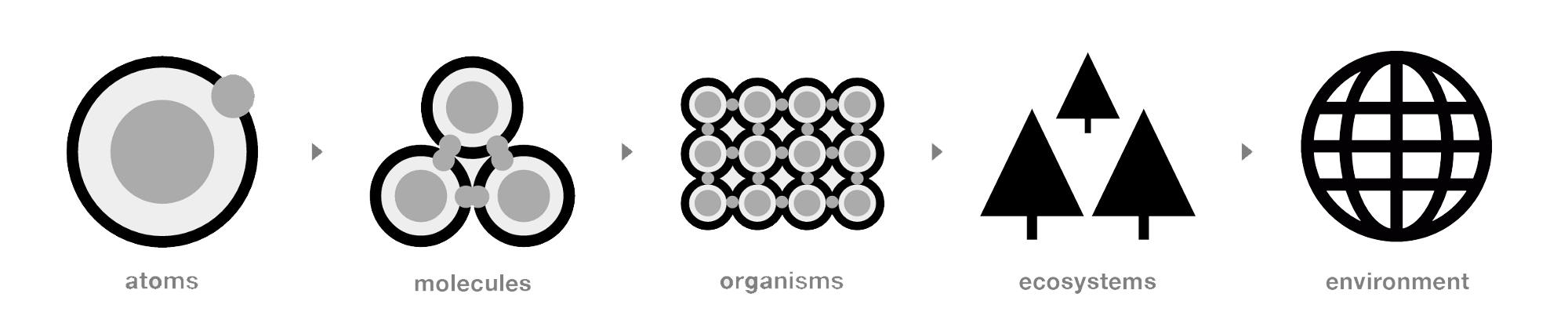


28

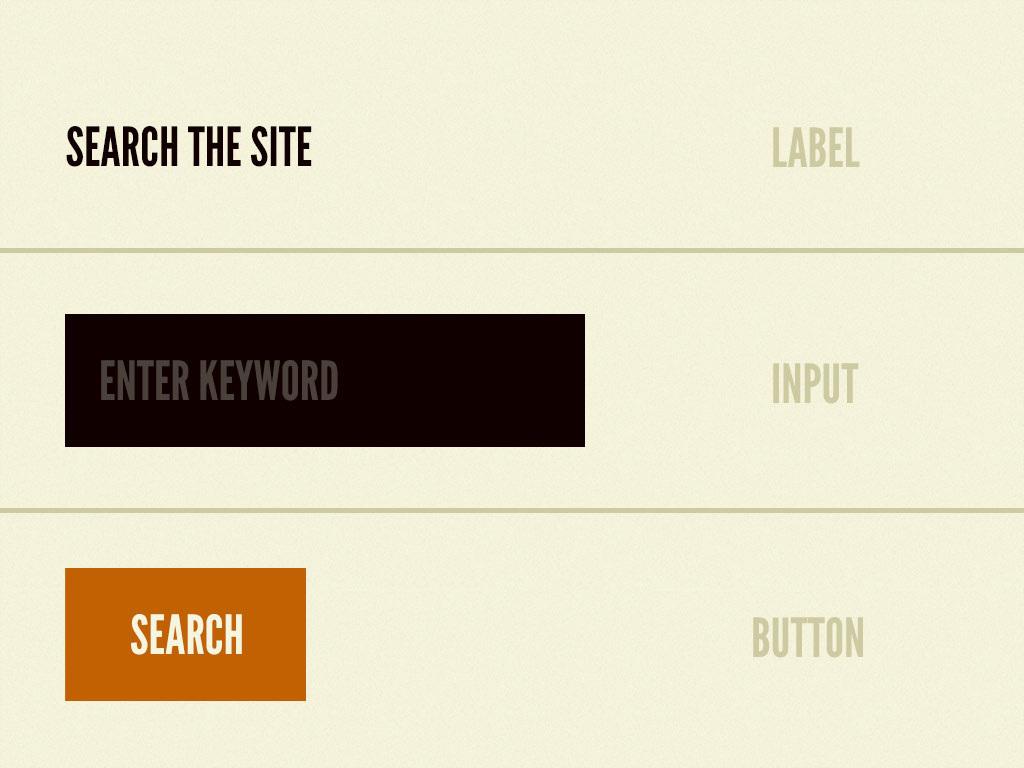
Principios de componentes atómicos



**Principios de componentes atómicos**



**Los átomos**



El bloque de construcción más simple



etiquetas HTML



No es muy útil por su cuenta



facilidad de estilo



muy reutilizable

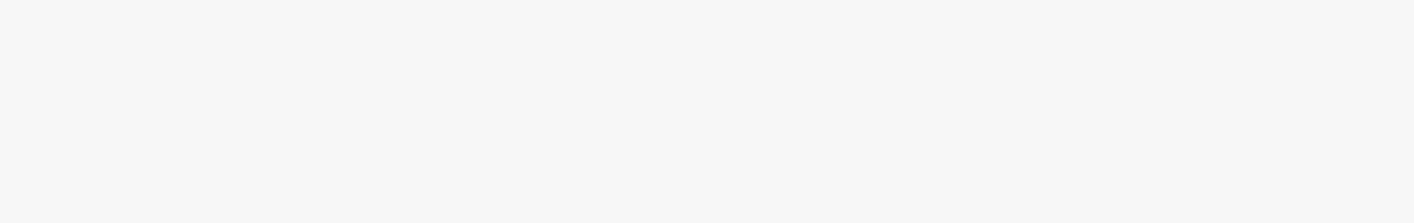


Fundación de la construcción de una marca



29

Principios de componentes atómicos



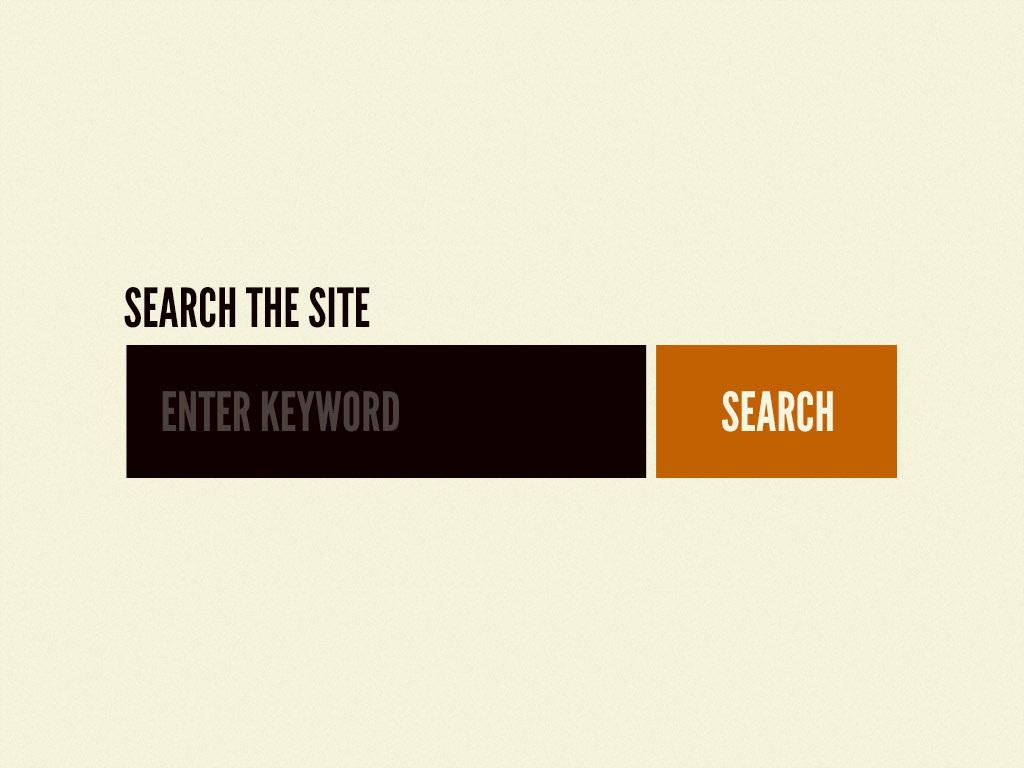
<Form>

<Label>Buscar</ Etiqueta>

<Input />

</ Form>

**Moléculas**



Grupos de átomos unidos



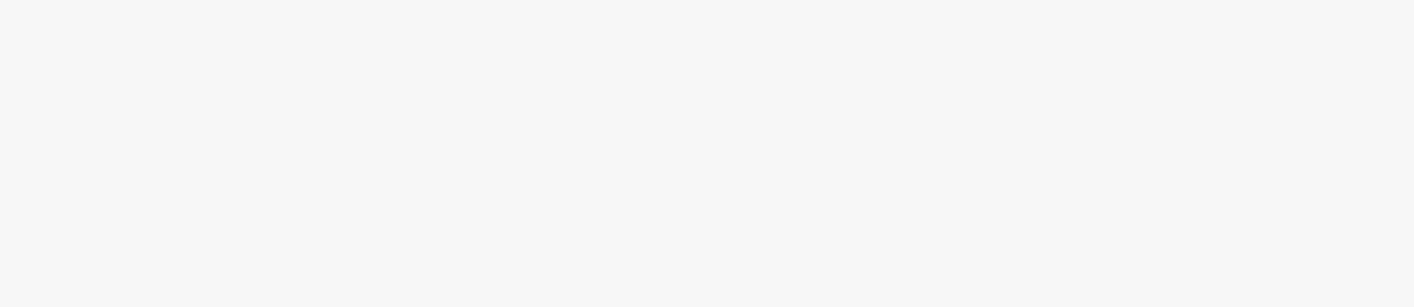
Servir como columna vertebral del sistema de diseño



Por ejemplo, un formulario de búsqueda



Hacer una cosa, hacerlo bien



<Forma onSubmit ={ onSubmit}>

<Label>Buscar</ Etiqueta>

<Input type ="texto" valor ={ Búsqueda} />

<Botón type ="enviar">Buscar</ Botón>

</ Form>



30

Principios de componentes atómicos



**organismos**



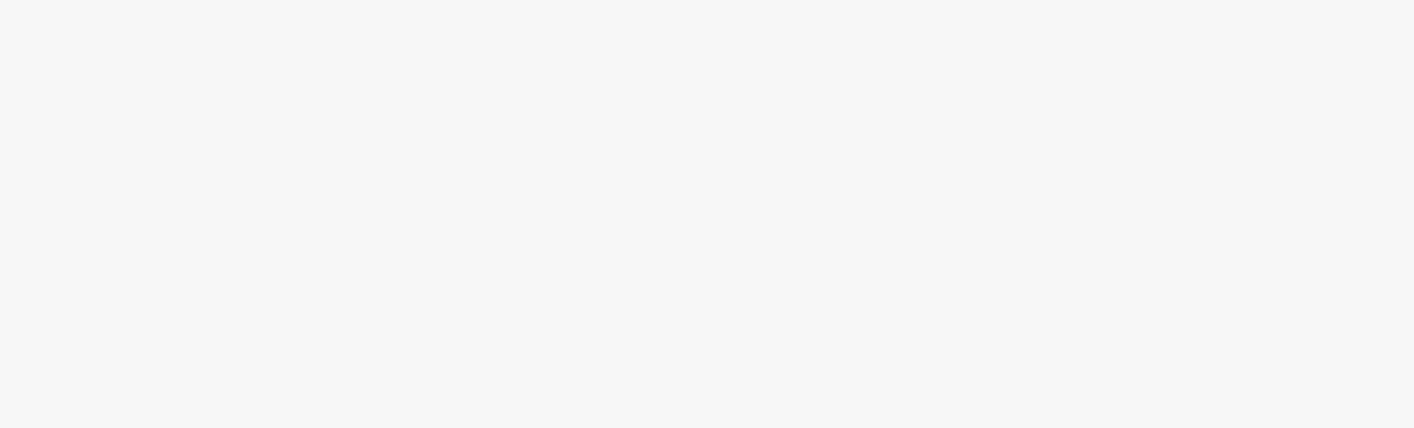
Grupos de moléculas



sección Distinct de una interfaz



Portátil, fácil de modificar



<Header>

<Navigator>

<Marca />

<NavItem a ="casa">Casa</ NavItem>

<NavItem a ="acerca de">Acerca de</ NavItem>

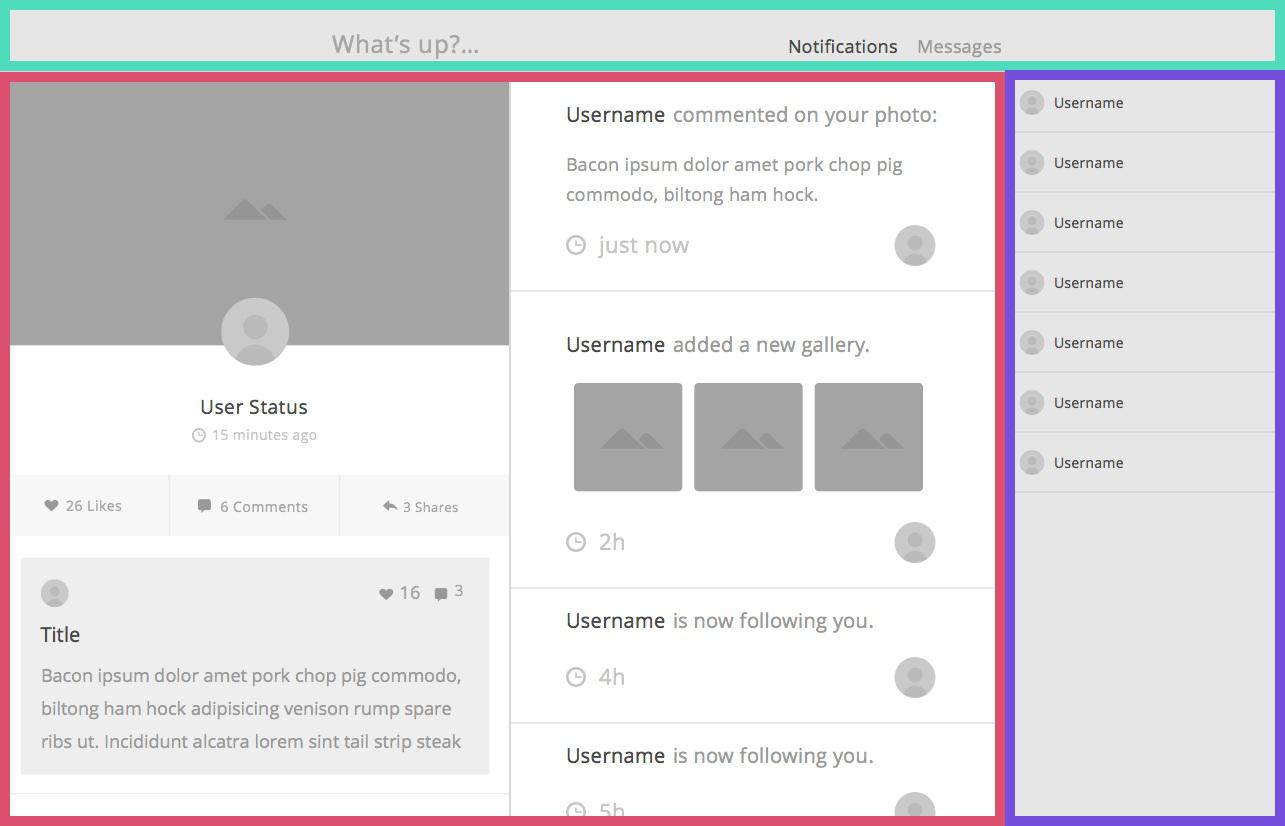
<NavItem a ="Blog">Blog</ NavItem>

</ Navigator>

<SearchForm />

</ Header>

**Ecosistema**



Lo que el cliente verá



31

Principios de componentes atómicos



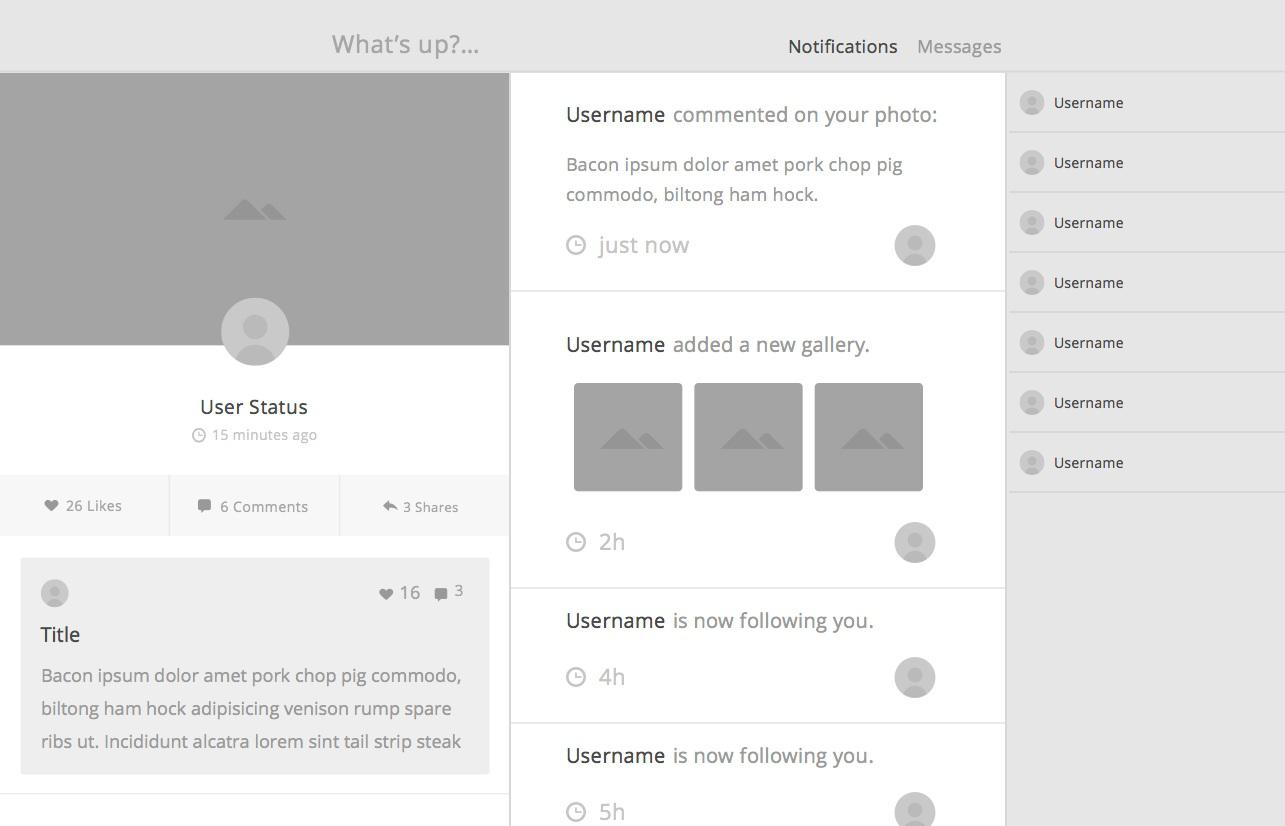
contenedores conectados



Muchos de los componentes que conforman una visión



**Ambiente**



Componente de raíz



normalmente, el <App /> componente



Representa todo lo envasados ​​juntos como una aplicación



Imágenes de:

El artículo de Brad heladas, [Diseño atómica](http://bradfrost.com/blog/post/atomic-web-design/)



[El artículo de Joey Di Nardo, Componentes atómica: Gestión Dinámica Reaccionar Componentes](https://medium.com/@yejodido/atomic-components-managing-dynamic-react-components-using-atomic-design-part-1-5f07451f261f#.9b2faky8s) [Diseño usando Atómica - Parte 1.](https://medium.com/@yejodido/atomic-components-managing-dynamic-react-components-using-atomic-design-part-1-5f07451f261f#.9b2faky8s)



32

Beneficios de este enfoque



**Los beneficios de este enfoque**

Principio de Responsabilidad Individual



Los componentes modulares



Construir una Marca



Walmart electrodo



33

El proceso



**El proceso**



**Comience con un Mock**

Romper la interfaz de usuario en una jerarquía



Principio de Responsabilidad Individual



Pensar en términos de Arquitectura de la Información



Los átomos primero, las moléculas de segundo



**componente Jerarquía**



34

El proceso



1.  FilterableProductTable : Contiene la totalidad del ejemplo
2.  Barra de búsqueda : Recibe toda entrada de usuario
3.  ProductTable : muestra y filtra la recogida de datos sobre la base de la entrada de usuario
4.  ProductCategoryRow : Muestra un encabezado para cada categoría
5.  ProductRow : Muestra una fila para cada producto



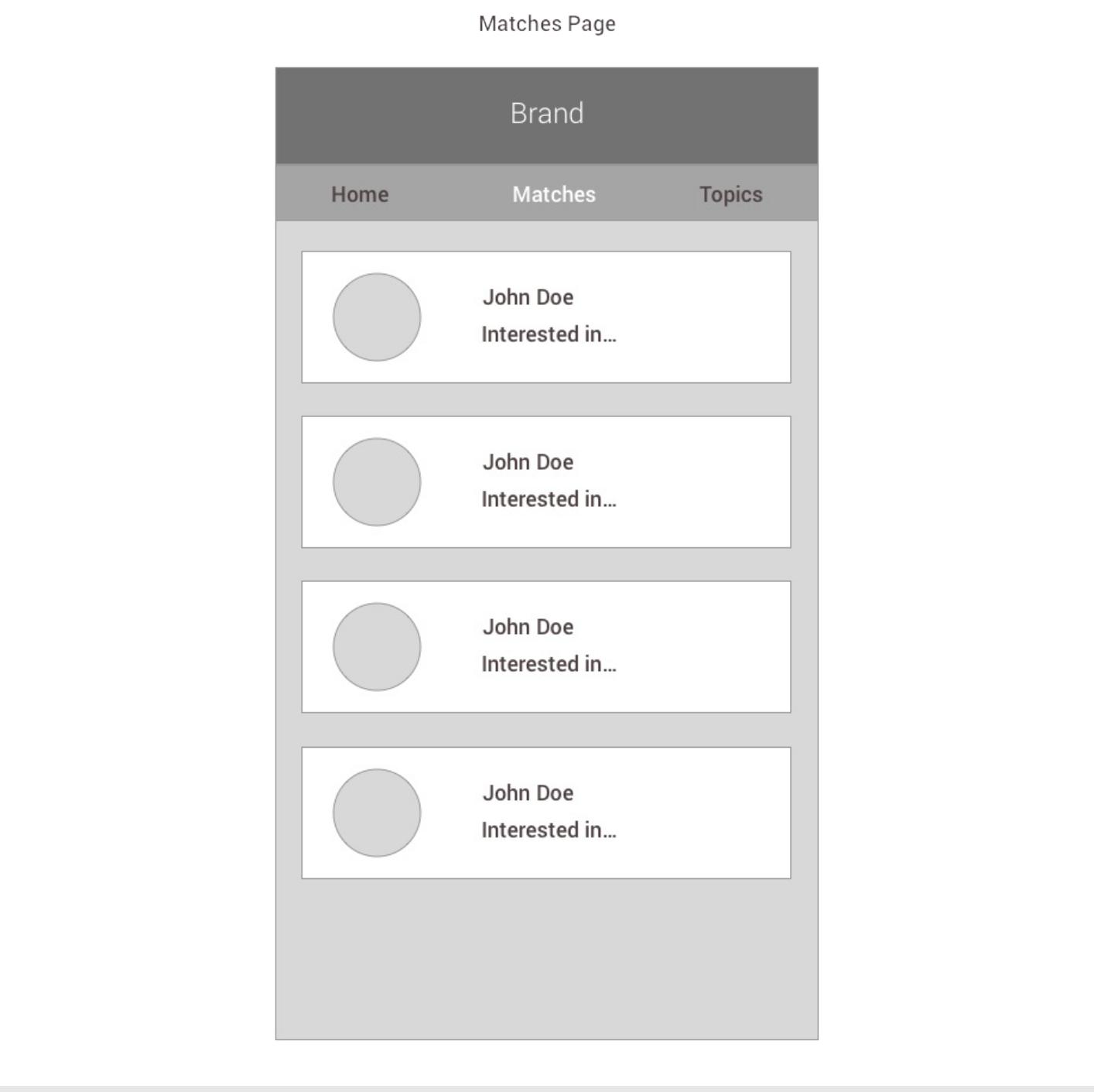
35

Tarea 1



**Tarea 1**

Hueco por la maqueta



**Respuesta 1**

Pase el ratón para ver la respuesta.

* Encabezamiento
* Navegador
* NavigatorLink



36

Tarea 1



* Contenido
* profilecard
* Imagen de perfil

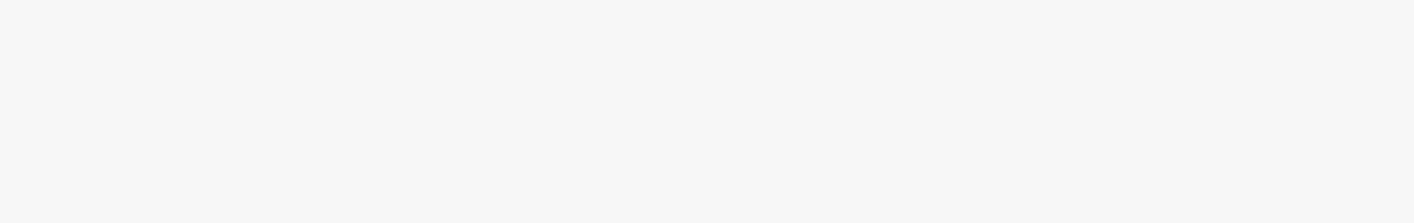


37

reaccionar componentes



**reaccionar componentes**



git checkout 02-componente

jspm instalar

NPM plazo dev

Abierto src / index.js



38

componentes sin Estado



**componentes sin Estado**

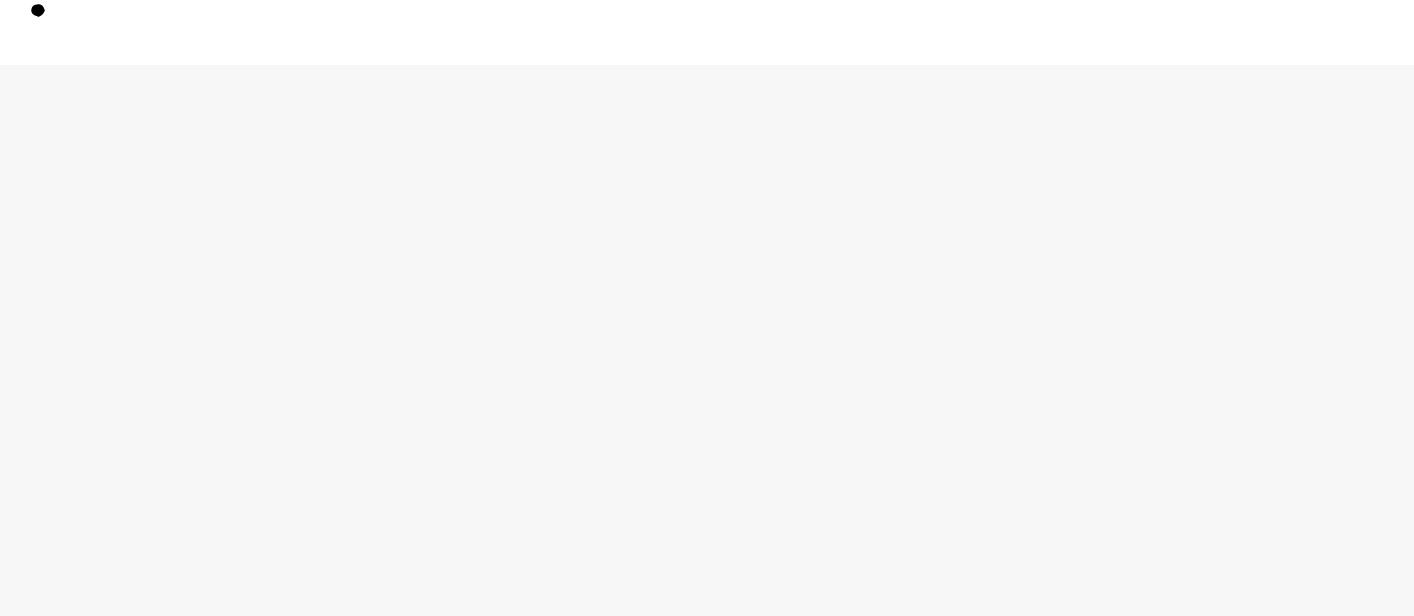
super sencillo



Un estado determinado (como accesorios) ... volver algún DOM (o componentes adicionales)



Puro



importar Reaccionar de 'reaccionar';

función HelloMessage({ nombre }) {

regreso(

<Div>Hola {nombre}</ Div>

);

};

HelloMessage.propTypes = {nombre: React.PropTypes.string};

HelloMessage.defaultProps = {nombre: 'Mundo' };

ReactDOM.render (<Nombre HelloMessage ="Alicia" />, MountNode);

**Útil 95% del tiempo**



39

Los componentes con estado

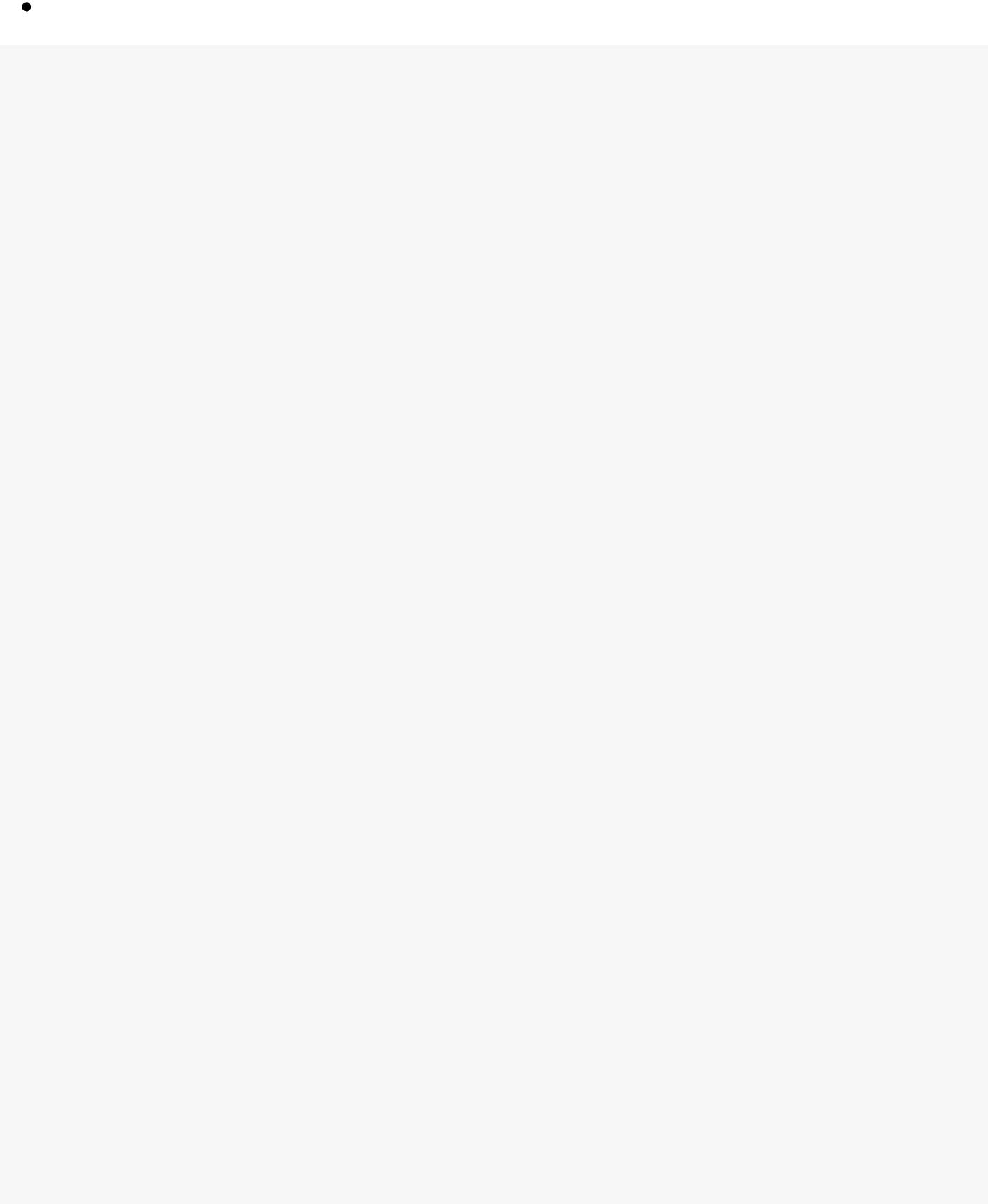


**Componente de estado**

Estado interno



Ganchos componente del ciclo de vida



importar Reaccionar, {} Componente de 'reaccionar';

clase Perfil se extiende Componente {

constructor(apoyos) {

súper(accesorios);

esta.state = {isLiked: falso };

}

componentDidMount () {

consola.Iniciar sesión('Componente con estado montado correctamente.');

}

\_toggleLikeState = () => {

esta.setState ({

isLiked: esta.state.isLiked

});

}

render () {

const {Nombre} = esta.accesorios;

const {IsLiked} = esta.estado;

regreso (

<Div>

<H3>{ nombre }</ H3>

<Span>{IsLiked? :} </ Span>

<Botón onClick ={ this.\_toggleLikeState}>

Al igual que alternar

</ Botón>

</ Div>

);

}

}

HelloMessage.propTypes = {nombre: React.PropTypes.string};

HelloMessage.defaultProps = {nombre: 'Mundo' };

ReactDOM.render (<Perfil name ="Alicia" />, MountNode);

**Es útil cuando ...**



40

Los componentes con estado



Tenemos que tener un estado interno



Tenemos que realizar una acción cuando el componente está montado



41

Stateful vs Componentes sin estado



**Stateful vs Componentes sin estado**

**stateful**

Cuando necesite estado interno



gráfico D3



Cuando tenga que utilizar un gancho de componentes del ciclo de vida



petición Ajax en el monte



animaciones de configuración



Acceder al nodo DOM prima para una biblioteca tercera parte



**Apátrida**

95% de las veces todo lo que necesita es un componente sin estado.

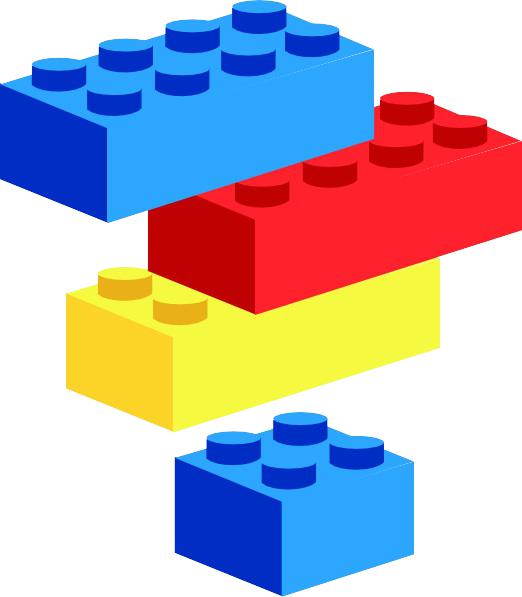


42

Composición



**Los componentes son Bricks**



Pensar en ellos como piezas de Lego



Construir pequeños componentes que juntos construyen componentes más grandes, pantallas

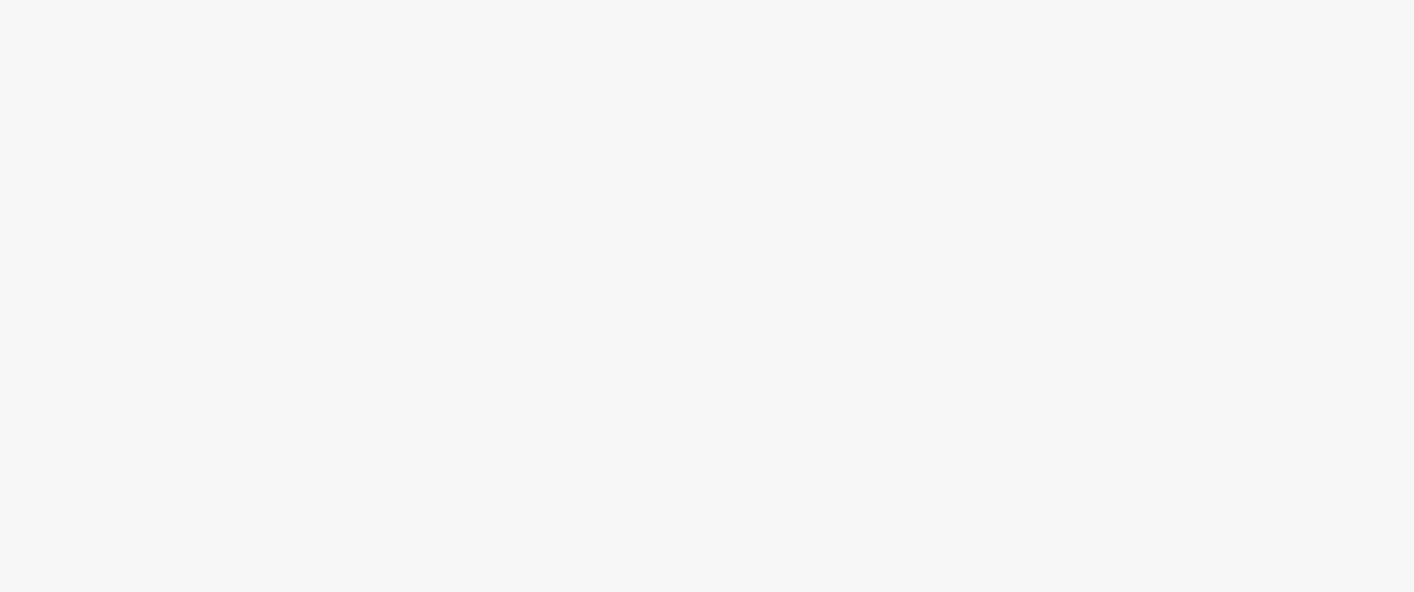


La composición es la clave!



**Composición**

Componemos estos ladrillos para construir cosas más grandes



<! - LoginForm.js ->

<Form>

<FormGroup>

<Label>Nombre de usuario</ Etiqueta>

<Nombre de entrada ="Nombre de usuario" />

<Label>Contraseña</ Etiqueta>

<Nombre de entrada ="contraseña" />

</ FormGroup>

<Botón>Enviar</ Botón>

</ Form>



43

Composición

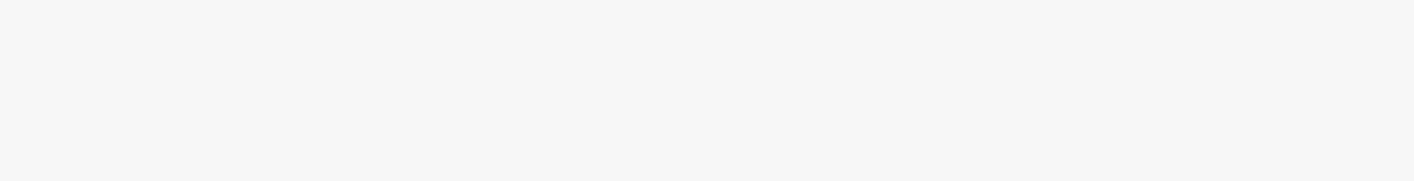


44

Tarea 2



**Tarea 2**

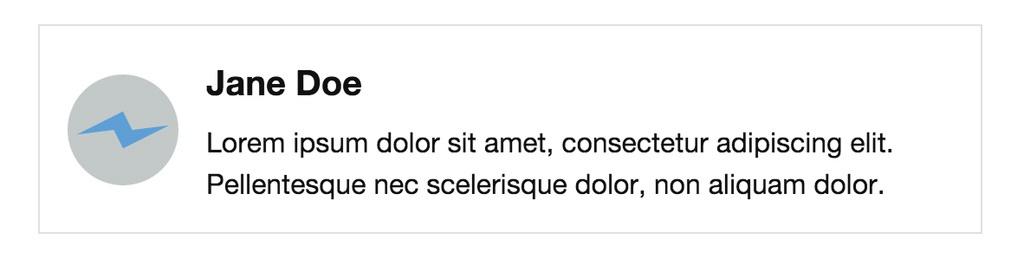


git checkout 02-componente

jspm instalar

NPM plazo dev

**Crear un componente que hace que un objeto de medios**



Debe estar formado por componentes separados:



<ProfileImage />



</ Tarjeta>



Contenido para el <Card> </ Tarjeta> debe ser proporcionado como niños



45

Tarea # 3



**Tarea # 3**

**Construir un componente que hace una lista de perfiles**

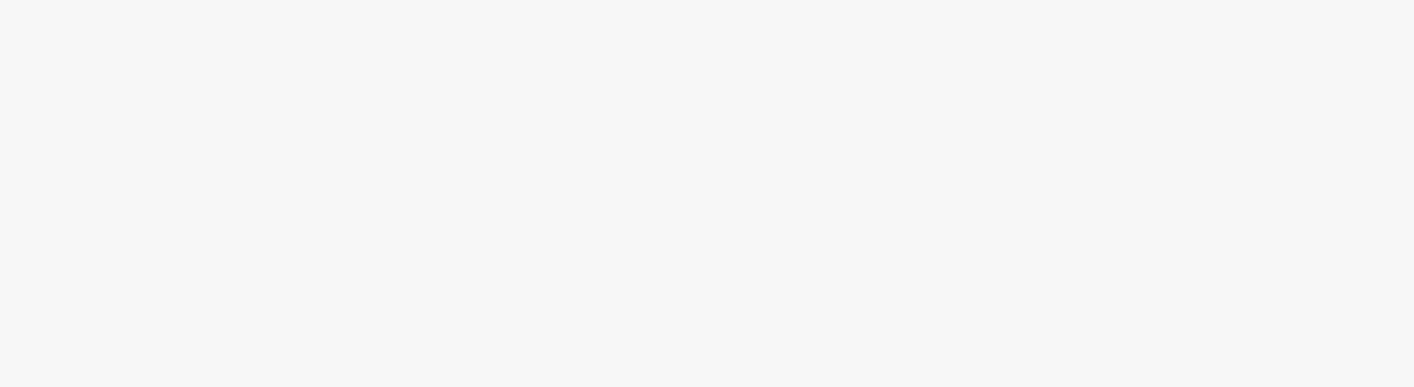
Utilice su creado previamente </ Tarjeta> componente



Crear un nuevo stateful <CardList /> componente



Almacenar el conjunto de perfiles en el estado de



[

{

carné de identidad: 1,

nombre: 'Jane React',

descripción: 'Persona viva más fresco'

},

...

]

**tarea bono**

Añadir un contador que indica la cantidad de personas en la lista



46

Tarea # 4



**Tarea # 4**

**Agregue un botón de borrar para cada perfil**

El botón de borrar debe eliminar un elemento de la matriz



Por favor use Array (). Filtro



**tarea bono**

Agregar una casilla de verificación para cada elemento de la lista, permitir que un usuario borrar todos los elementos seleccionados.



47

Tarea # 5



**Tarea # 5**

**Añadir la posibilidad de añadir un perfil de la lista**

Requerirá mover el estado de <ProfileList />



Crear un nuevo componente contenedor llamado <App /> que almacena el estado No se olvide de los componentes de degradación!



Etiqueta , Entrada , Formar , Formulario de perfil



**tarea bono**

Añadir capacidad de Inline-editar cada elemento.



48

Inmutable



**Immutable.js**

[Immutable.js](https://facebook.github.io/immutable-js/) es una biblioteca que ofrece colecciones genéricas inmutables.



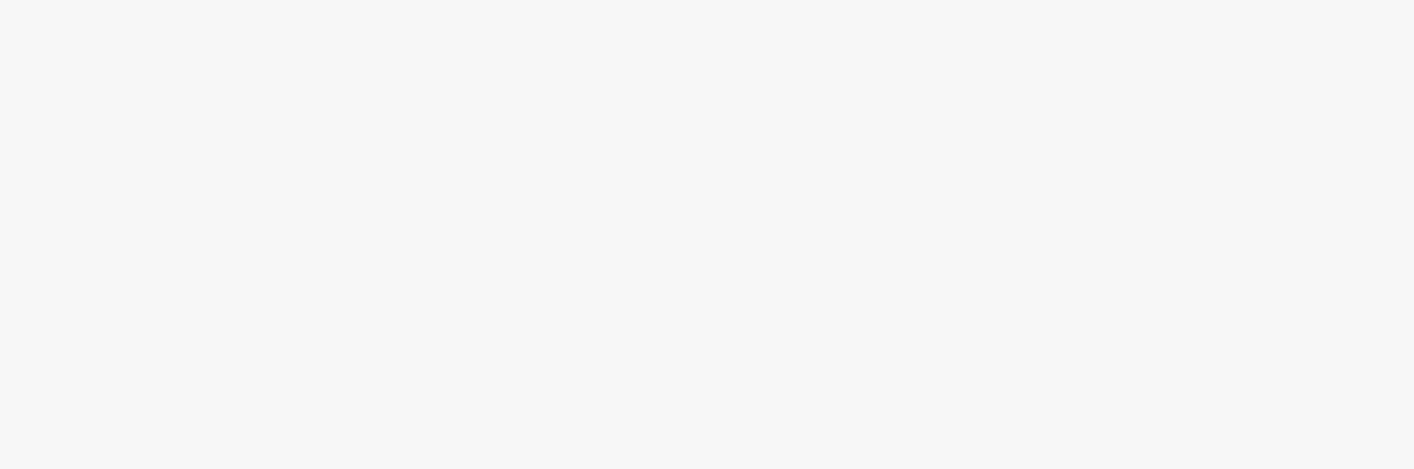
49

¿Qué es la inmutabilidad?



**¿Qué es la inmutabilidad?**

**Inmutabilidad** es un patrón de diseño en el que algo no se puede modificar después de ser instanciado. Si queremos cambiar el valor de esa cosa, hay que volver a crearlo con el nuevo valor en su lugar. Algunos tipos de JavaScript son inmutables y algunos son mutables, es decir, su valor puede cambiar sin tener que volver a crearlo. Vamos a explicar esta diferencia con algunos ejemplos:



dejar película = {

nombre: 'Guerra de las Galaxias',

episodio: 7

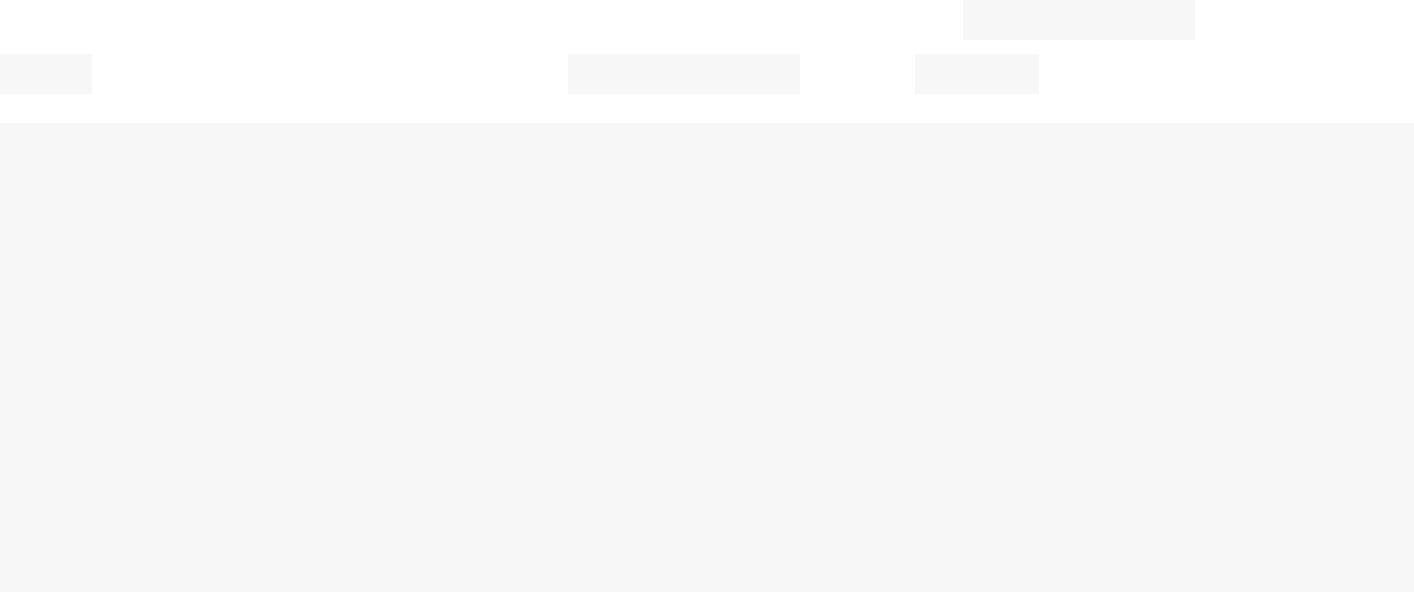
};

dejar myEp = movie.episode;

movie.episode = 8;

consola.log (myEp); // salidas 7

Como se puede ver, en este caso, aunque hemos cambiado el valor de movie.episode , El valor de myEp no cambió. Eso es porque movie.episode 'Tipo s, número , Es inmutable.



dejar movie1 = {

nombre: 'Guerra de las Galaxias',

episodio: 7

};

dejar movie2 = movie1;

movie2.episode = 8;

consola.log (movie1.episode); // salidas 8

En este caso, sin embargo, cambiando el valor del episodio en un objeto también cambió el valor de la otra. Eso es porquemovie1 y movie2 son del tipo de objeto y objetos son mutables.



De los tipos incorporados de JavaScript, éstos son inmutables:

Boole



Número



Cuerda



Símbolo



Nulo



50

¿Qué es la inmutabilidad?



Indefinido



Estos son mutables:

Objeto



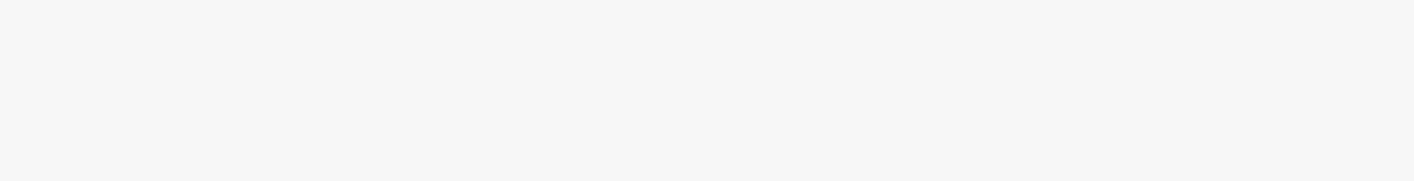
Formación



Función



Cadena es un caso inusual, ya que puede repetirse durante el uso de ... de indexadores y proporciona numéricos como una matriz, pero hacer algo como:



dejar message = 'Hola Mundo';

mensaje[5] = '-';

consola.log (mensaje); // escribe Hola mundo

**Nota:** Esto generará un error en modo estricto y error de forma silenciosa en modo no estricto.



51

El caso de la inmutabilidad



**El caso de la inmutabilidad**

Una de las cosas más difíciles de manejar en la estructuración de una aplicación es la gestión de su estado. Esto es especialmente cierto cuando la aplicación se puede ejecutar código de forma asíncrona. Digamos que se ejecuta alguna pieza de código, pero algo hace que se espere (como una solicitud HTTP o la entrada del usuario). Después de su completado, se da cuenta del estado de su expecting cambiado debido a alguna otra pieza de código que se ejecuta de forma asíncrona y cambió su valor.

Se trata de ese tipo de comportamiento a pequeña escala podría ser manejable, pero esto puede aparecer por todas partes una aplicación y puede ser un verdadero dolor de cabeza ya que la aplicación se hace más grande con más interacciones y una lógica más compleja.

Inmutabilidad intenta resolver esto asegurándose de que cualquier objeto que ha sido referenciada en una parte del código no todos de un repentino puede ser cambiado por otro parte del código a menos que tengan la capacidad de volver a enlazar directamente.



52

Soluciones de JavaScript



**soluciones de JavaScript**

Algunas de las nuevas características se han añadido en la ES6 que permiten una fácil implementación de patrones de datos inmutables.



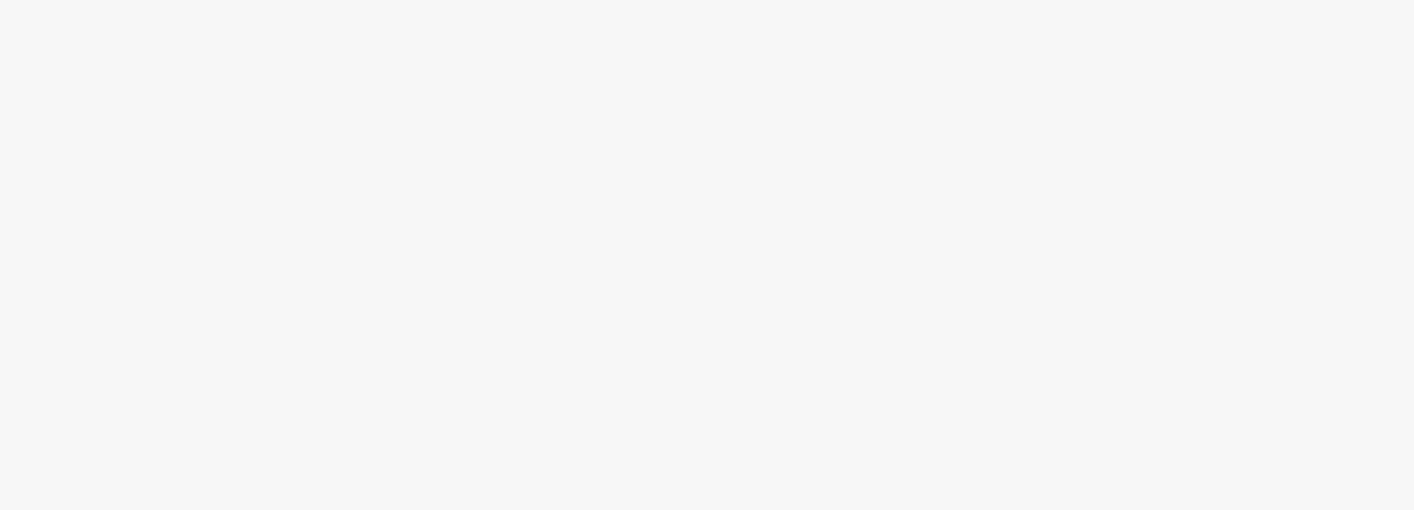
53

Object.assign



**Object.assign**

Object.assign nos permite fusionar las propiedades de un objeto en otro uno, reemplazando los valores de propiedades con nombres que coinciden. Podemos usar esto para copiar los valores de un objeto sin alterar la existente.



dejar movie1 = {

nombre: 'Guerra de las Galaxias',

episodio: 7

};

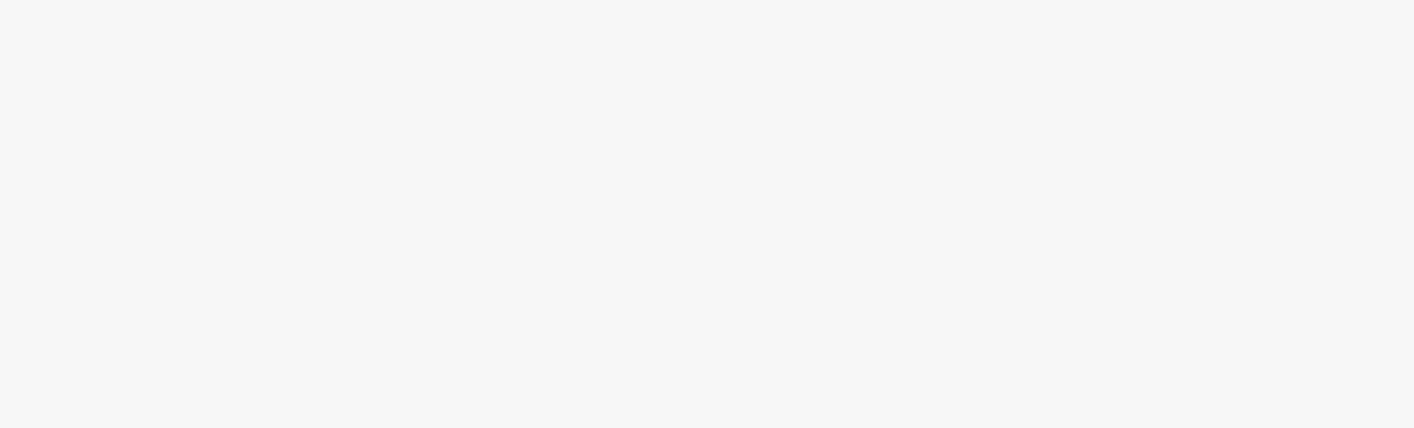
dejar movie2 = Objeto.assign ({}, movie1);

movie2.episode = 8;

consola.log (movie1.episode); // escribe 7

consola.log (movie2.episode); // escribe 8

Como se puede ver, aunque tenemos alguna manera de copiar un objeto, no hemos hecho inmutable, ya que hemos sido capaces de establecer la propiedad del episodio a 8. Además, ¿cómo podemos modificar la propiedad episodio en este caso? Lo hacemos a través de la llamada de asignación:



dejar movie1 = {

nombre: 'Guerra de las Galaxias',

episodio: 7

};

dejar movie2 = Objeto.assign ({}, movie1, {episodio: 8 });

consola.log (movie1.episode); // escribe 7

consola.log (movie2.episode); // escribe 8



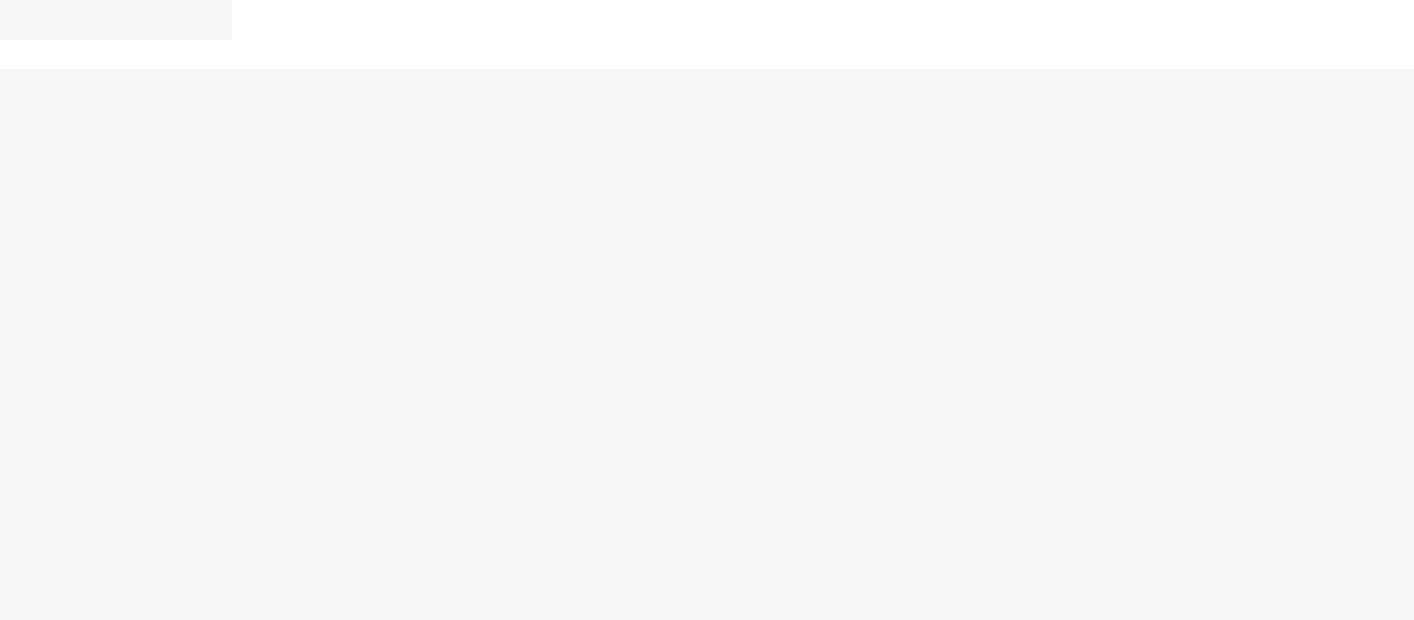
54

Object.freeze



**Object.freeze**

Object.freeze nos permite deshabilitar mutación objeto.



dejar movie1 = {

nombre: 'Guerra de las Galaxias',

episodio: 7

};

dejar movie2 = Objeto.congelar(Objeto.assign ({}, movie1));

movie2.episode = 8; // error de forma silenciosa en modo no estricto, // lanza error en modo estricto

consola.log (movie1.episode); // escribe 7

consola.log (movie2.episode); // escribe 7

Un problema con este patrón, sin embargo, es cuánto más prolija nuestro código es y lo difícil que es leer y entender lo que realmente está pasando con nuestros datos con todas las llamadas a la plancha de caldera Object.freeze y Object.assign. Necesitamos alguna de las interfaces más sensato para crear e interactuar con los datos inmutables, y ahí es donde encaja Immutable.js.



**Nota:** Object.freezeTambién es muy lento y no debe ser hecho con grandes matrices.



55

Fundamentos Immutable.js



**fundamentos Immutable.js**

Para resolver nuestro problema de la mutabilidad, Immutable.js necesita proporcionar versiones inmutables de los dos principales tipos mutables, objetos y Array.



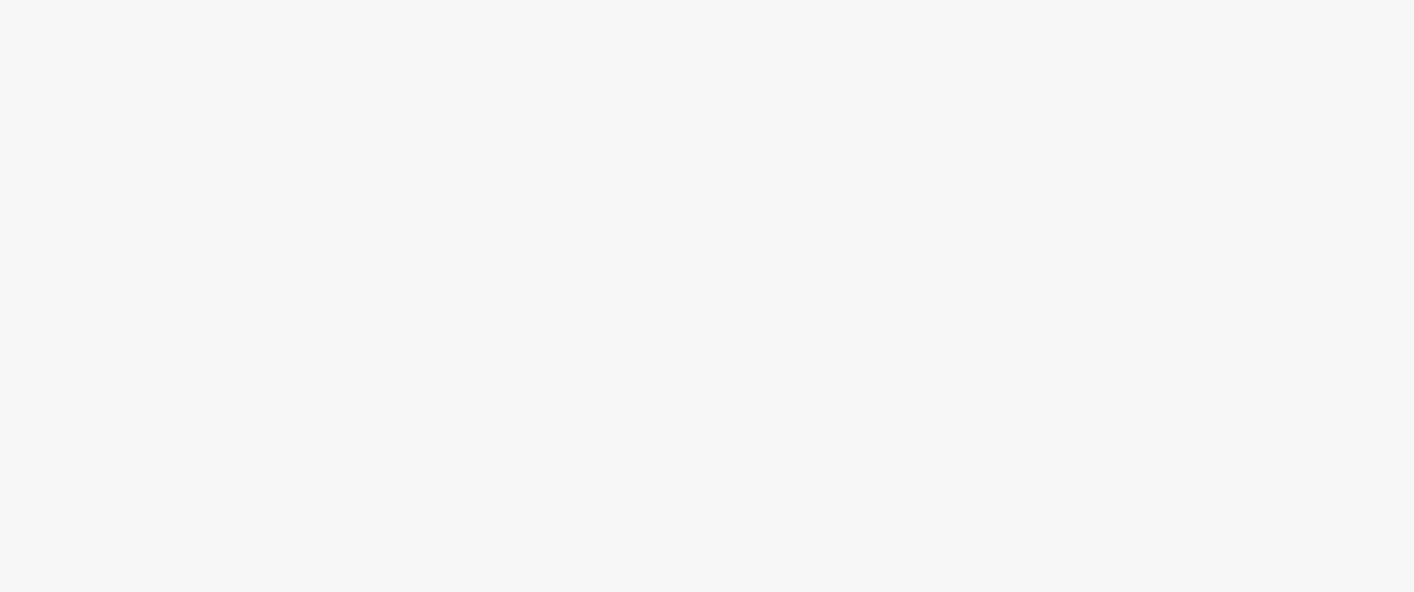
56

Immutable.Map



**Immutable.Map**

Mapa es la versión inmutable de la estructura de objetos de JavaScript. Debido a objetos JavaScript teniendo la sintaxis literal de objeto concisa, a menudo se utiliza como almacén de claves-valor con llave tipo de bienestar cuerda. Este patrón sigue la estructura de datos del mapa. Vamos a revisar el ejemplo anterior, pero el usoImmutable.Map en lugar.



importar \* como Inmutable de 'inmutable';

dejar movie1 = Immutable.Map ({

nombre: 'Guerra de las Galaxias',

episodio: 7

});

dejar movie2 = movie1;

movie2 = movie2.set ('episodio', 8);

consola.log (movie1.get ('episodio')); // escribe 7 consola.log (movie2.get ('episodio')); // escribe 8

En lugar de unir el objeto literal directamente a movie1 , Se pasa como argumento a Immutable.Map . Esto cambia la forma en que interactúan con las propiedades del Película1.



Para obtener el valor de una propiedad, que llamamos el obtener método, pasando el nombre de la propiedad que queremos, como la forma en que usaríamos indexador cadena de un objeto.



Para establecer el valor de una propiedad, que llamamos el conjuntométodo, pasando el nombre de la propiedad y el nuevo valor. Tenga en cuenta que no va a mutar el objeto de mapa existente. Devuelve un nuevo objeto con la propiedad actualizado así que tenemos que volver a vincular elmovie2 variable para el nuevo objeto.



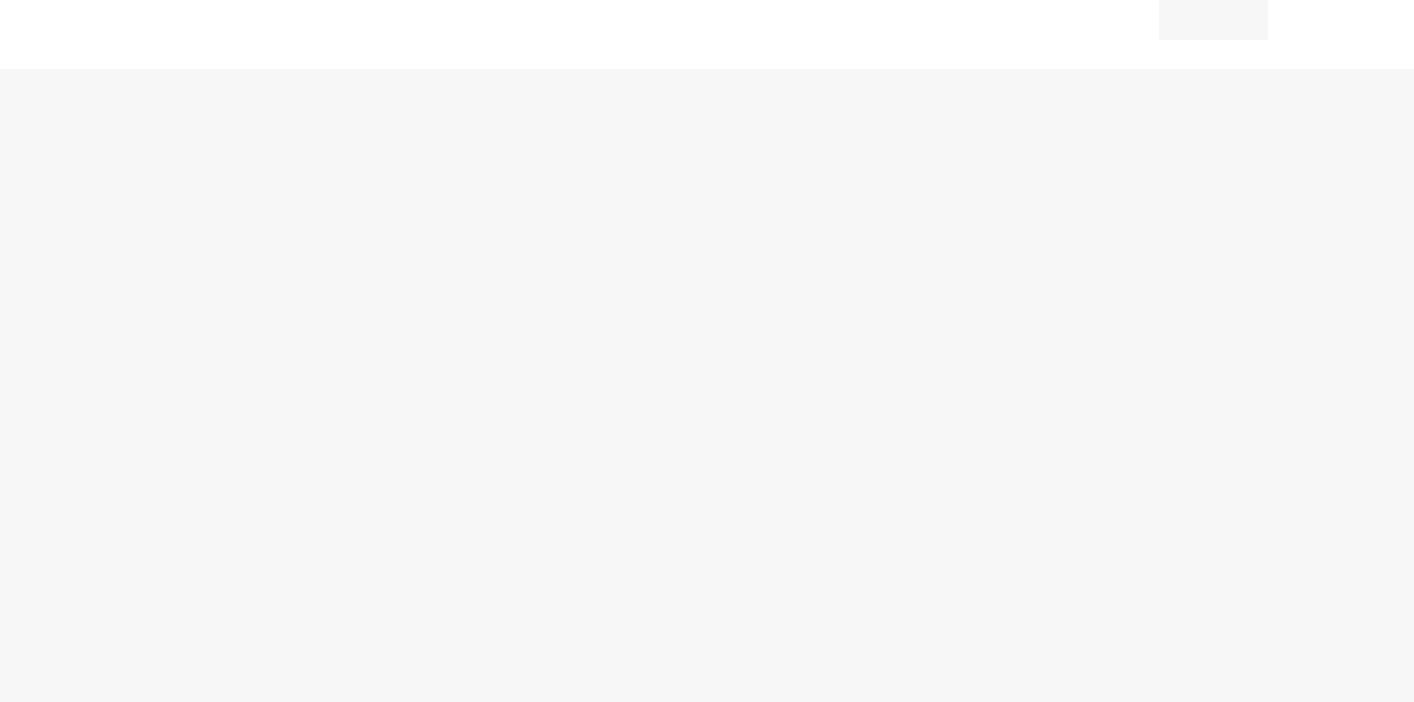
57

Immutable.Map



**Map.merge**

A veces queremos actualizar múltiples propiedades. Podemos hacer esto mediante el unir método.



dejar BaseButton = Immutable.Map ({

texto: '¡Haz click en mi!',

estado: 'inactivo',

anchura: 200,

altura: 30

});

dejar submitButton = baseButton.merge ({

texto: 'Enviar',

estado: 'activo'

});

consola.log (submitButton);

// escribe {text: 'Enviar', Estado: 'activo', anchura: 200, altura: 30}



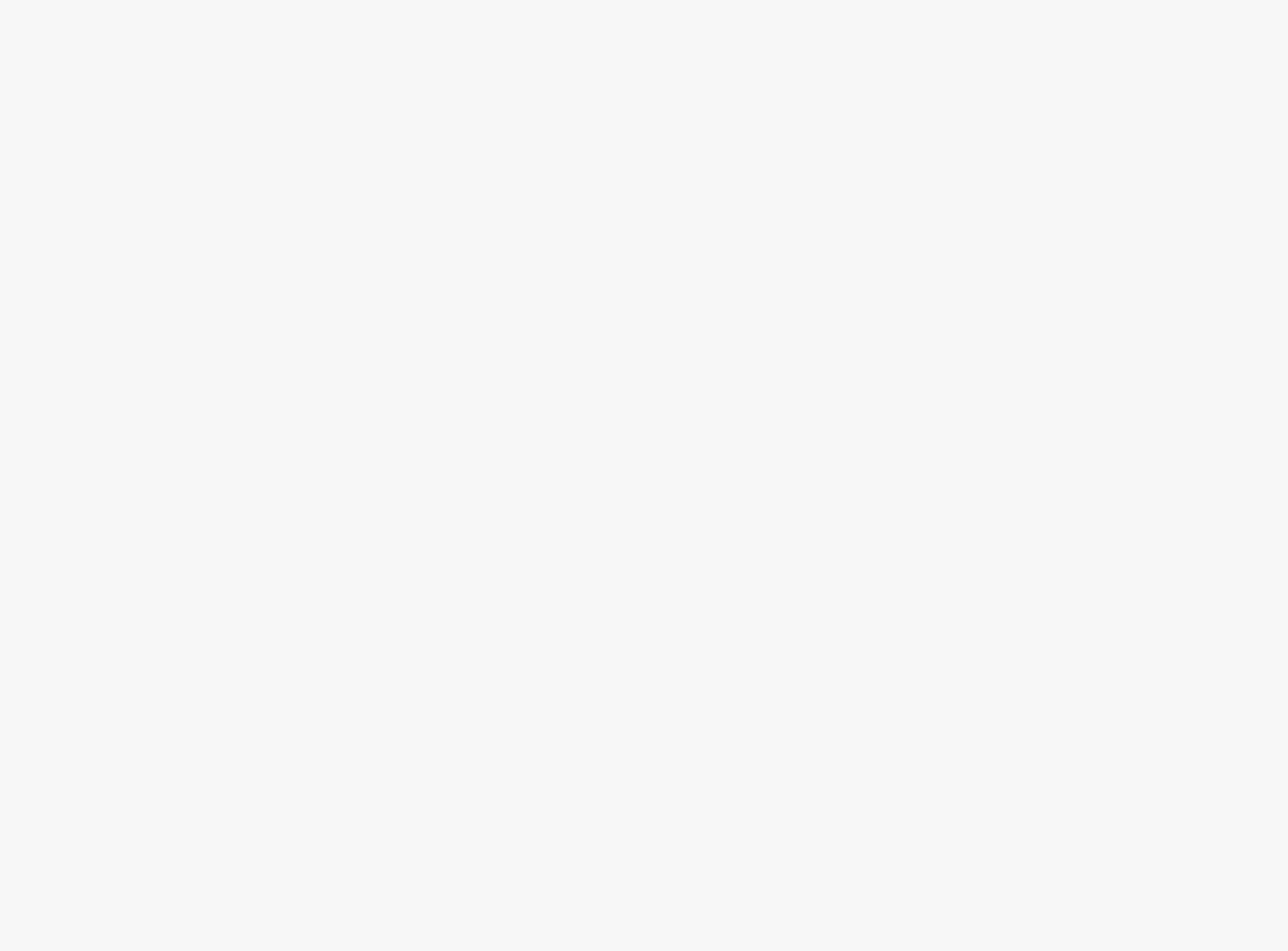
58

Los objetos anidados



**Los objetos anidados**

Immutable.Map envolturas objetos superficialmente, es decir, si tiene un objeto con propiedades unido a tipos mutables entonces esas propiedades pueden ser mutados.



dejar película Immutable.Map = ({

nombre: 'Guerra de las Galaxias',

episodio: 7,

actores: [

{ nombre: 'Daisy Ridley', personaje: 'Rey'},

{ nombre: 'Harrison Ford', personaje: 'Han Solo' }

],

MPAA: {

clasificación: 'PG-13',

razón: 'Acción de la violencia de ciencia ficción'

}

});

movie.get ('actores').popular();

movie.get ('MPAA') .rating = 'PG';

consola.log (movie.toObject ());

/ \* Escribe

{Nombre: Star Wars, episodio: 7,

actores: [{nombre: 'Daisy Ridley', personaje: 'Rey'}],

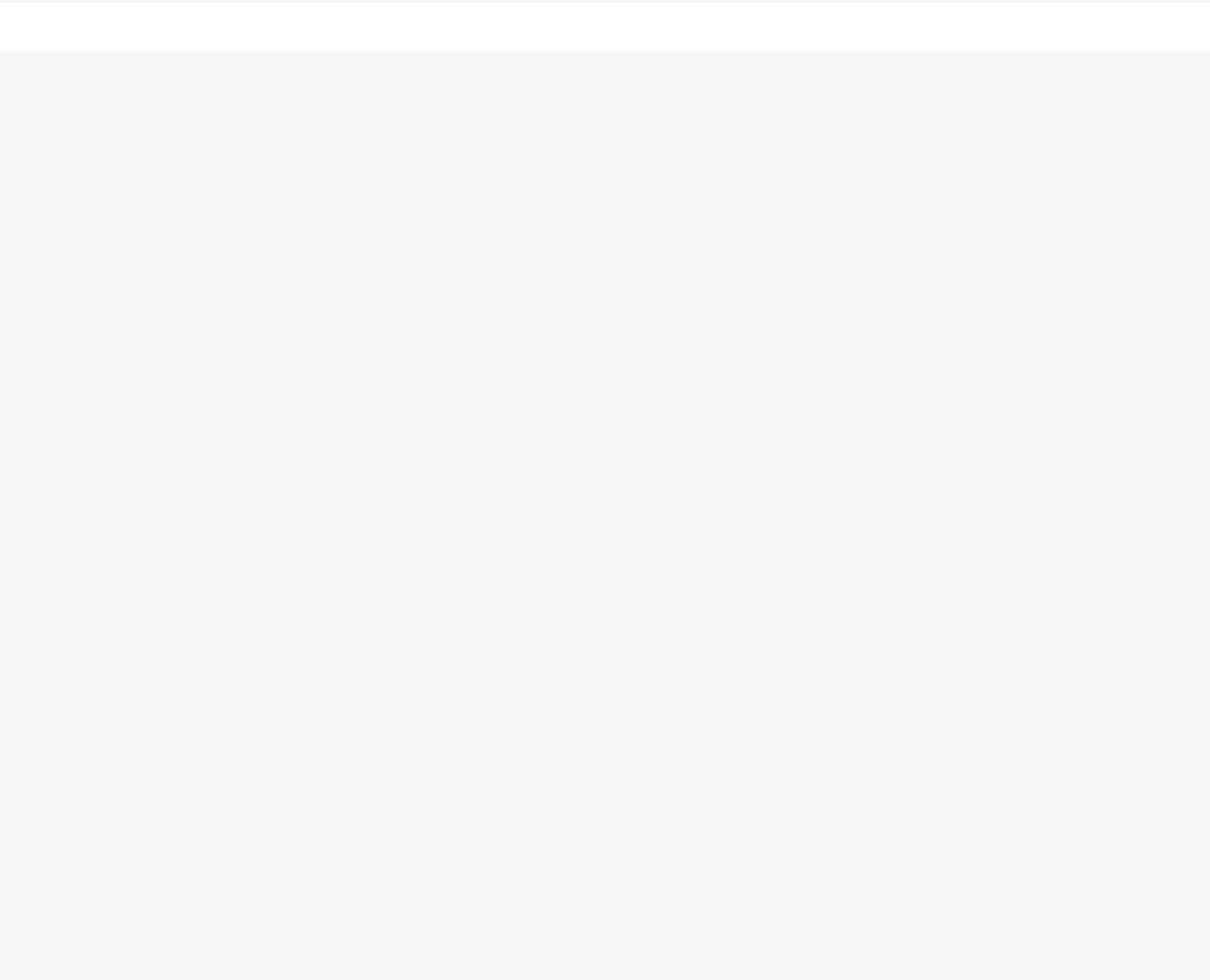
MPAA: {calificación: 'PG', razón: 'ciencia ficción de la violencia de acción'}} \* /

Para evitar este problema, utilice Immutable.fromJS en lugar.



59

Los objetos anidados



dejar cine = {(Immutable.fromJS

nombre: 'Guerra de las Galaxias',

episodio: 7,

actores: [

{ nombre: 'Daisy Ridley', personaje: 'Rey'},

{ nombre: 'Harrison Ford', personaje: 'Han Solo' }

],

MPAA: {

clasificación: 'PG-13',

razón: 'Acción de la violencia de ciencia ficción'

}

});

movie.get ('actores').popular();

movie.get ('MPAA') .rating = 'PG';

consola.log (movie.toObject ());

/ \* Escribe

{Nombre: Star Wars, episodio: 7,

actores: Listado [Mapa { "name": "Margarita Ridley", "carácter": "Rey"}, mapa { "name": "Ha

rrison Ford", "carácter": "Han Solo"}],

MPAA: Mapa { "Nota": "PG-13", "razón": "ciencia ficción de la violencia de acción"}} \* /

Así que digamos que desea modificar movie.mpaa.rating , Se podría pensar en hacer algo así

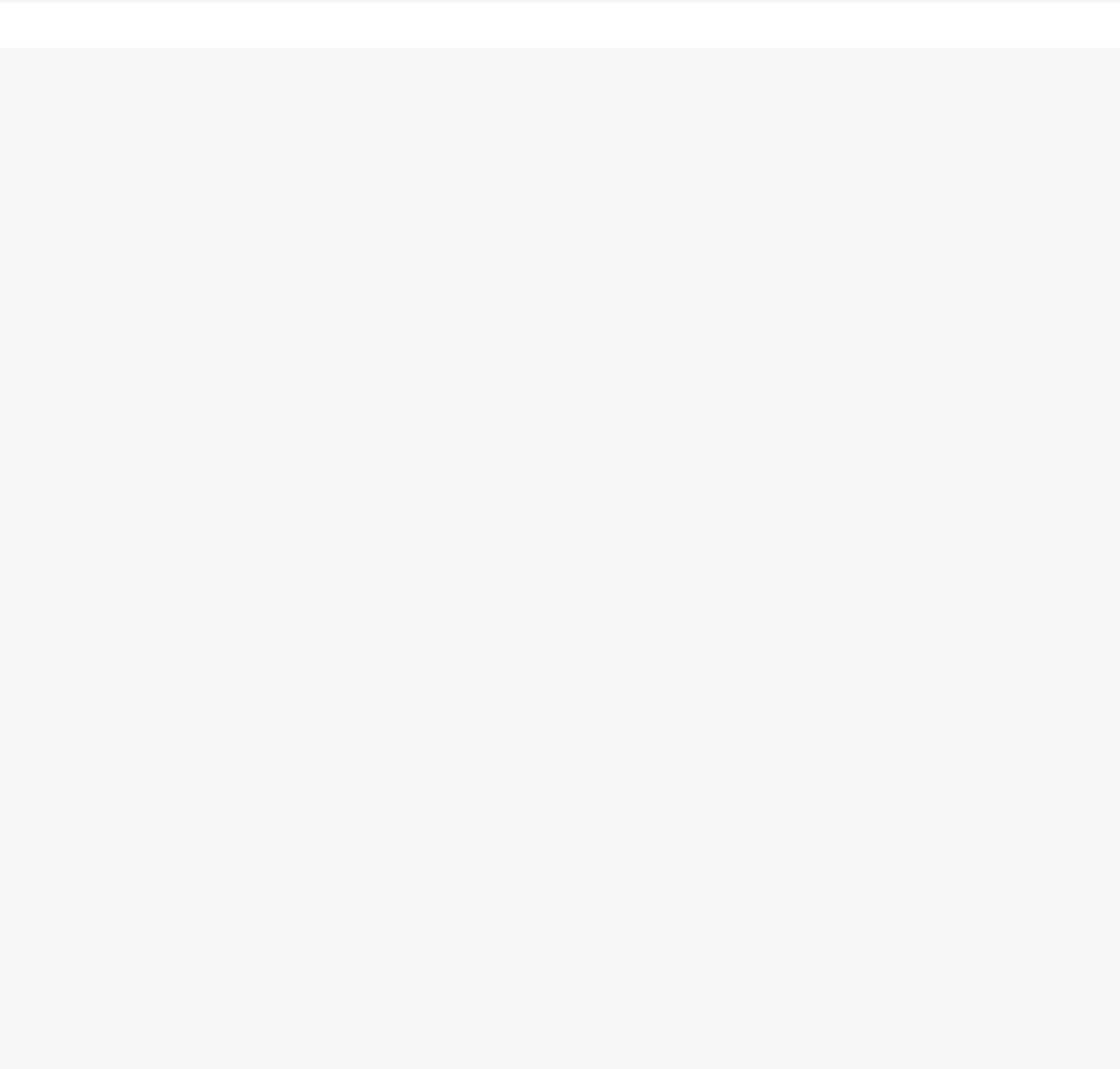


esta: = movie.get película ( 'MPAA'). set ( 'rating', 'PG'). Sin embargo,conjunto siempre devolverá la instancia Mapa llamando que en este caso devuelve el Mapa unido a la MPAAclave en lugar de la película que quería. Tenemos que utilizar laestablecer en método para actualizar las propiedades anidadas.



60

Los objetos anidados



dejar cine = {(Immutable.fromJS

nombre: 'Guerra de las Galaxias',

episodio: 7,

actores: [

{ nombre: 'Daisy Ridley', personaje: 'Rey'},

{ nombre: 'Harrison Ford', personaje: 'Han Solo' }

],

MPAA: {

clasificación: 'PG-13',

razón: 'Acción de la violencia de ciencia ficción'

}

});

= película de cine

.actualizar('actores', Actores => actors.pop ())

.establecer en(['MPAA', 'clasificación'], 'PG')

.actualizar('actores', Actores => actors.push ({

nombre: 'John Boyega',

personaje: 'Finlandés'

}));

consola.log (movie.toObject ());

/ \* Escribe

{Nombre: Star Wars, episodio: 7,

actores: Listado [Mapa { "name": "Margarita Ridley", "carácter": "Rey"}, mapa { "name": "J

ohn Boyega", "carácter": "Finn"}],

MPAA: Mapa { "Nota": "PG", "razón": "ciencia ficción de la violencia de acción"}} \* /

También hemos añadido en una llamada a map.updateque, a diferencia conjunto, acepta una función como el segundo argumento en lugar de un valor. Esta función acepta el valor existente en esa clave y debe devolver el nuevo valor de esa clave.



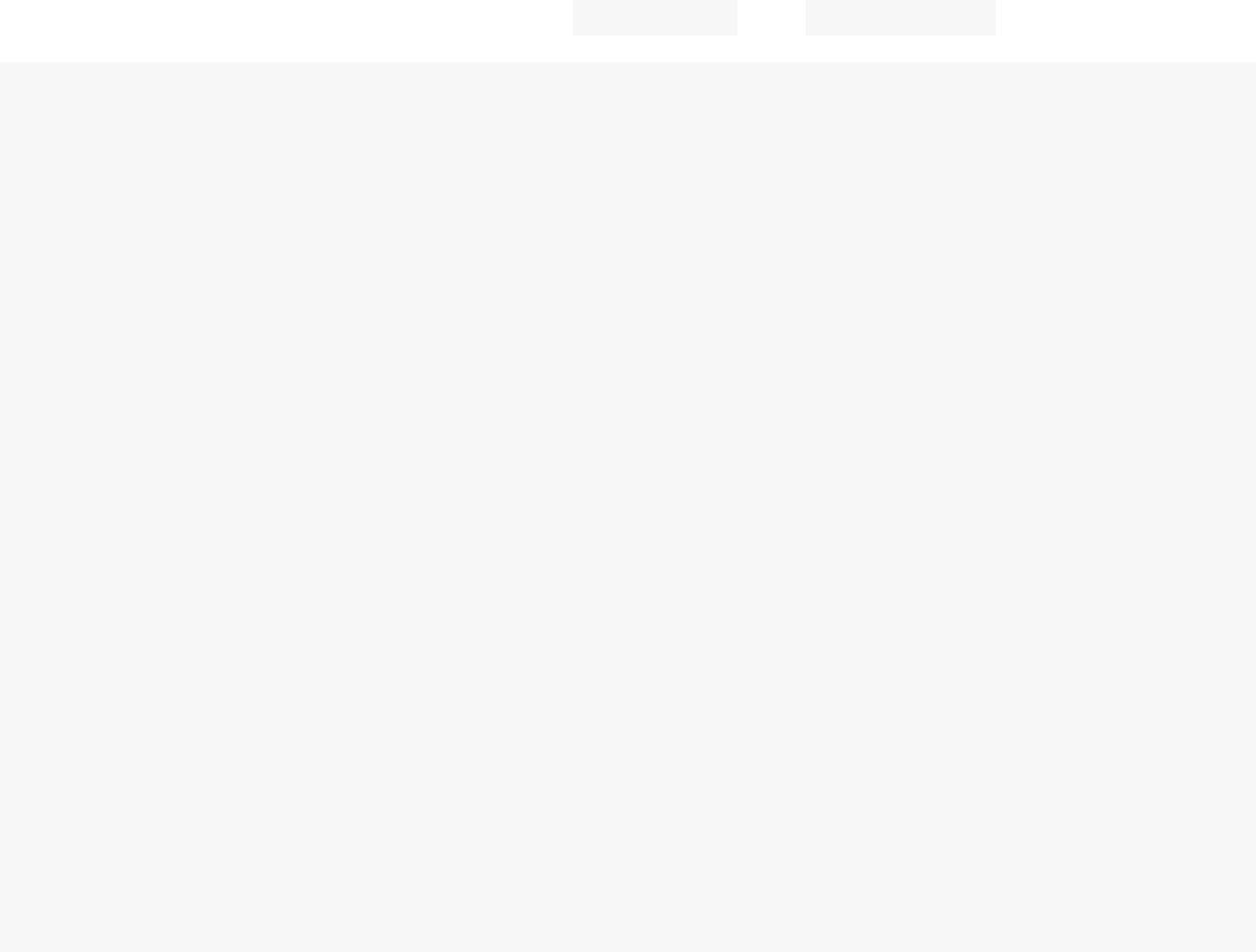
61

Los objetos anidados



**eliminación de claves**

Las llaves se pueden eliminar de mapas utilizando el Map.delete y Map.deleteIn métodos.



dejar cine = {(Immutable.fromJS

nombre: 'Guerra de las Galaxias',

episodio: 7,

actores: [

{ nombre: 'Daisy Ridley', personaje: 'Rey'},

{ nombre: 'Harrison Ford', personaje: 'Han Solo' }

],

MPAA: {

clasificación: 'PG-13',

razón: 'Acción de la violencia de ciencia ficción'

}

});

película movie.delete = ('MPAA');

consola.log (movie.toObject ());

/ \* Escribe

{Nombre: 'Star Wars',

episodio: 7,

actores: Listado [Mapa { "name": "Margarita Ridley", "carácter": "Rey"}, mapa { "name": "Ha

rrison Ford", "carácter": "Han Solo"}]}

\* /



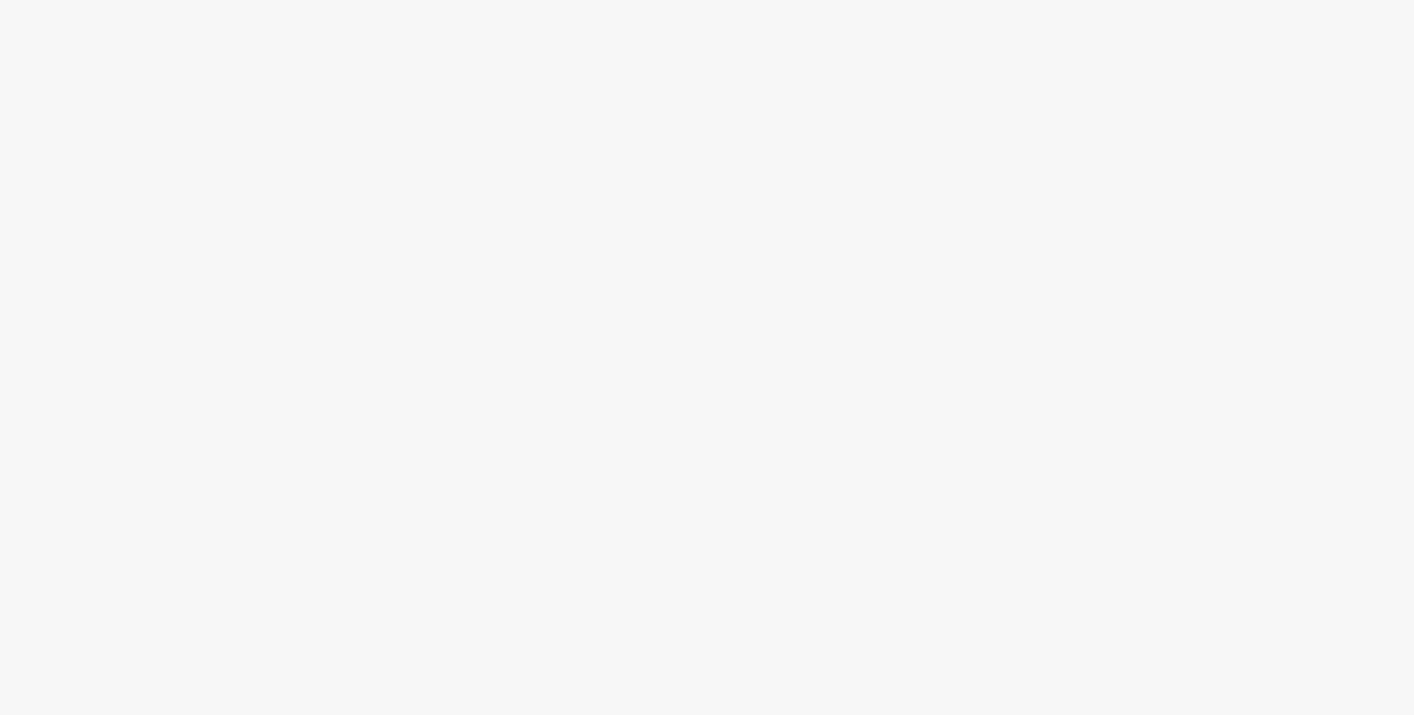
62

Los objetos anidados



**Los mapas son iterables**

Mapas en Immutable.js son iterable, lo que significa que se puede mapa , filtrar , reducir , Etc. cada par clave-valor en el mapa.



dejar características = Immutable.Map ({

'Enviar' enlaces: cierto,

'enviar archivos': cierto,

'almacenamiento local': cierto,

-espejo notificaciones ': falso,

'Api acceso': falso

});

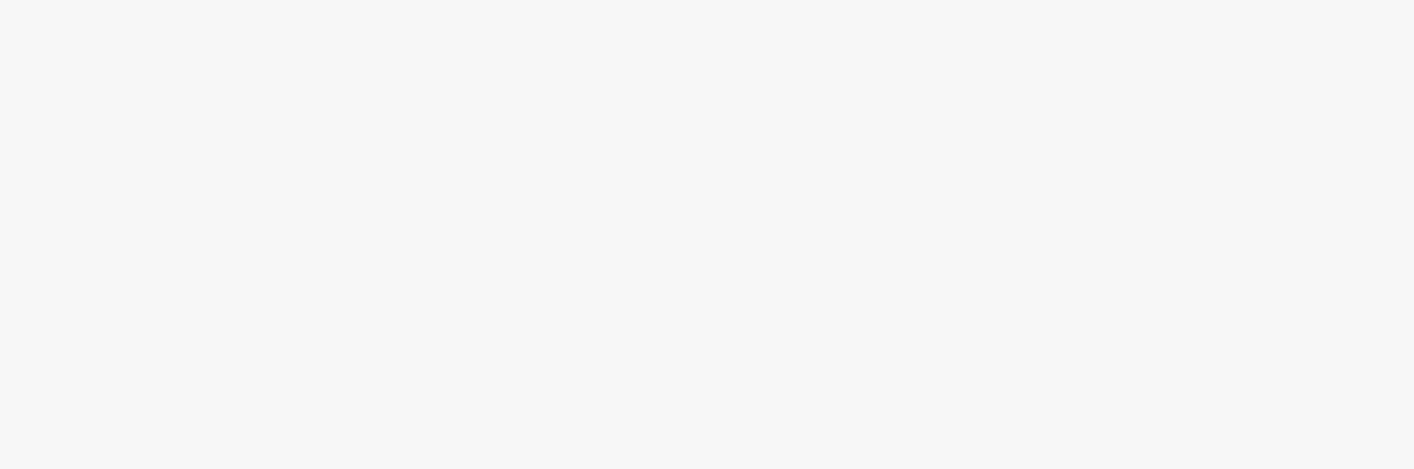
dejar myFeatures = features.reduce ((providedFeatures, proporcionado, característica) => { Si(previsto)

providedFeatures.push (función);

regreso providedFeatures;

}, []);

consola.log (myFeatures); // [ 'enlaces Enviar', 'archivos Enviar', 'local de almacenamiento']



const MapMap = Immutable.Map ({a: 0, B: 1, C: 2 }); mapMap.map (i => i \* 30);

const mapFilter = Immutable.Map ({a: 0, B: 1, C: 2 });

mapFilter.filter (i => i% 2);

const mapreduce = Immutable.Map ({a: 10, B: 20, C: 30 });

mapReduce.reduce ((acc, i) => acc + i, 0);



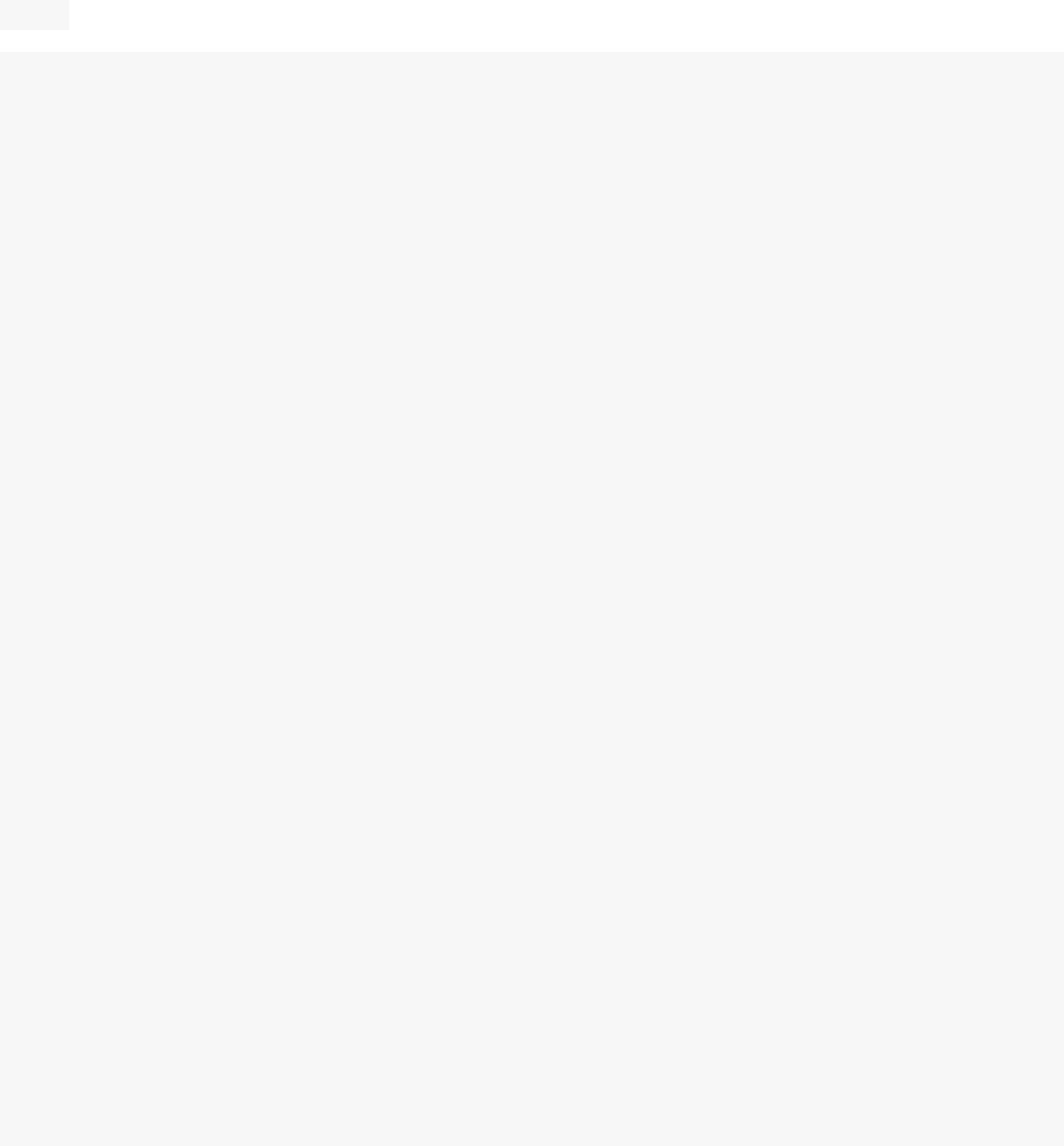
63

Immutable.List



**Immutable.List**

Lista es la versión inmutable de la estructura de matriz de JavaScript.



dejar películas = Immutable.fromJS ([ // utilizar de nuevo para fromJS profunda inmutabilidad

{

nombre: 'La comunidad del anillo',

liberado: 2001,

clasificación: 8.8

},

{

nombre: 'Las dos torres',

liberado: 2002,

clasificación: 8.7

}

]);

películas movies.push = (({Immutable.Map

nombre: 'El retorno del Rey',

liberado: 2003

}));

películas (= movies.update2, Película => movie.set ('clasificación', 8.9)); // 0 basada

películas movies.zipWith = (

(Película, seriesNumber) => movie.set ('episodio', SeriesNumber), Immutable.Range (1, Movies.size + 1) // propiedad de tamaño en lugar de longitud

);

consola.log (películas);

/ \* Escribe

lista [

Mapa { "name": "La comunidad del anillo", "liberado": 2001, "Nota": 8.8 ", episod

E ": 1},

Mapa { "name": "Las dos torres", "liberados": 2002, "Nota": 8,7, "episodio": 2},

Mapa { "name": "El retorno del rey", "liberado": 2003, "Nota": 8,9, "episodio":

3}]

\* /

Aquí se utiliza la Immutable.fromJSllamar de nuevo ya que tenemos objetos almacenados en la matriz. Llamamosempujarañadir elementos a la lista, al igual que lo llamaríamos en una matriz, pero ya que estamos creando una nueva copia que tenga que volver a enlazar la variable. Tenemos lo mismoconjunto y actualizarllama cuando queremos actualizar los elementos en índices específicos. También tenemos acceso a un arreglo de funciones comomapa , reducir con soporte para extras como la que estamos usando aquí, zipWith .



64

Immutable.List



sesenta y cinco

Actuación



**Actuación**

Debido a tener que asignar memoria y tener que copiar la estructura de datos cada vez que se realice una modificación, esto puede conducir potencialmente a un gran número de operaciones adicionales que tienen que realizarse en función de qué tipo de cambios se hacen y cómo muchos de ellos. Para demostrar la diferencia, aquí hay una[prueba de funcionamiento](https://jsperf.com/immutable-js-data-structure-perf1/4). Haciendo asignación de memoria y copia en grandes cadenas puede ser caro, incluso en un objeto de poca profundidad.



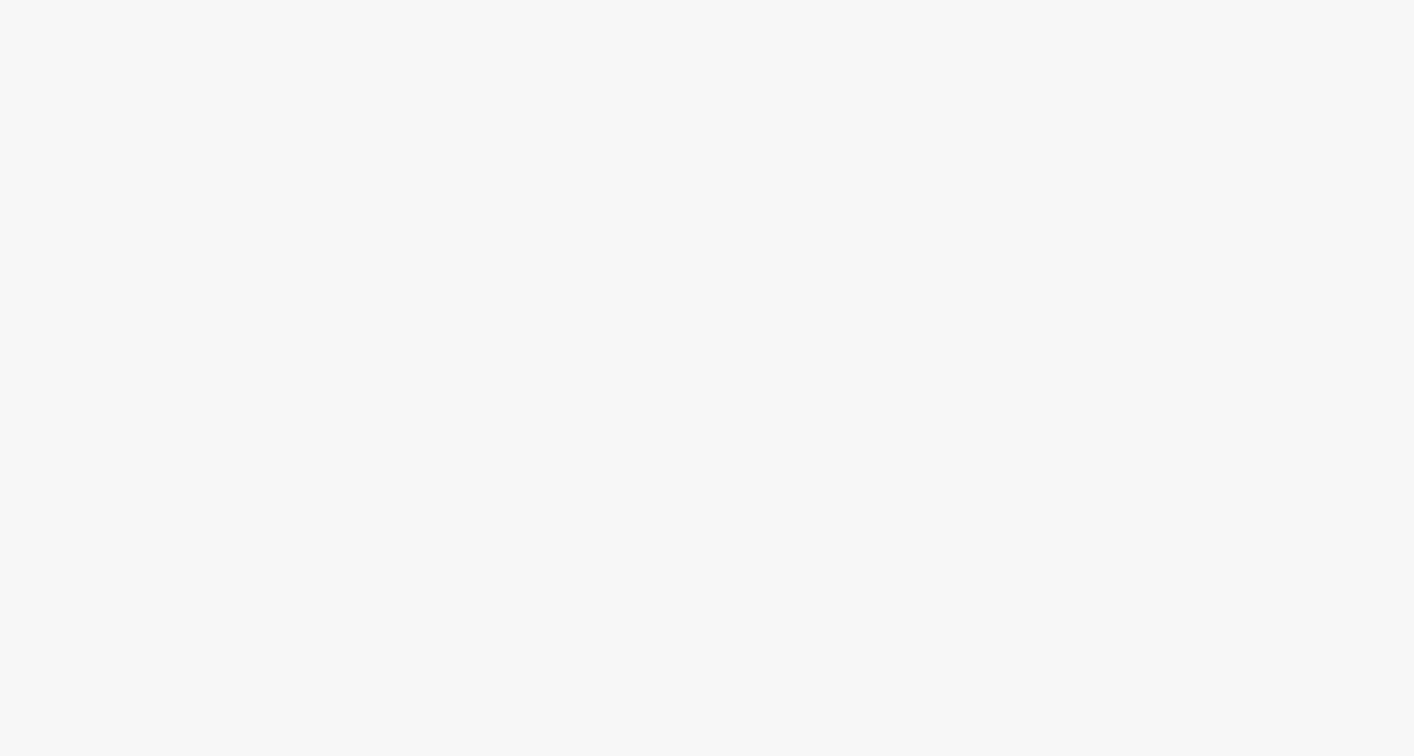
66

Estructuras de datos persistentes y transitorias



**estructuras de datos persistentes y transitorias**

Immutable data structures are also sometimes referred to as persistent data structures, since its values persist for its lifetime. Immutable.js provides the option for transient data structures, which is a mutable version of a persistent data structure during a transformation stage and returning a new immutable version upon completion. This is one approach to solving the performance issues we encountered earlier. Let's revisit the immutable case outlined in the performance example, but using a transient data structure this time:



import \* as Immutable from 'immutable';

let list = Immutable.List();

list = list.withMutations(mutableList => {

let val = "";

return Immutable.Range(0, 1000000)

.forEach(() => {

val += "concatenation";

mutableList.push(val);

});

});

console.log(list.size); // writes 1000000

list.push('');

console.log(list.size); // writes 1000000

As we can see in [this performance test](http://jsperf.com/immutable-js-data-structure-perf2/2), the transient list builder was still a lot slower than the fully mutable version, but much faster than the fully immutable version. Also, if you pass the mutable array to Immutable.List or Immutable.fromJS , you'll find the transient version closes the performance gap. The test also shows how slow Object.freeze can be compared to the other 3.



67

Official Documentation



**Official documentation**

For more information on Immutable.js, visit the official documentation at

<https://facebook.github.io/immutable-js/>.



68

Exercises



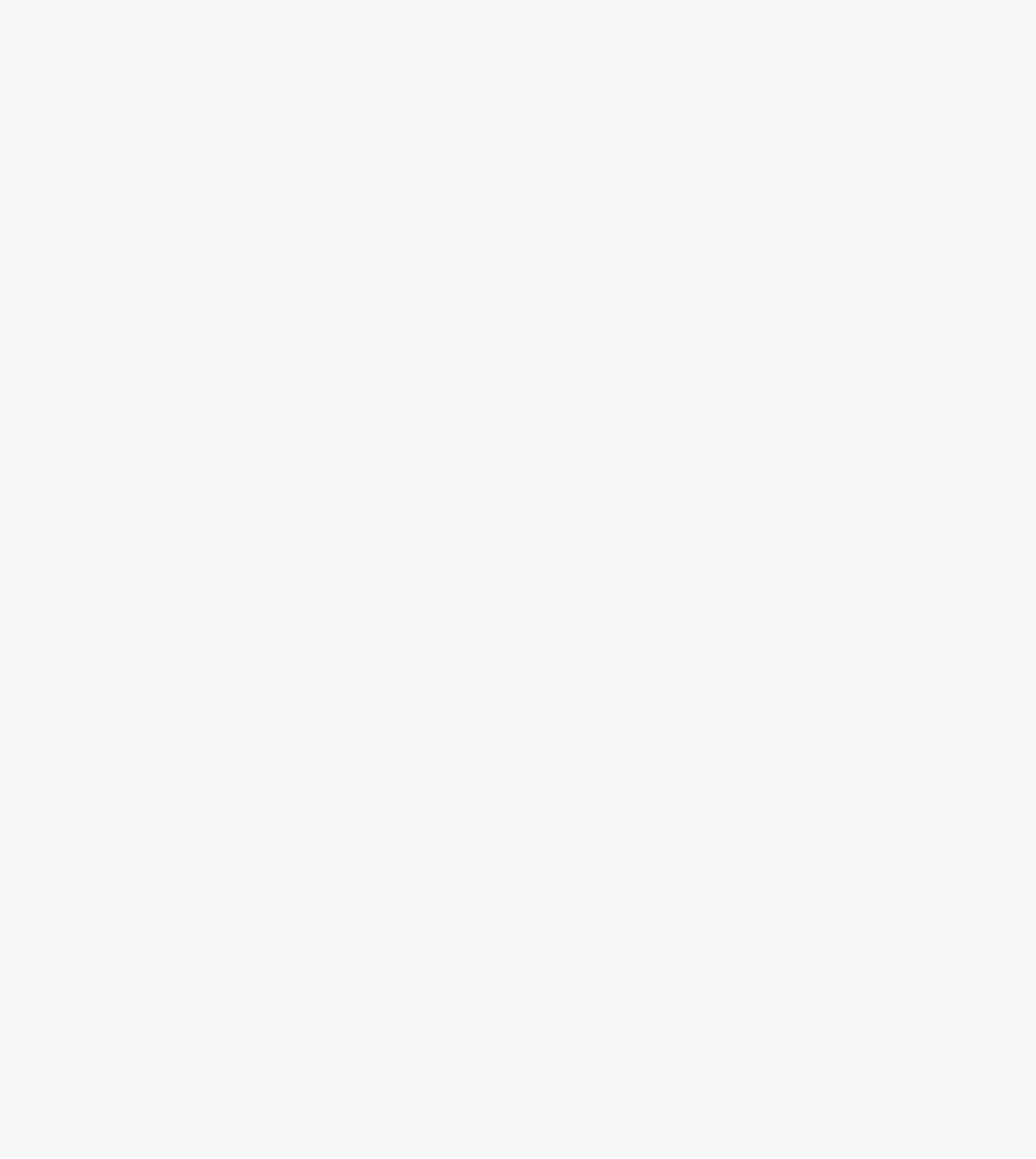
**Let's Try It Out**

**Immutable Repl**

[neilff.github.io/immutable-repl](http://neilff.github.io/immutable-repl)

**Dataset**

For all the tasks in this section use this as the initial dataset.



[

{

"\_id": "56e18ce608c0a0190da963f8",

"index": 0,

"guid": "5e0dbf88-33f1-4b84-bdca-ac21719bf0e8",

"isActive": false,

"balance": "$1,284.82",

"picture": "http://placehold.it/32x32",

"age": 36,

"eyeColor": "blue",

"name": {

"first": "Lauren",

"last": "Stanley"

},

"company": "HAIRPORT",

"email": "lauren.stanley@hairport.name",

"phone": "+1 (876) 425-2958",

"address": "456 Front Street, Wacissa, Virginia, 9236",

"about": "Dolor aliqua enim irure mollit. Sunt ullamco laborum reprehenderit labor e. Eu consequat laborum consectetur voluptate laborum fugiat quis tempor amet nulla. I rure duis reprehenderit irure officia sit magna deserunt. Incididunt eu aliquip proide nt id amet enim dolor reprehenderit ut ipsum est elit ea.",

"registered": "Friday, August 8, 2014 4:08 PM",

"latitude": "41.628375",

"longitude": "104.950835",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

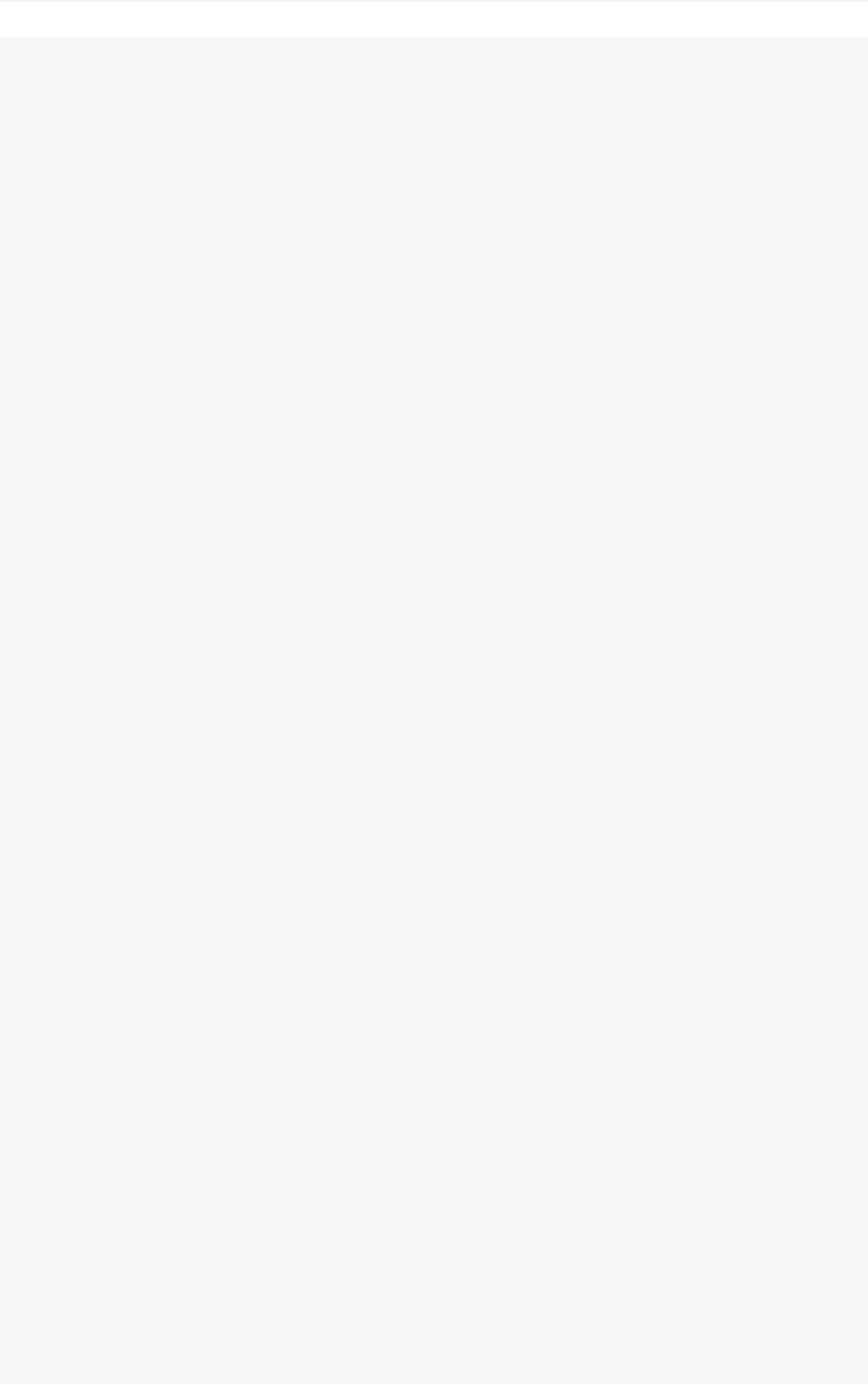
5,

6,



69

Exercises



7,

8,

9

],

"friends": [

3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Lauren! You have 9 unread messages.",

"favoriteFruit": "banana"

},

{

"\_id": "56e18ce6dc7d5ade1e3c7889",

"index": 1,

"guid": "7ceca65c-cc8d-4f88-ab00-b5d00b72e27f",

"isActive": true,

"balance": "$1,423.68",

"picture": "http://placehold.it/32x32",

"age": 35,

"eyeColor": "brown",

"name": {

"first": "Schmidt",

"last": "Floyd"

},

"company": "ANIXANG",

"email": "schmidt.floyd@anixang.org",

"phone": "+1 (913) 595-3119",

"address": "274 Norfolk Street, Freeburn, Nevada, 1869",

"about": "Exercitation deserunt quis commodo ad qui aliqua proident mollit labore mollit. Deserunt occaecat in pariatur mollit aute consequat reprehenderit in deserunt magna ad. Aliquip labore do mollit officia laboris in aliquip magna aliqua. Sunt occae cat eiusmod ea amet dolore consectetur aute consequat adipisicing et nisi fugiat. Aute eiusmod quis duis ipsum occaecat culpa eiusmod Lorem amet laborum occaecat adipisicin g minim. Labore exercitation laborum sint enim veniam labore officia. Aliquip do esse consectetur amet.",

"registered": "Sunday, October 12, 2014 8:17 AM",

"latitude": "-3.271053",

"longitude": "-124.321634",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

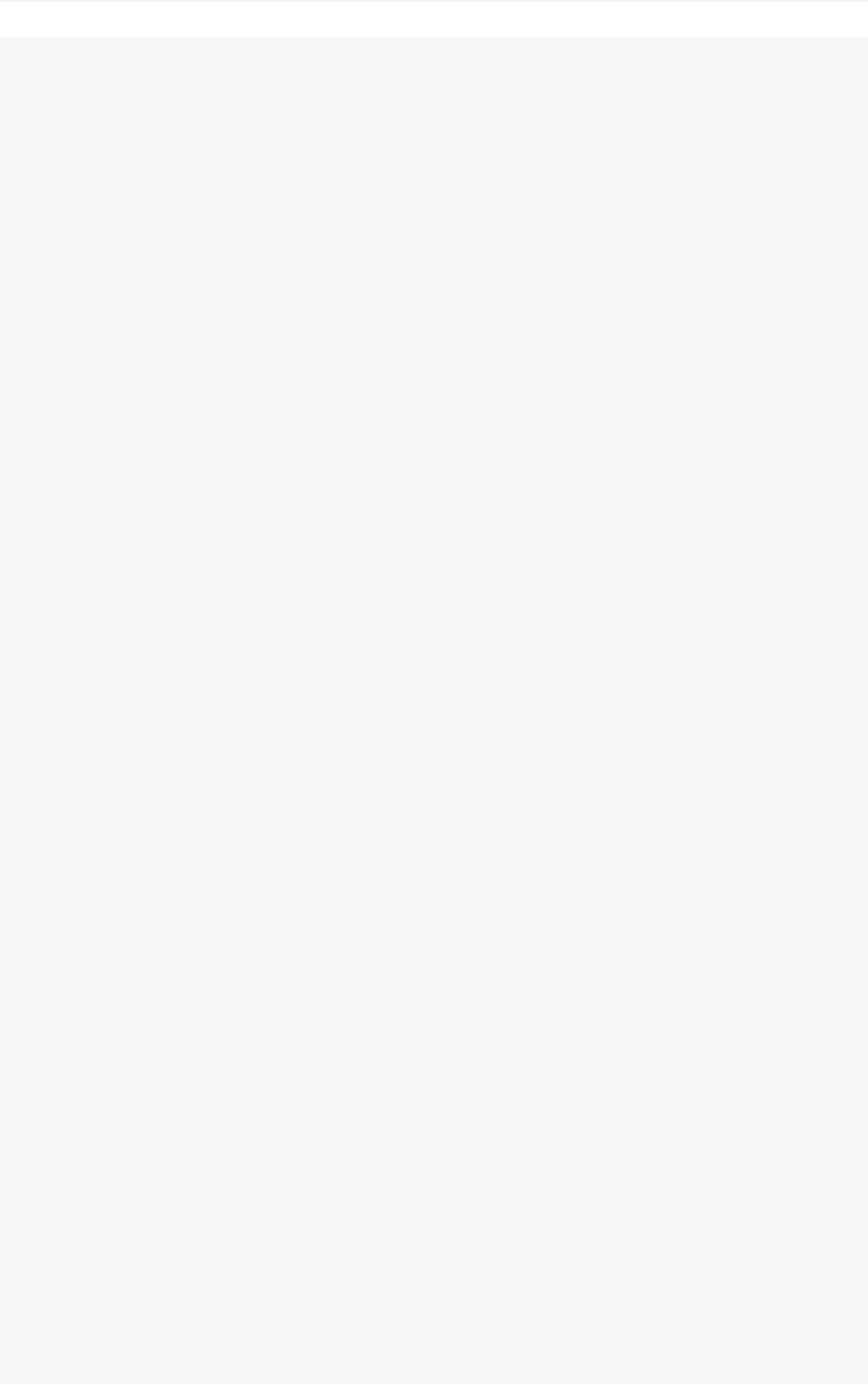
5,

6,



70

Exercises



7,

8,

9

],

"friends": [

3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Schmidt! You have 9 unread messages.",

"favoriteFruit": "apple"

},

{

"\_id": "56e18ce603784459df38b06c",

"index": 2,

"guid": "b19ffa1d-ca97-4e94-809e-3bf82df7fd40",

"isActive": true,

"balance": "$2,420.16",

"picture": "http://placehold.it/32x32",

"age": 30,

"eyeColor": "blue",

"name": {

"first": "Jane",

"last": "Wheeler"

},

"company": "DIGINETIC",

"email": "jane.wheeler@diginetic.co.uk",

"phone": "+1 (826) 545-3381",

"address": "385 Morgan Avenue, Manila, Puerto Rico, 8503",

"about": "Dolore velit dolor exercitation non voluptate cillum aliquip excepteur. Eiusmod mollit et nostrud pariatur amet reprehenderit deserunt elit ex. Do adipisicing qui pariatur cupidatat ut sint proident incididunt ipsum. Reprehenderit aliquip elit labore mollit consequat ipsum est sunt culpa. Est incididunt qui ea incididunt. Exerci tation pariatur laborum sit occaecat sint ea eiusmod et Lorem amet in magna elit. Eu v eniam eu qui laborum eiusmod esse ullamco ipsum proident exercitation et exercitation officia.",

"registered": "Saturday, July 4, 2015 9:47 PM",

"latitude": "-5.955075",

"longitude": "37.129517",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

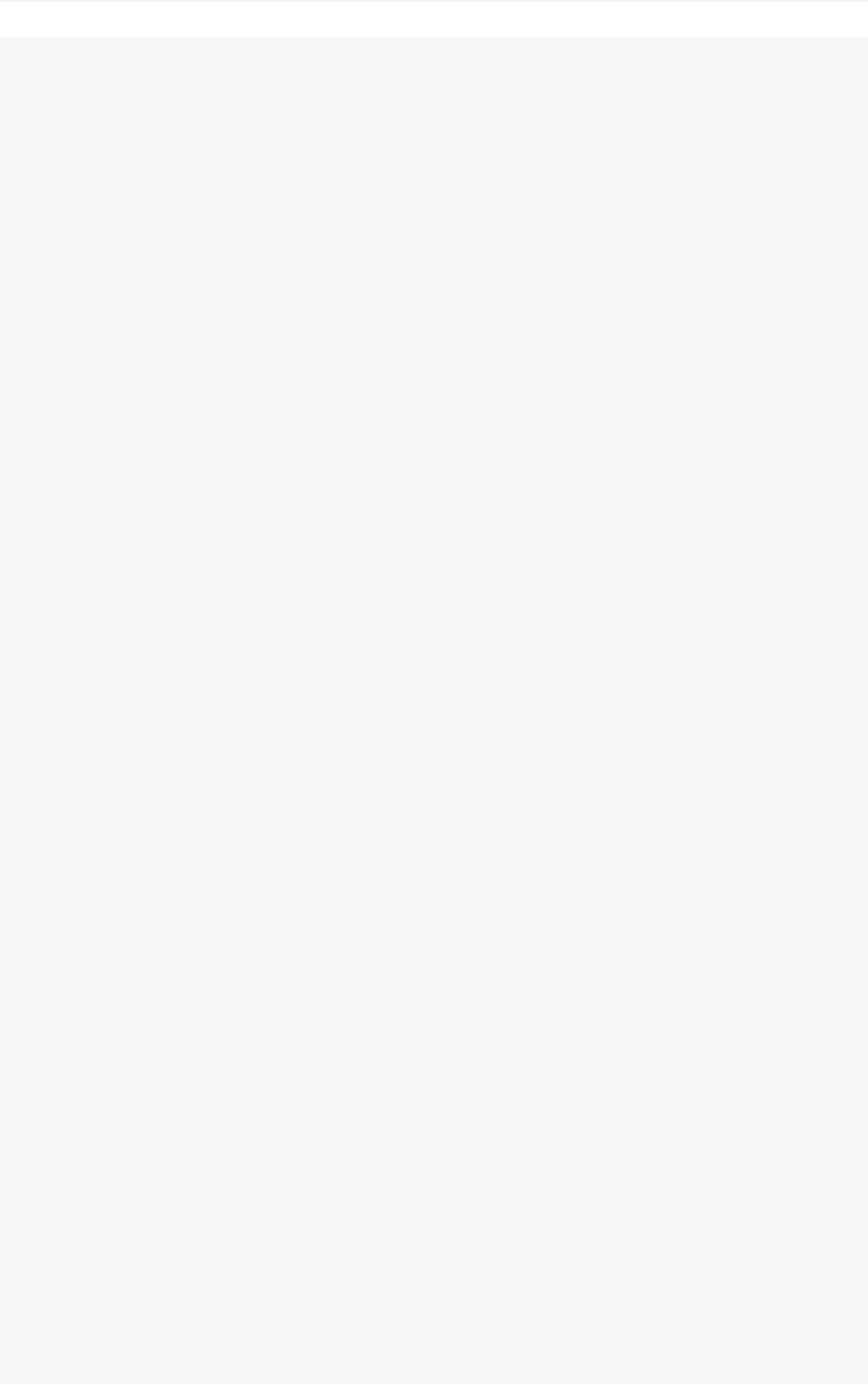
5,

6,



71

Exercises



7,

8,

9

],

"friends": [

3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Jane! You have 10 unread messages.",

"favoriteFruit": "apple"

},

{

"\_id": "56e18ce6adf25f0905c47a64",

"index": 3,

"guid": "d9547c25-8437-48d3-b3d6-ef890343b843",

"isActive": false,

"balance": "$2,059.14",

"picture": "http://placehold.it/32x32",

"age": 29,

"eyeColor": "green",

"name": {

"first": "Brennan",

"last": "Santos"

},

"company": "SPEEDBOLT",

"email": "brennan.santos@speedbolt.com",

"phone": "+1 (964) 417-3448",

"address": "327 Bills Place, Strong, Maryland, 4414",

"about": "Et dolor sit eiusmod eu labore velit. Laboris veniam consequat eiusmod a liqua ex in adipisicing deserunt quis eiusmod ullamco ut reprehenderit. Velit reprehen derit elit cupidatat laborum consequat ipsum quis consequat dolor magna sit nostrud. L aborum et minim irure ad elit dolore eu amet. Esse elit ex officia sit culpa pariatur nostrud anim sint nostrud culpa eiusmod non qui. Cupidatat ea dolor dolor ea pariatur et deserunt consequat est incididunt sit voluptate ipsum nostrud. Elit quis deserunt e st in qui sunt nulla ut.",

"registered": "Thursday, June 5, 2014 2:35 AM",

"latitude": "22.827405",

"longitude": "-50.704291",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

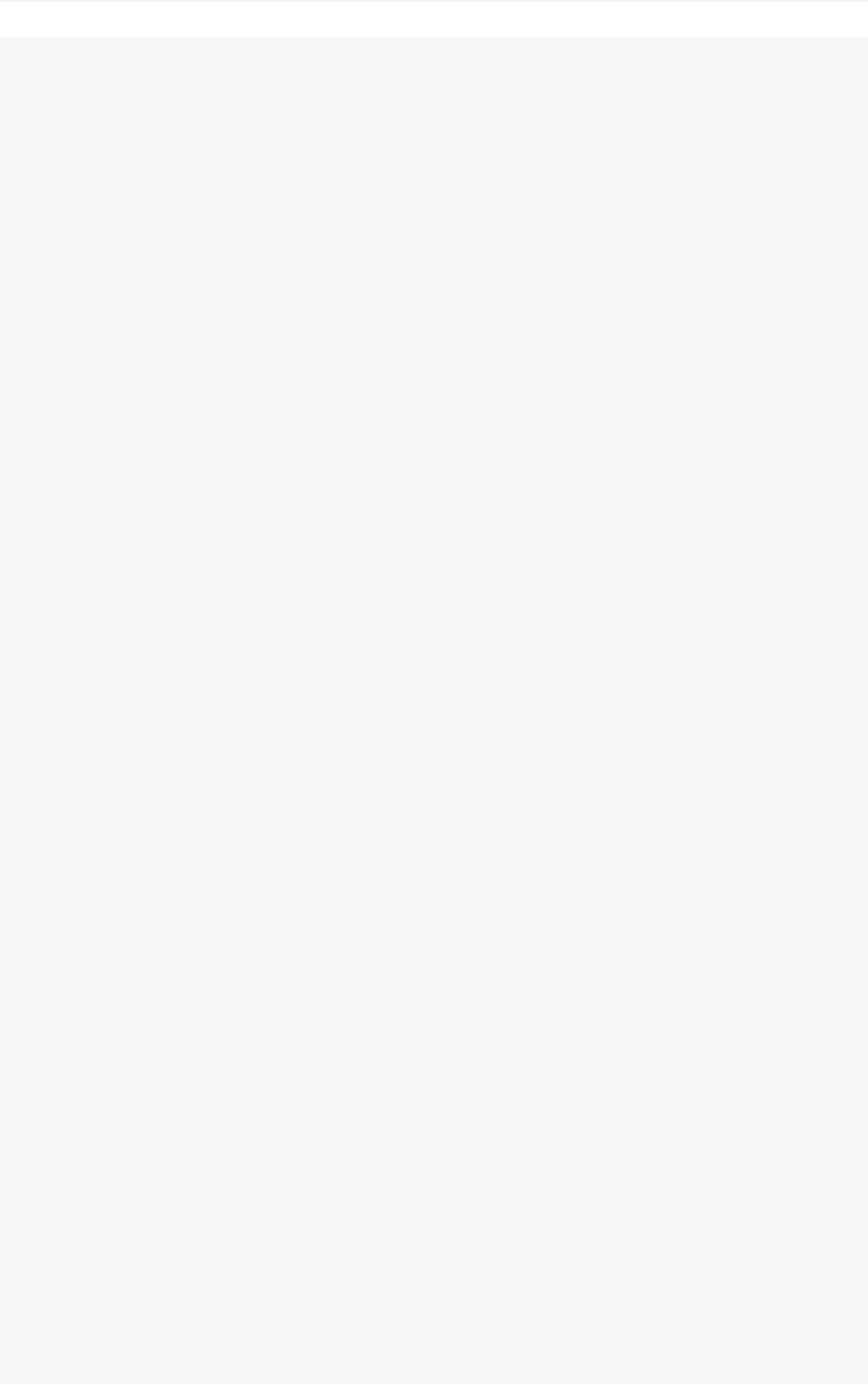
5,

6,



72

Exercises



7,

8,

9

],

"friends": [

3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Brennan! You have 9 unread messages.",

"favoriteFruit": "strawberry"

},

{

"\_id": "56e18ce671021dc16753b56d",

"index": 4,

"guid": "725fb6f9-d900-4764-8f41-7fe2779b2dc9",

"isActive": false,

"balance": "$2,399.10",

"picture": "http://placehold.it/32x32",

"age": 37,

"eyeColor": "blue",

"name": {

"first": "Perez",

"last": "Turner"

},

"company": "CENTICE",

"email": "perez.turner@centice.us",

"phone": "+1 (855) 446-3306",

"address": "596 Varick Street, Genoa, Arkansas, 6957",

"about": "Veniam est dolor laboris eiusmod. Nostrud duis est nostrud aliquip in la borum qui culpa. Sunt mollit adipisicing amet laboris esse.",

"registered": "Monday, September 8, 2014 11:25 PM",

"latitude": "23.65985",

"longitude": "-65.321713",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

5,

6,

7,

8,

9

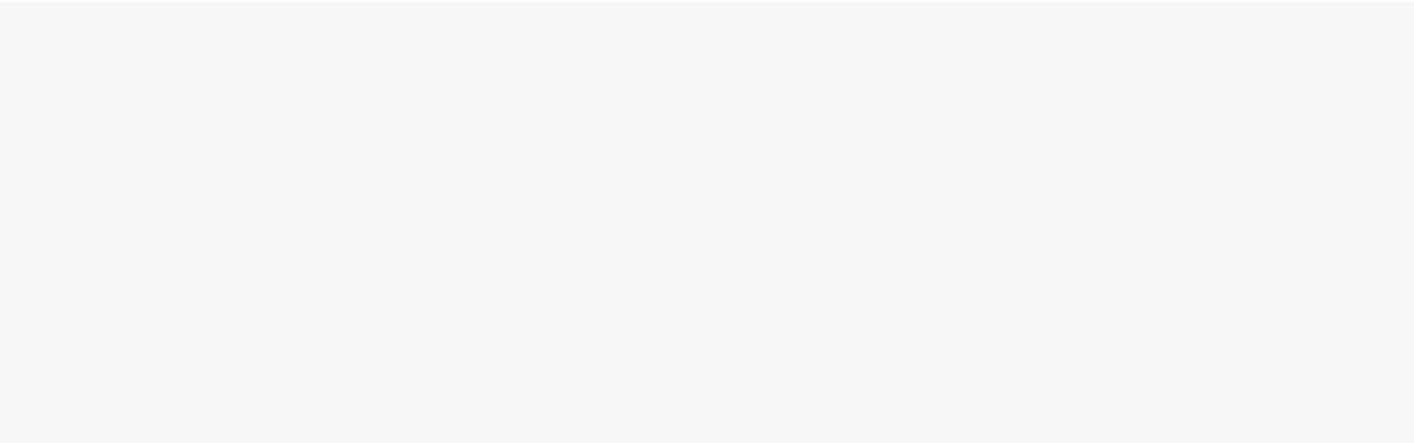
],

"friends": [



73

Exercises



3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Perez! You have 5 unread messages.",

"favoriteFruit": "banana"

}

]



74

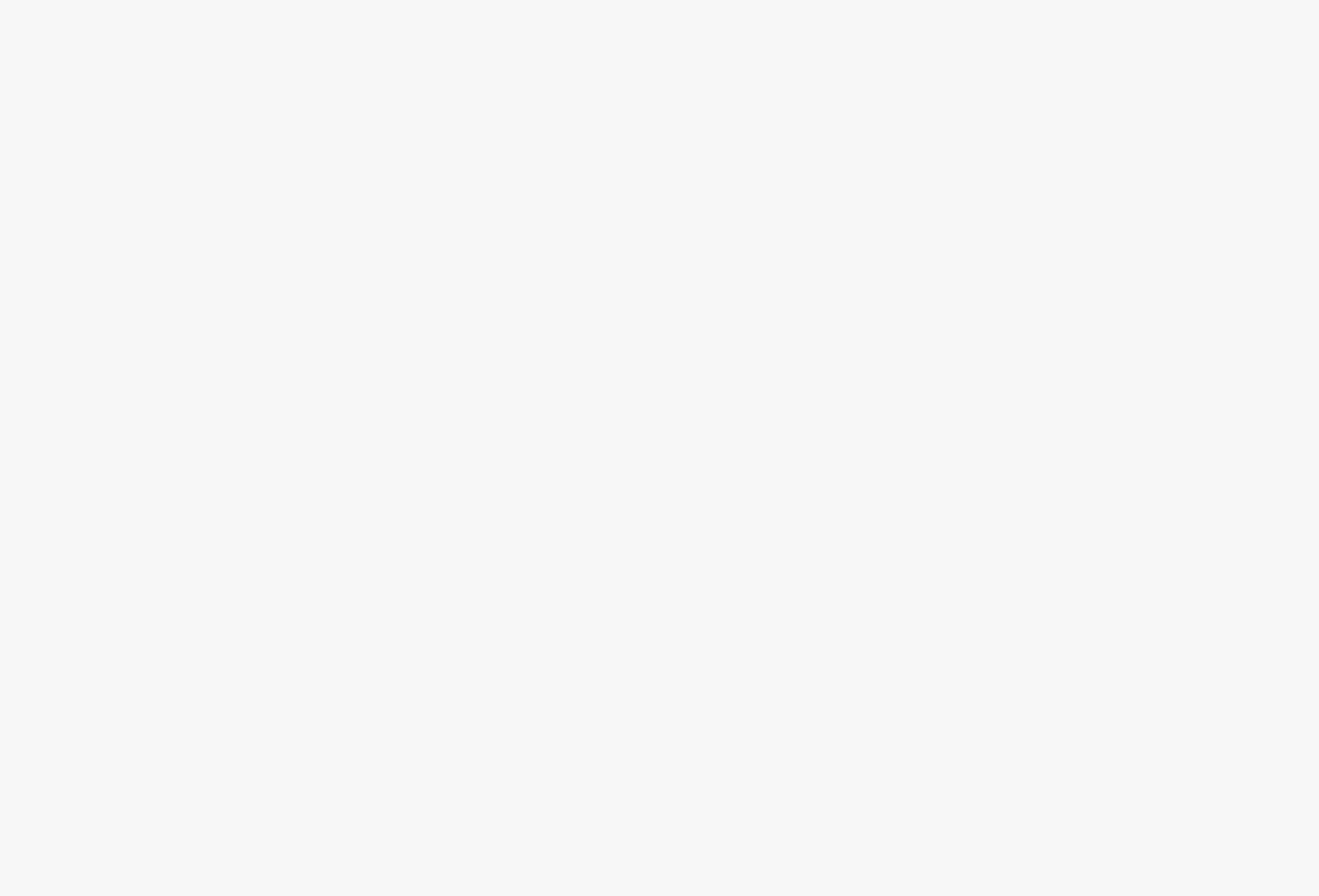
Task #1



**Task #1**

**Generate a list of the names of people**

Expected Result



[

{

first: 'Lauren',

last: 'Stanley'

},

{

first: 'Schmidt',

last: 'Floyd'

},

{

first: 'Jane',

last: 'Wheeler'

},

{

first: 'Brennan',

last: 'Santos'

},

{

first: 'Perez',

last: 'Turner'

}

]



**Answer**

Map over the Immutable list and then use the get method to access the name prop.



75

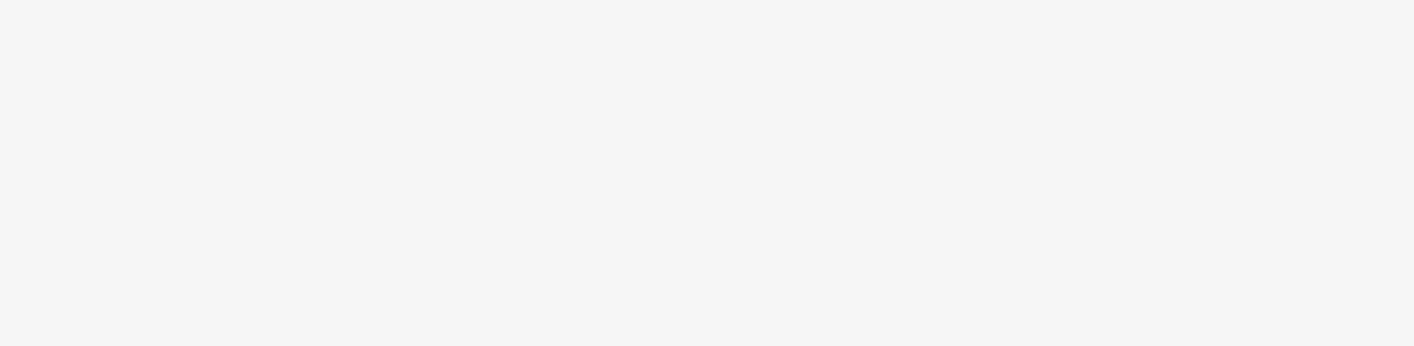
Task #2



**Task #2**

**Generate a list of last names of people**

Expected Result



[

'Stanley',

'Floyd',

'Wheeler',

'Santos',

'Turner',

]



**Answer**

Map over the Immutable list and then use the getIn method to access the name.last prop.



76

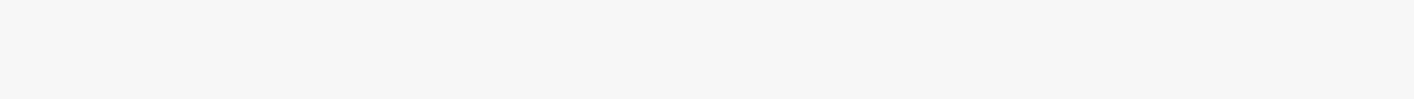
Task #3



**Task #3**

**Get the last name of the last person**

Expected Result



'Turner'



**Answer**

Use last to get the last item in an Immutable list. Then use the getIn method to access the name.last prop.



77

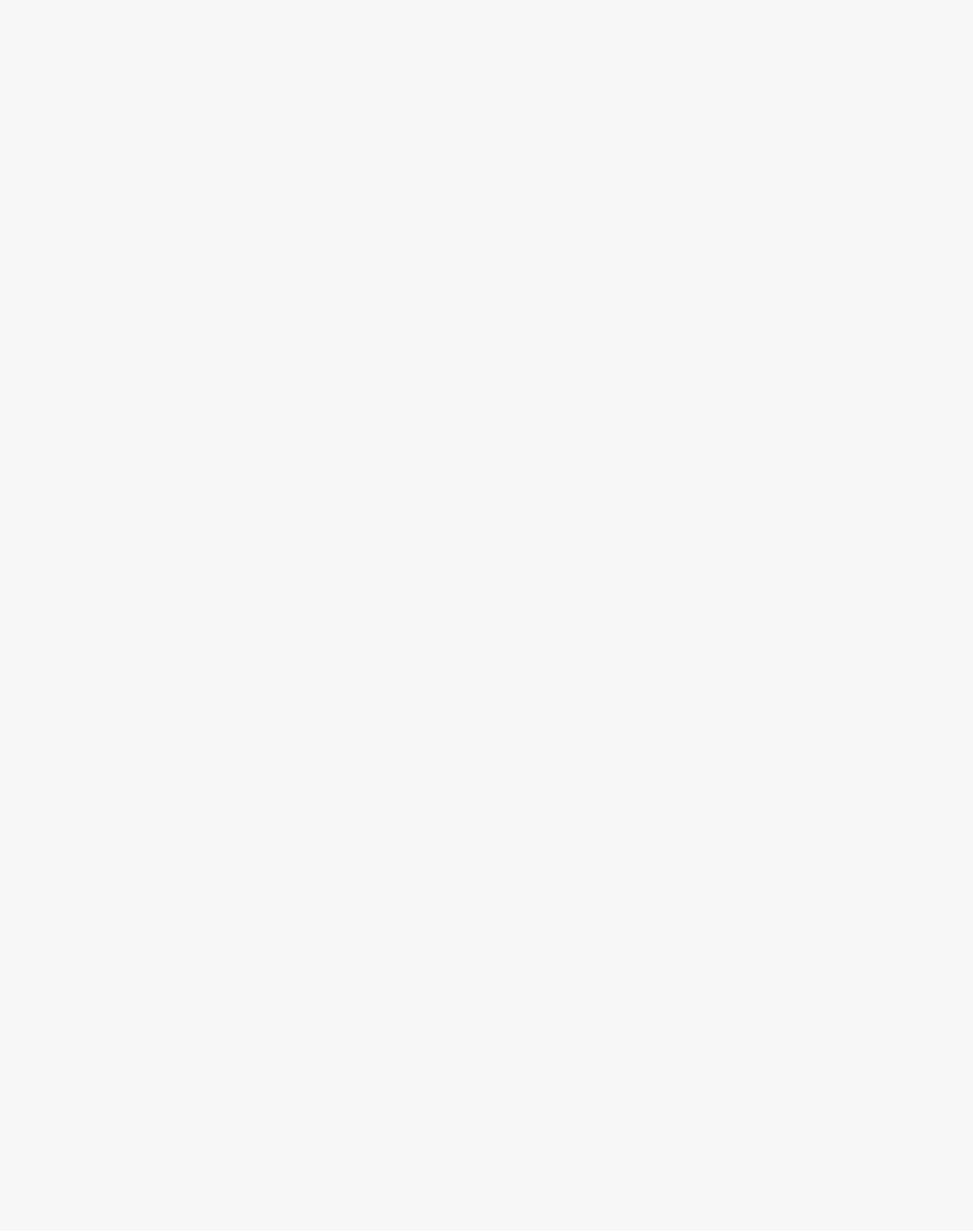
Task #4



**Task #4**

**Generate a list of users with brown eyes**

Expected Result



[

{

"\_id": "56e18ce6dc7d5ade1e3c7889",

"index": 1,

"guid": "7ceca65c-cc8d-4f88-ab00-b5d00b72e27f",

"isActive": true,

"balance": "$1,423.68",

"picture": "http://placehold.it/32x32",

"age": 35,

"eyeColor": "brown",

"name": {

"first": "Schmidt",

"last": "Floyd"

},

"company": "ANIXANG",

"email": "schmidt.floyd@anixang.org",

"phone": "+1 (913) 595-3119",

"address": "274 Norfolk Street, Freeburn, Nevada, 1869",

"about": "Exercitation deserunt quis commodo ad qui aliqua proident mollit labore mollit. Deserunt occaecat in pariatur mollit aute consequat reprehenderit in deserunt magna ad. Aliquip labore do mollit officia laboris in aliquip magna aliqua. Sunt occae cat eiusmod ea amet dolore consectetur aute consequat adipisicing et nisi fugiat. Aute eiusmod quis duis ipsum occaecat culpa eiusmod Lorem amet laborum occaecat adipisicin g minim. Labore exercitation laborum sint enim veniam labore officia. Aliquip do esse consectetur amet.",

"registered": "Sunday, October 12, 2014 8:17 AM",

"latitude": "-3.271053",

"longitude": "-124.321634",

"tags": [

7,

"veniam"

],

"range": [

0,

1,

2,

3,

4,

5,

6,

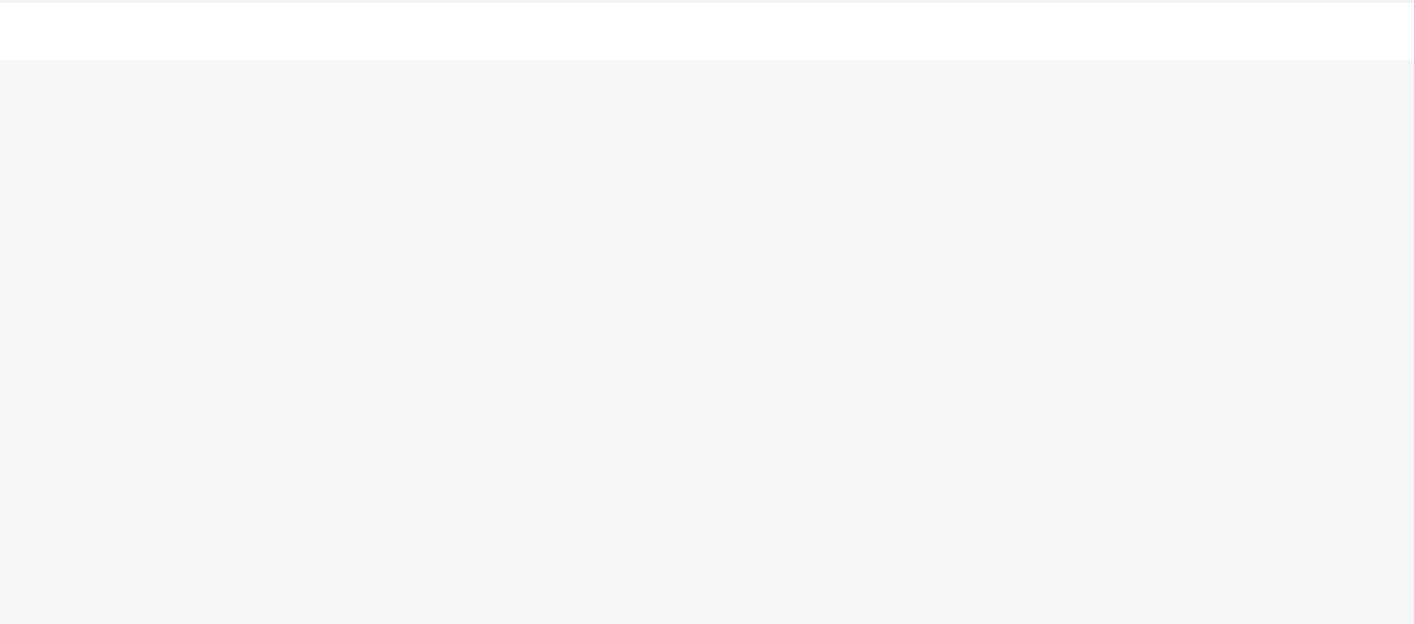
7,

8,



78

Task #4



9

],

"friends": [

3,

{

"id": 1,

"name": "Mccall Petersen"

}

],

"greeting": "Hello, Schmidt! You have 9 unread messages.",

"favoriteFruit": "apple"

}

]



**Answer**

Use Immutable list's filter method.

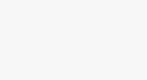


79

Task #5

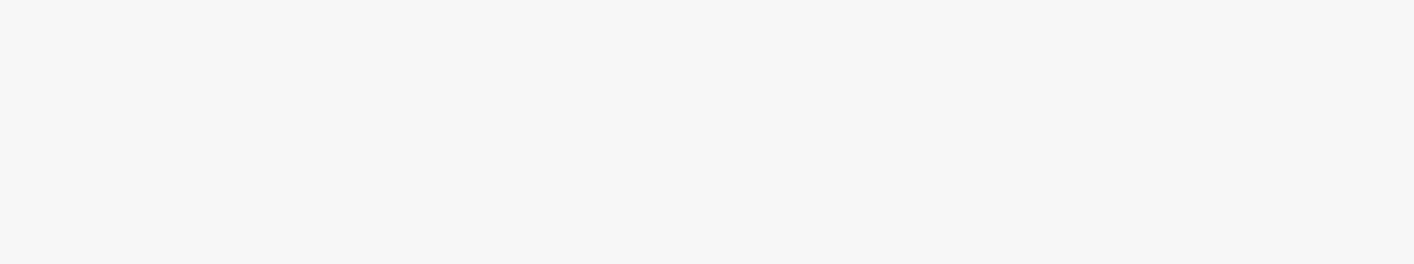


**Task #5**



**Generate a Map of the eye colors to count of people with that color**

Expected Result



{

blue: 3,

brown: 1,

green: 1

}



**Answer**

Use the Immutable list's groupBy method to generate groups of people by eye color. Then use map to generate counts. Or just use the List's reduce method.



80

Task #6



**Task #6**

**Add your own profile to the end of the list**



**Answer**

Use the Immutable list's push method.



81

Task #7



**Task #7**

**Change the name of the first person to Marty Robbins**



**Answer**

Use the Immutable list's setIn method.



82

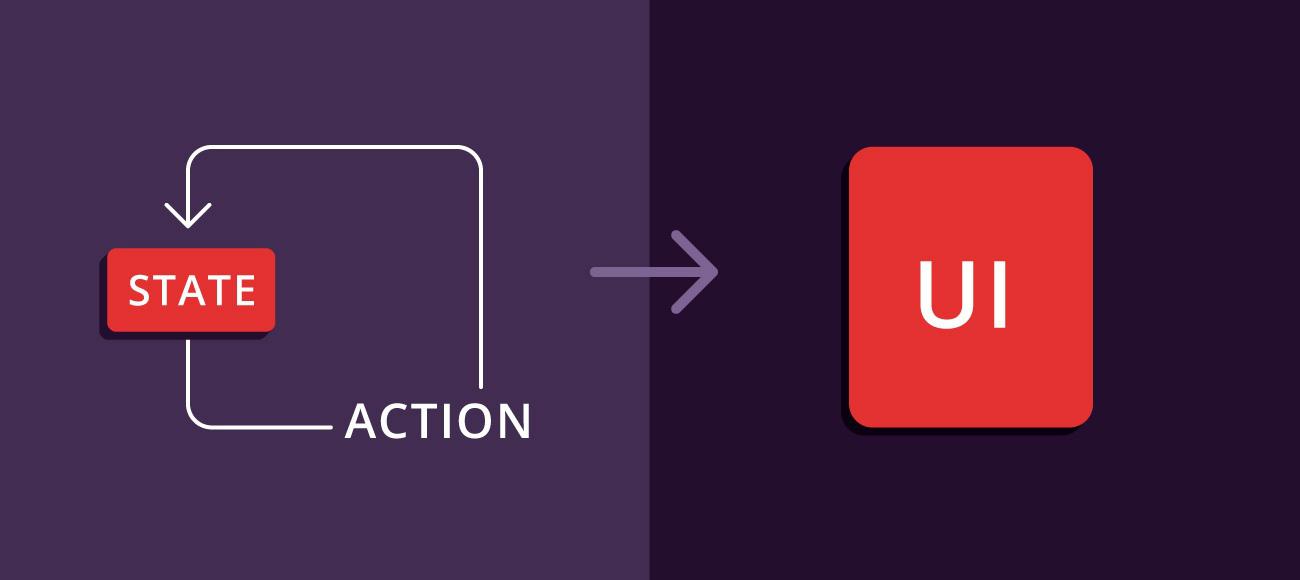
Redux



**Redux**

Redux is an application state manager for JavaScript applications, and keeps with the core principals of flux-architecture by having a unidirectional data flow in your application.

How it differs from traditional Flux though, is that instead of multiple stores, you have one global application state. The state is calculated and returned in the reducer. The state management is held else where.



**Three Principles of Redux**

**1. Single Source of Truth**

Entire state is stored in an object tree



Easy to debug



Easy to inspect application



Easy to hydrate initial state



**2. State Is Read Only**

Only way to mutate state is to emit an action



Actions describe what happened



Views, network callbacks, etc. will never mutate state



Mutations are centralized and happen in strict order



No race conditions



Actions are objects, they can be logged, serialized, stored, and replayed



83

Redux



**3. Changes Are Made With Pure Functions**

Reducers are responsible for modifying the state tree



Pure functions



Take in previous state, action, and return new state



Can be split out into smaller reducers to manage specific parts of state tree



**Resources**

[Redux Documentation](http://redux.js.org/)



[React-Redux - React Bindings for Redux](https://github.com/reactjs/react-redux)



[React Redux Starter Kit](https://github.com/rangle/react-redux-starter)



[Getting Started with Redux - Egghead.io](https://egghead.io/series/getting-started-with-redux)



[Idiomatic Redux - Egghead.io](https://egghead.io/courses/building-react-applications-with-idiomatic-redux)



84

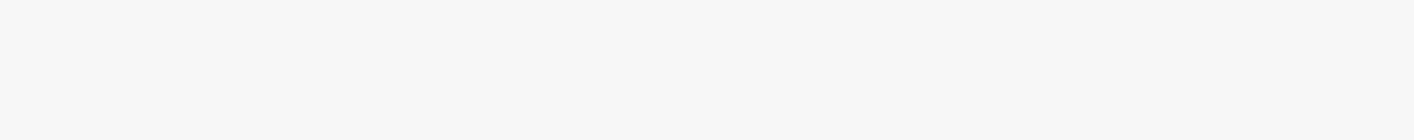
Review of Reducers and Pure Functions



**Quick Review of Reducers and Pure Functions**

One of the core concepts of Redux is the reducer. A reducer is simply a pure function that iterates over a collection of values, and returns a new single value at the end of it.

The simplest examples of a reducer, is a sum function:



const x = [1,2,3].reduce((value,state) => value + state, 0)

// x == 6



85

Redux Reducers

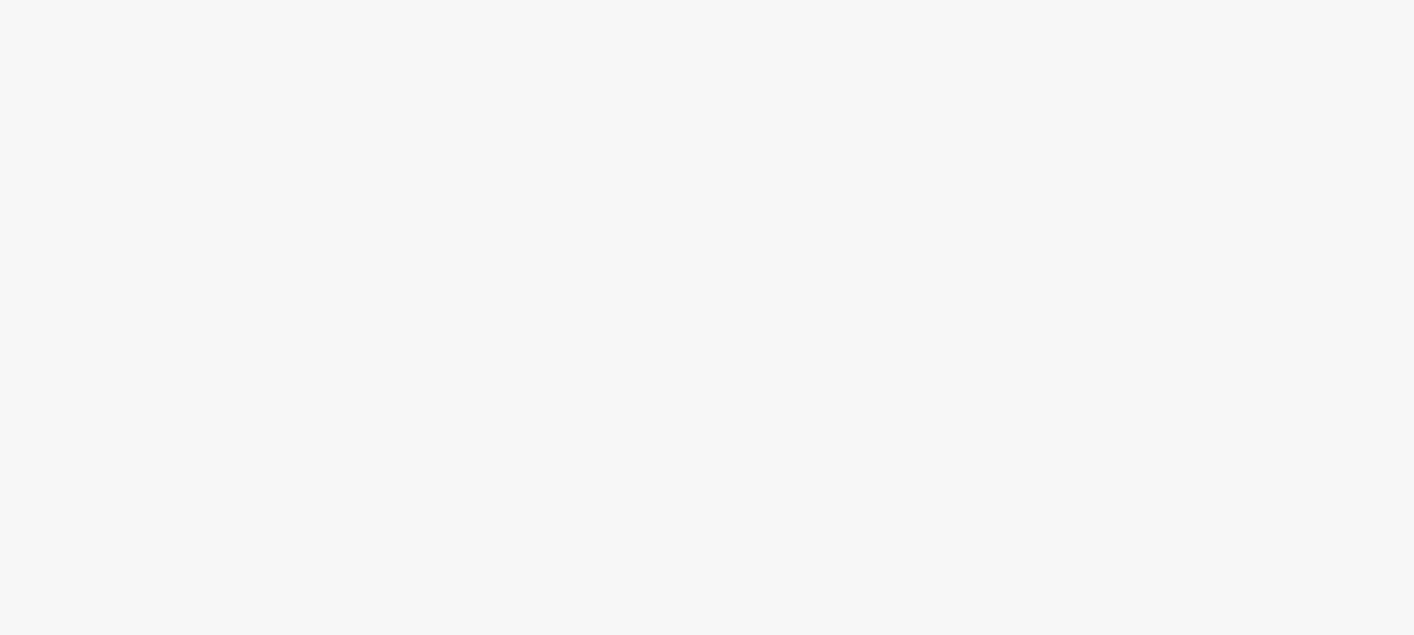


**Redux Reducers**

While a very simple idea, it is very powerful. With Redux, you replay a series of events into the reducer - and get your new application state as a result.

Reducers in a Redux application should not mutate the state, but return a copy of it, and be side-effect free. Lets take a look at a simple counter reducer.

**Simple Reducer**



const INCREMENT\_COUNTER = 'INCREMENT\_COUNTER'; const DECREMENT\_COUNTER = 'DECREMENT\_COUNTER';

export default function counter(state = 0, action) {

switch (action.type) {

case INCREMENT\_COUNTER:

return state + 1;

case DECREMENT\_COUNTER:

return state - 1;

default:

return state;

}

}

We can see here that we are passing in an initial state, and an action. To handle each action

* we have setup a switch statement. Instead of each reducer needing to explicitly subscribe to the dispatcher - every action gets passed into every reducer, which handles the action it is interested in or otherwise returns the state along to the next reducer.

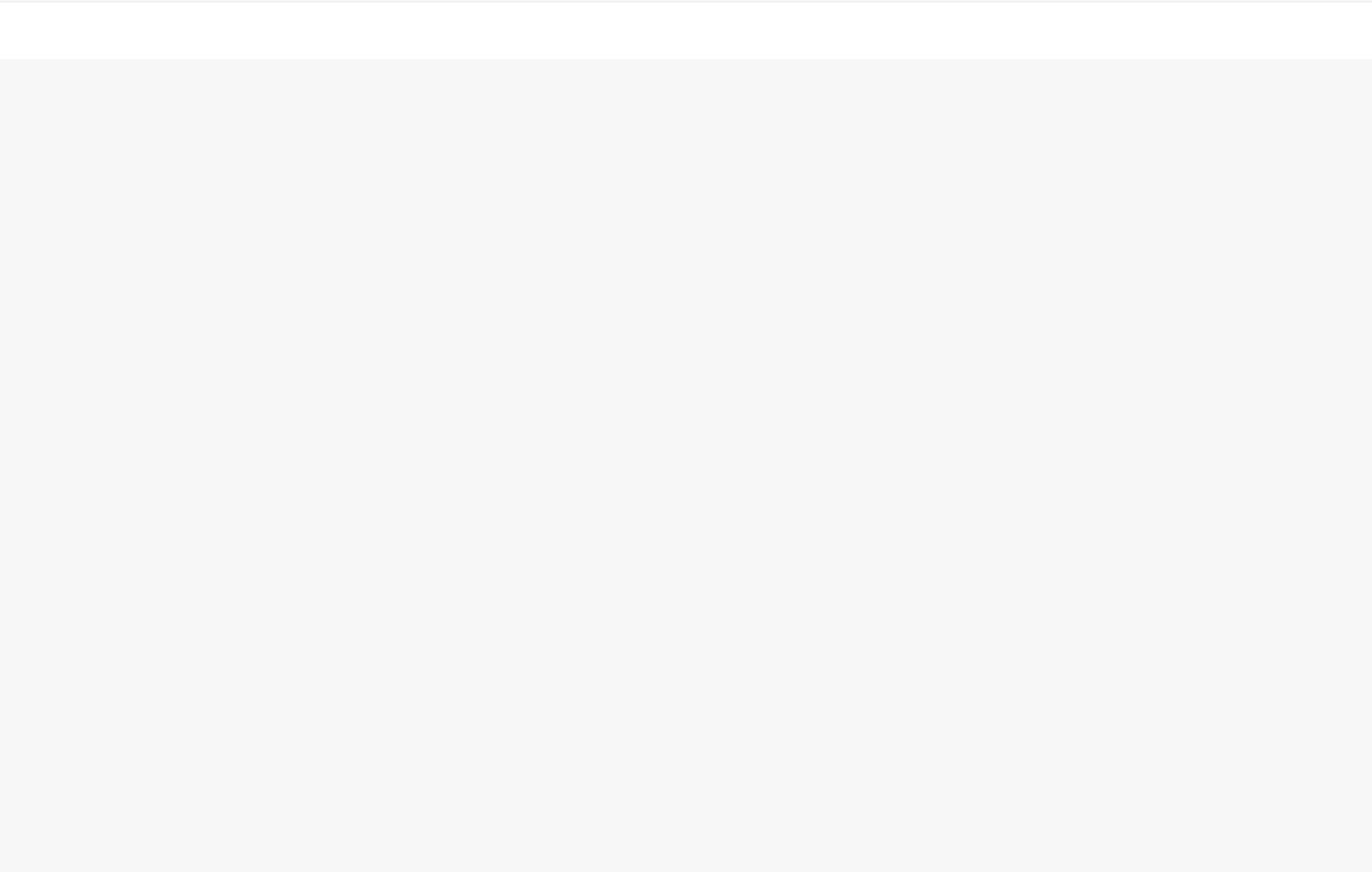
Reducers in Redux should be side-effect free, that means that they should not modify things outside of the application state. Instead, they should reflect the state of the application. This is why side-effect causing operations, such as updating a record in a database, generating an id, etc should be handled elsewhere in the application - such as in the action creators, or middleware.

Another consideration when creating your reducers, is to ensure that they are immutable and not modifying the state of your application. If you mutate your application state, it can cause unexpected behavior. There are a few ways to help maintain immutability in your reducers. One, is using new ES6 features such as the spread operator for objects and arrays.



86

Redux Reducers



let immutableObjectReducer = (state = { someValue: 'value'} , action) => { switch(action.payload) {

case SOME\_ACTION:

return Object.assign({}, state, { someValue: action.payload.value });

default:

return state;

}

}

let immutableArrayReducer = (state = [1,2,3], action) => { switch(action.payload) {

case ADD\_ITEM:

return [...state,action.payload.value]

break;

default:

return state;

}

}

However, if dealing with complex or deeply nested objects - it can be difficult to maintain immutability in your application using this syntax. This is where a library like ImmutableJS can help.



87

Redux Actions



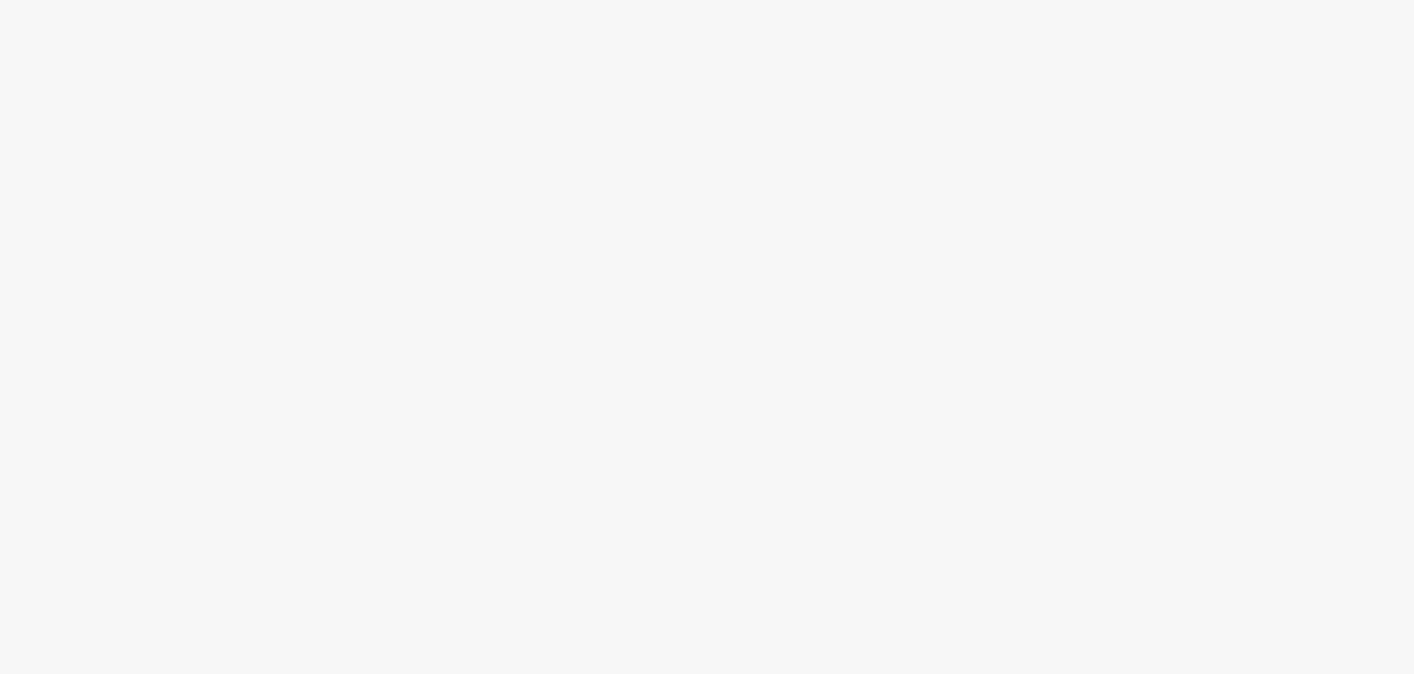
**Redux Actions**

Redux actions, generally should return a simple JSON object. This is because they should be serialized and replayable into the application state. Even if your actions need to return promises, the final dispatched action should remain a plain JSON object.

Redux actions are generally where side-effects should happen, such as making API calls, or generating ID's. This is because when the final action gets dispatched to the reducers, we want to update the application state to reflect what has already happened.

Lets take a look at the actions that are used in this example. For now, lets just focus on the synchronous actions.

**Synchronous Actions**



const INCREMENT\_COUNTER = 'INCREMENT\_COUNTER'; const DECREMENT\_COUNTER = 'DECREMENT\_COUNTER';

export function increment() {

return {

type: INCREMENT\_COUNTER

};

}

export function decrement() {

return {

type: DECREMENT\_COUNTER

};

}

As you can see, the actions creators are simple functions that take parameters, and return a JSON object containing more information. Actions follows the [Flux Standard Action](https://github.com/acdlite/flux-standard-action) and contain the properties:

**type**: a string/enum representing the action



**payload**: the data that you want to pass into the reducer if applicable



**meta?** : optional - any extra information



When using Redux, libraries like react-redux will take care of wrapping your actions into the dispatch so they will get passed off to the store appropriately.



88

Redux Actions

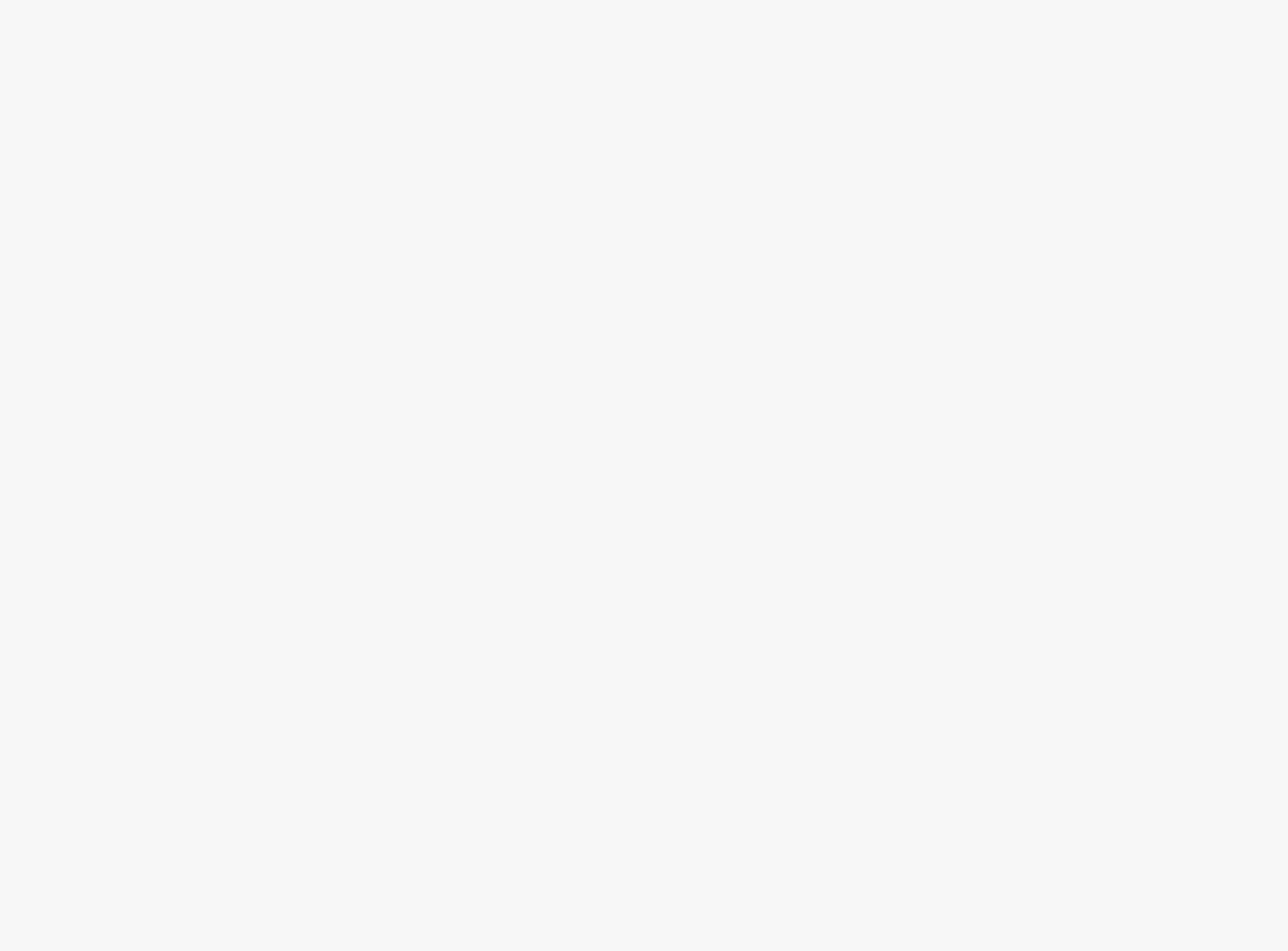


**Asynchronous Actions**

To do async operations, or have actions that return something other than a plain JSON object, you need to register a middleware with redux. For our examples, we can use the thunk middleware, and setting this up is covered later in the training. For now, all you need to know is that once you register a middleware with redux, you can make dispatch and getState available to your actions. To show how these are used, lets take a look at the



incrementIfOdd and increaseAsync actions.



// ...

export function incrementIfOdd() {

return (dispatch, getState) => {

const { counter } = getState();

if (counter % 2 === 0) {

return;

}

dispatch(increment());

};

}

const delay = (timeInMs) => {

return new Promise((resolve,reject) => {

setTimeout(() => resolve() , timeInMs);

});

}

export function incrementAsync(timeInMs = 1000) { return dispatch => {

delay(timeInMs).then(() => dispatch(increment())); };

}

In the incrementIfOdd action, we are making use of the getState function to get the current state of the application.

In the incrementAsync action, we are making use of dispatch. For example, we have created a Promise that will resolve after the delay. Once the Promise resolves, we can then do a dispatch with the increase action. However, this promise could also be an API call, with the dispatched action containing the result of the API call.



89

Configuring your Application to use Redux



**Configuring Your Application to Use Redux**

Once you have the reducers and actions created, it is time to configure your application to make use of Redux. For this, we will need to:

Create our application reducer



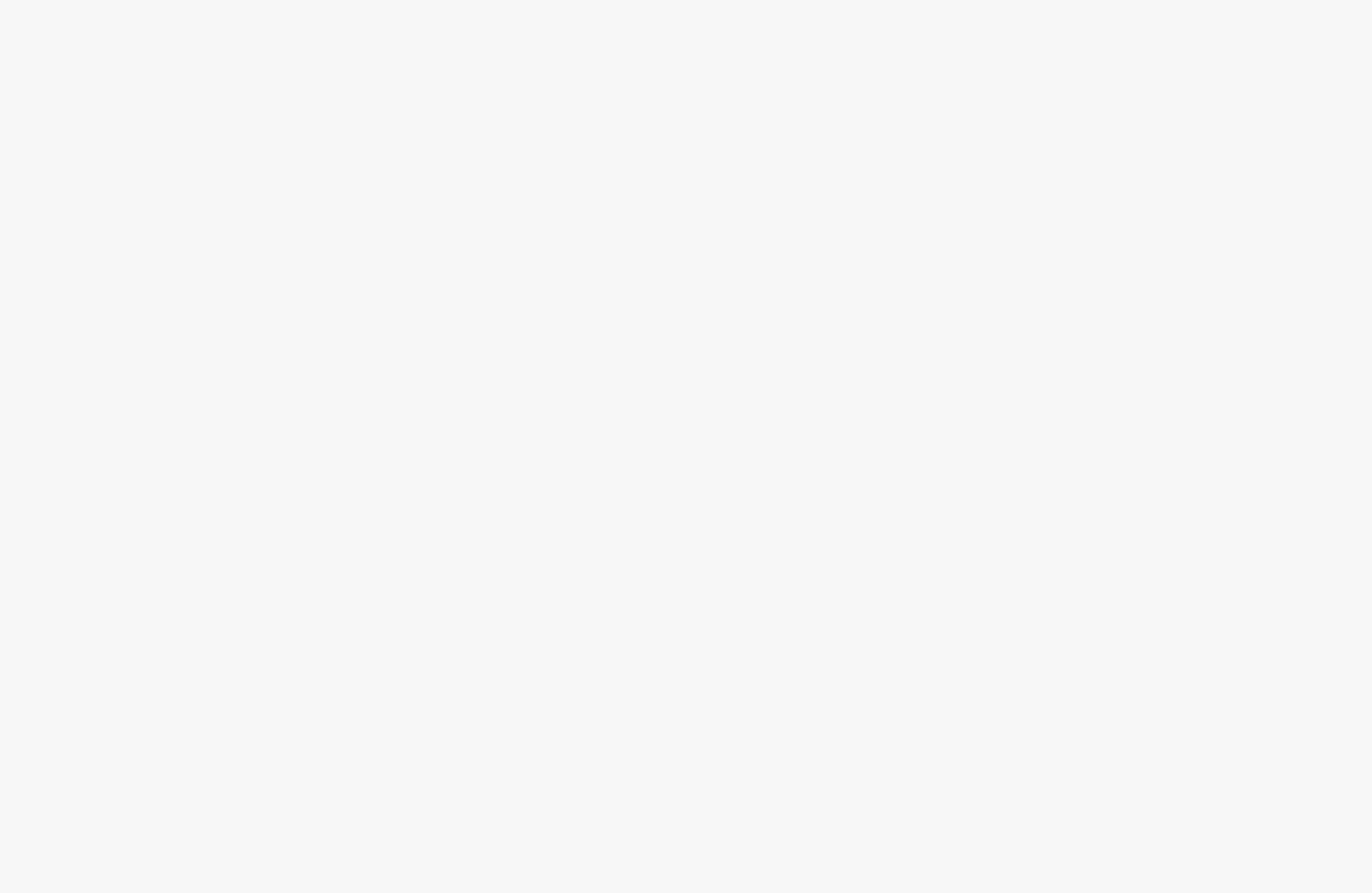
Create and configure a store



Subscribe to the store and update the view



**Create Our Application Reducer**



import { createStore, combineReducers } from 'redux';

export const INCREASE = '@@reactTraining/INCREASE'; export const DECREASE = '@@reactTraining/DECREASE';

const INITIAL\_STATE = 0;

function counterReducer(state = INITIAL\_STATE, action = {}) { switch (action.type) {

case INCREASE:

return state + 1;

case DECREASE:

return state - 1;

default:

return state;

}

}

const rootReducer = combineReducers({ counter: counterReducer });

What combineReducers does, is allows us to break out our application into smaller reducers with a single area of concern. Each reducer that you pass into it, will become a property on the state. So when we are subscribing to our state changes, we will be passed in a state object with a property counter, and any other reducers you have provided.

**Create and Configure a Store**



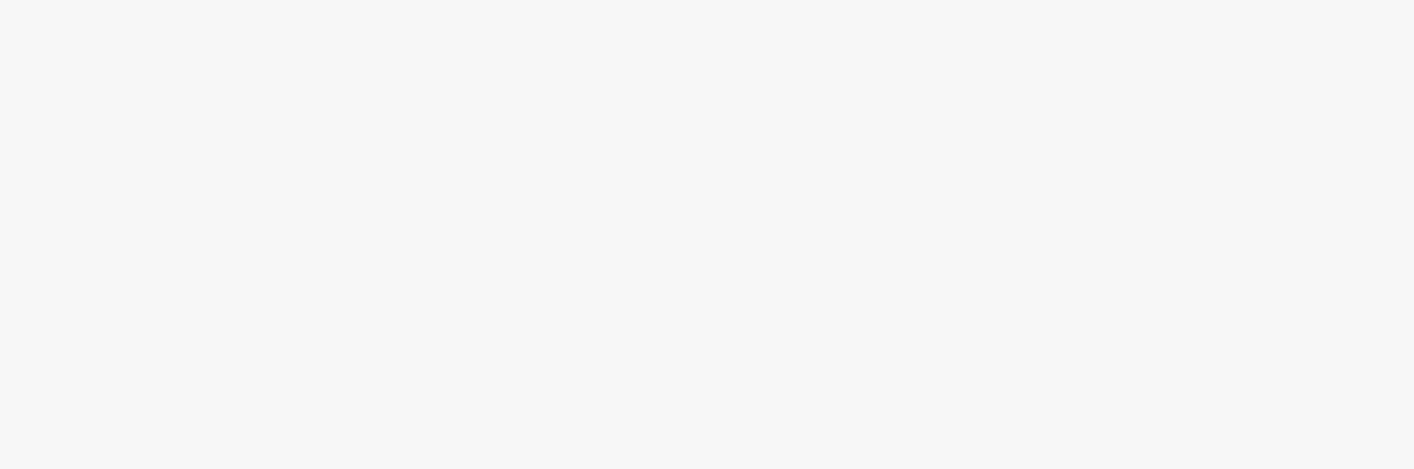
90

redux-logger

Configuring your Application to use Redux



When creating a store in redux, this is where you provide the middleware you want to use, and the reducer that you want to have for your application.



import { createStore, applyMiddleware, compose } from 'redux'; import thunk from 'redux-thunk';

//...

const store = compose(

applyMiddleware(

thunk

),

)(rootReducer);

In this example, we are creating a store that is using the thunk middleware, which will allow our actions to return non-JSON objects such as promises. We could also use other

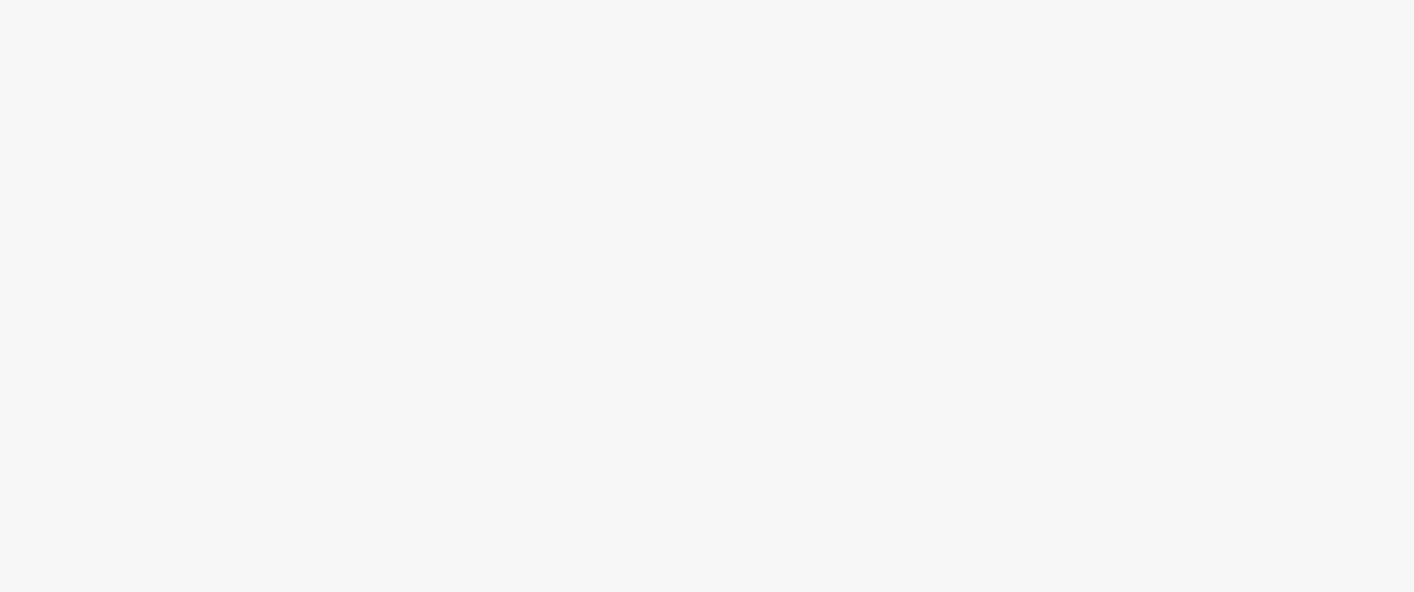


middlewares such as , which will provides some logging functionality to the

application.

**Subscribe to State Changes**

Now that we have created our state reducer, and created a store. We now need to subscribe to the store and update our view with the latest state.



//...

store.subscribe(() => {

ReactDOM.render(

<div>

<pre>{ JSON.stringify(store.getState(), null, 2) }</pre>

<button onClick={ () => store.dispatch(increase()) }>Increase</button> <button onClick={ () => store.dispatch(decrease()) }>Decrease</button>

</div>,

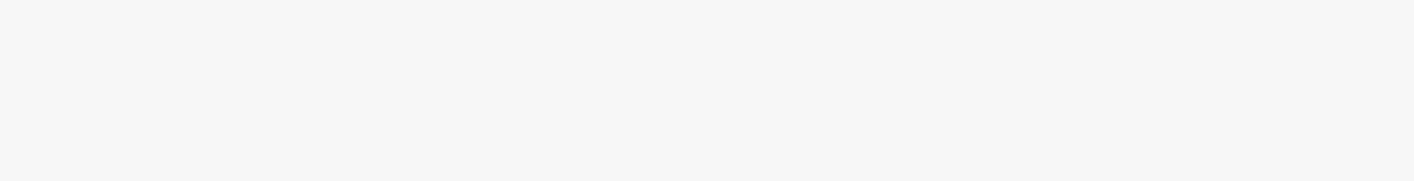
document.getElementById('root')

);

});

store.dispatch({ type: 'INIT' });

**Full Example**



git checkout 04-redux

jspm install

npm run dev



91

Configuring your Application to use Redux



92

Using Redux with Components



**Using Redux With Components**

Instead of having to manage the store subscribtions manually we can use react-redux to connect our store to React components. To demonstrate how this works, let's take a look at the counter example.

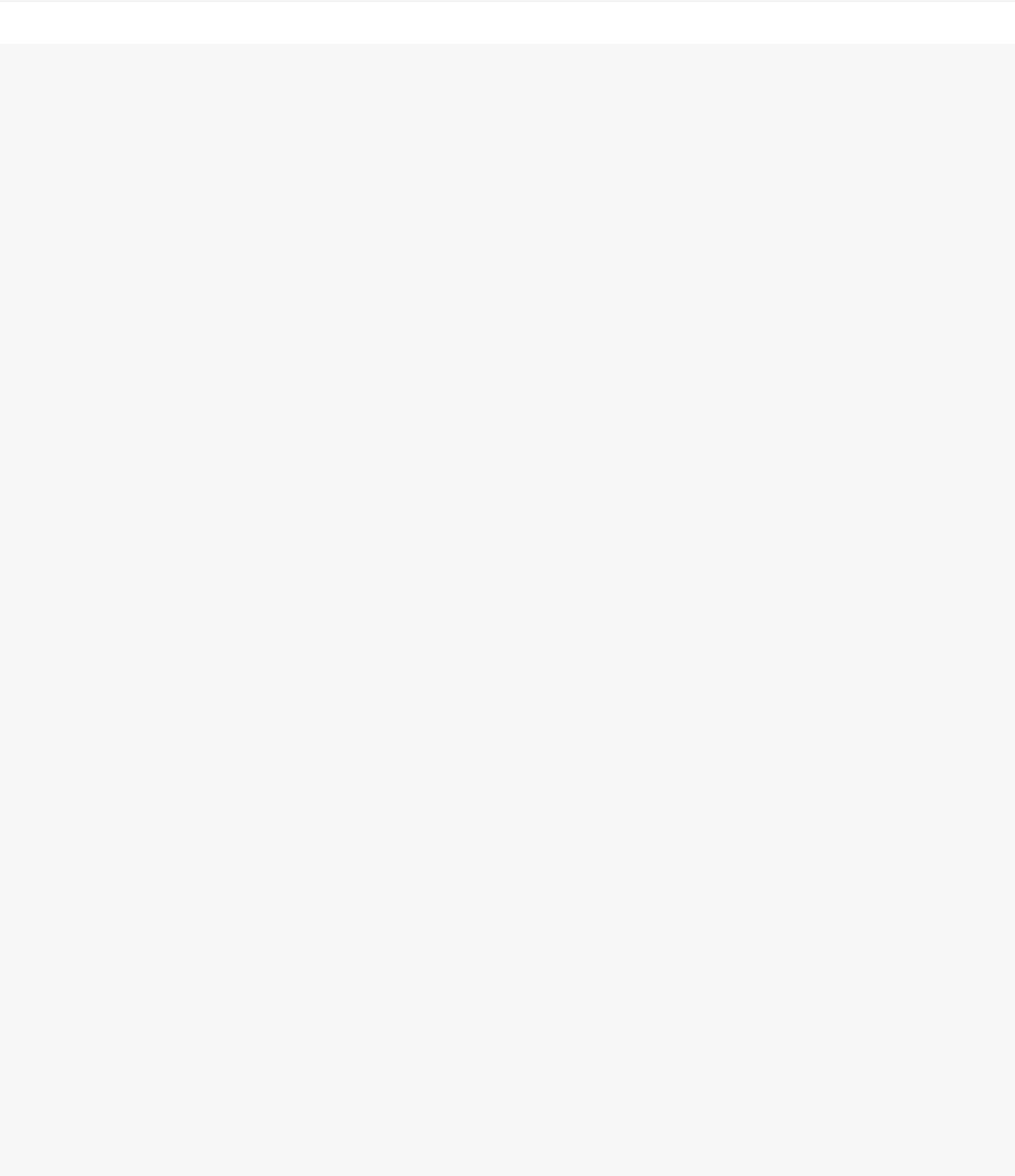
**Counter Example**

We start by building out a counter component. The component will be responsible for keeping track of how many times it was clicked, and displaying the amount.



93

Using Redux with Components



import React from 'react';

import { connect } from 'react-redux';

import { increase, decrease } from '../reducers/counter';

function mapStateToProps(state) {

return {

counter: state.counter,

};

}

function mapDispatchToProps(dispatch) {

return {

onIncrease: () => dispatch(increase()),

onDecrease: () => dispatch(decrease()),

};

}

const Counter = ({ onIncrease, onDecrease, counter }) => { return (

<div>

<pre>{ counter }</pre>

<button onClick={ onIncrease }>Increase</button> <button onClick={ onDecrease }>Decrease</button>

</div>

);

};

const App = connect(

mapStateToProps,

mapDispatchToProps,

)(Counter);

ReactDOM.render(

App,

document.getElementById('root')

);

The template syntax should be familiar by now, displaying a counter value, and handling some click events. Lets take a look at the use of connect .

mapStateToProps : connect will subscribe to Redux store updates. Any time it updates, mapStateToProps will be called. It's result must be a plain object. Which will then be passed to the component as props.



mapDispatchToProps : Optional. The store's dispatch method is passed in as an argument here. You can then use that to wrap your actions or pass dispatch into them. The result of this function is also passed into the component as props. Tip: you could use the [bindActionCreators()](http://gaearon.github.io/redux/docs/api/bindActionCreators.html) helper from Redux to simplify this.

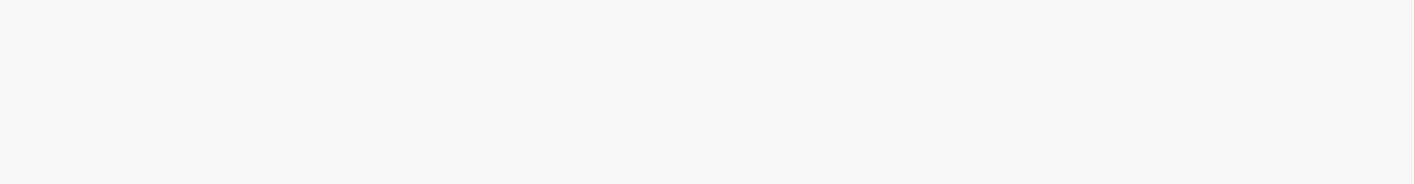


94

Using Redux with Components



**Full Example**



git checkout 04a-redux

jspm install

npm run dev



95

Redux and Component Architecture



**Redux and Component Architecture**

In the previous example, our counter component is a smart component. It knows about redux, the structure of the state, and the actions it needs to call. While in theory you can drop this component into any area of your application and have it just work. But, it will be tightly bound to that specific slice of state, and those specific actions. For example, what if we wanted to have multiple counters tracking different things on the page? For example, counting the number of red clicks vs blue clicks.

To help make components more generic and reusable, it is worth considering smart component, or container components - and dumb components.

|  |  |  |
| --- | --- | --- |
|  | **Container Components** | **Presentational Components** |
|  |  |  |
| **Location** | Top level, route handlers | Middle and leaf components |
|  |  |  |
| **Aware of Redux** | Yes | No |
| **To read data** | Subscribe to Redux state | Read data from props |
|  |  |  |
| **To change data** | Dispatch Redux actions | Invoke callbacks from props |
| [Redux Docs](http://redux.js.org/docs/basics/UsageWithReact.html) |  |  |



96

Routing



**Routing in React**

This section will look at setting up [React Router](https://github.com/reactjs/react-router).



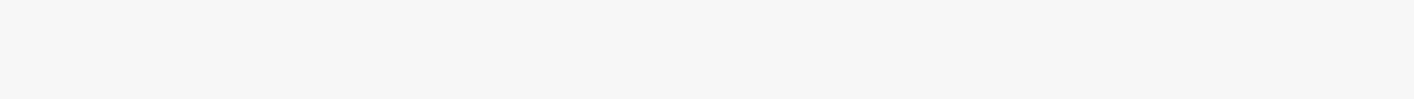
97

React Router



**React Router**

**Setup**



npm install --save react-router

**Routing**

[React router](https://github.com/reactjs/react-router/) is the root component rendered, which will render the app components as children based on the url path.

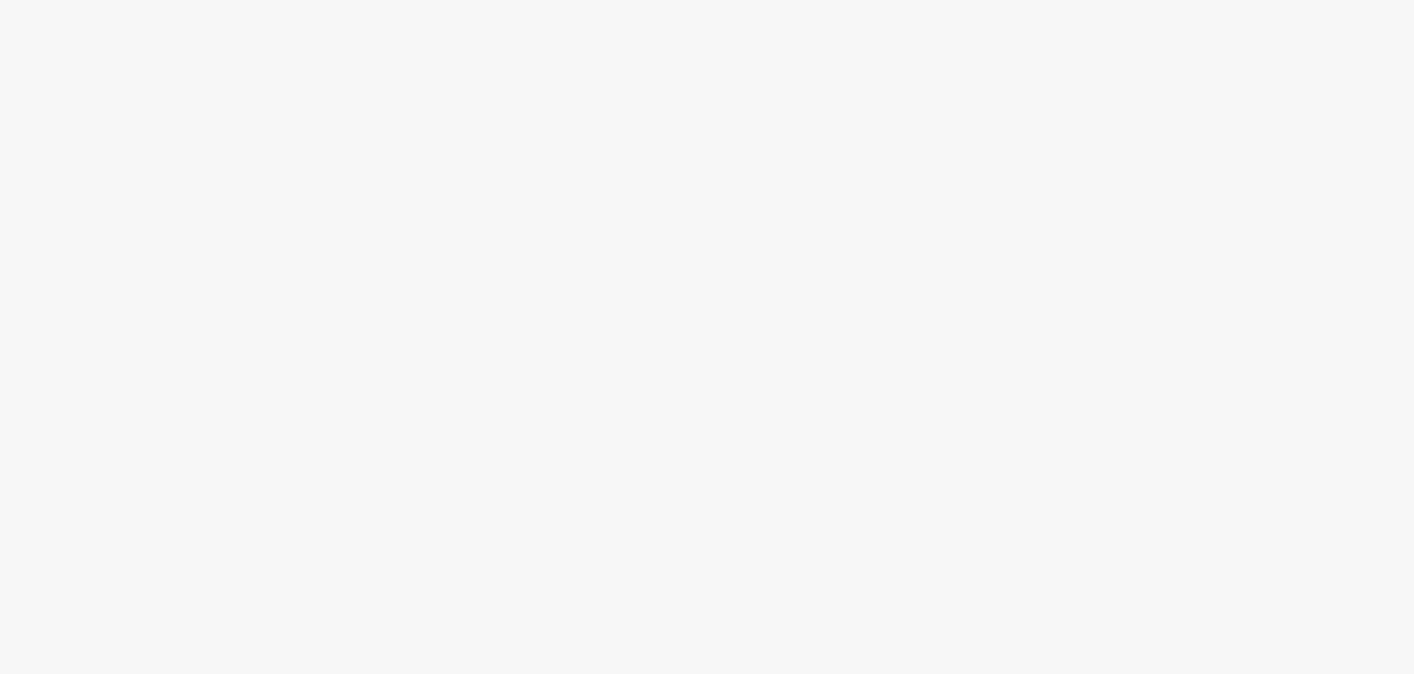
React router has configuration components: Router , Route , IndexRoute , and Redirect . Routing can be defined declarative using Route tags or via a config object passed in the Router component. IndexRoute nested within a Route specifies the default nested component to mount (i.e. Home in the example below)



The route accepts onLeave and onEnter hooks that can be used to complete tasks, such as authentication or data persistence, before the the route unmounts or mounts .



For maintaining browser history, bring in browserHistory or hashHistory . For cleaner urls, [it is recommended to use browserHistory , however, this requires server configuration for](https://github.com/reactjs/react-router/blob/master/docs/guides/Histories.md) [production.](https://github.com/reactjs/react-router/blob/master/docs/guides/Histories.md)



import { Router, Route, browserHistory, IndexRoute } from 'react-router'

... // import components and React dependencies

* declarative definition render((

<Router history={browserHistory}> <Route path="/" component={App}> <IndexRoute component={Home} />

<Route path="about" component={About} /> <Route path="/products" component={Products}>

<Route path="products/:id" component={Product} /> </Route>

</Route>

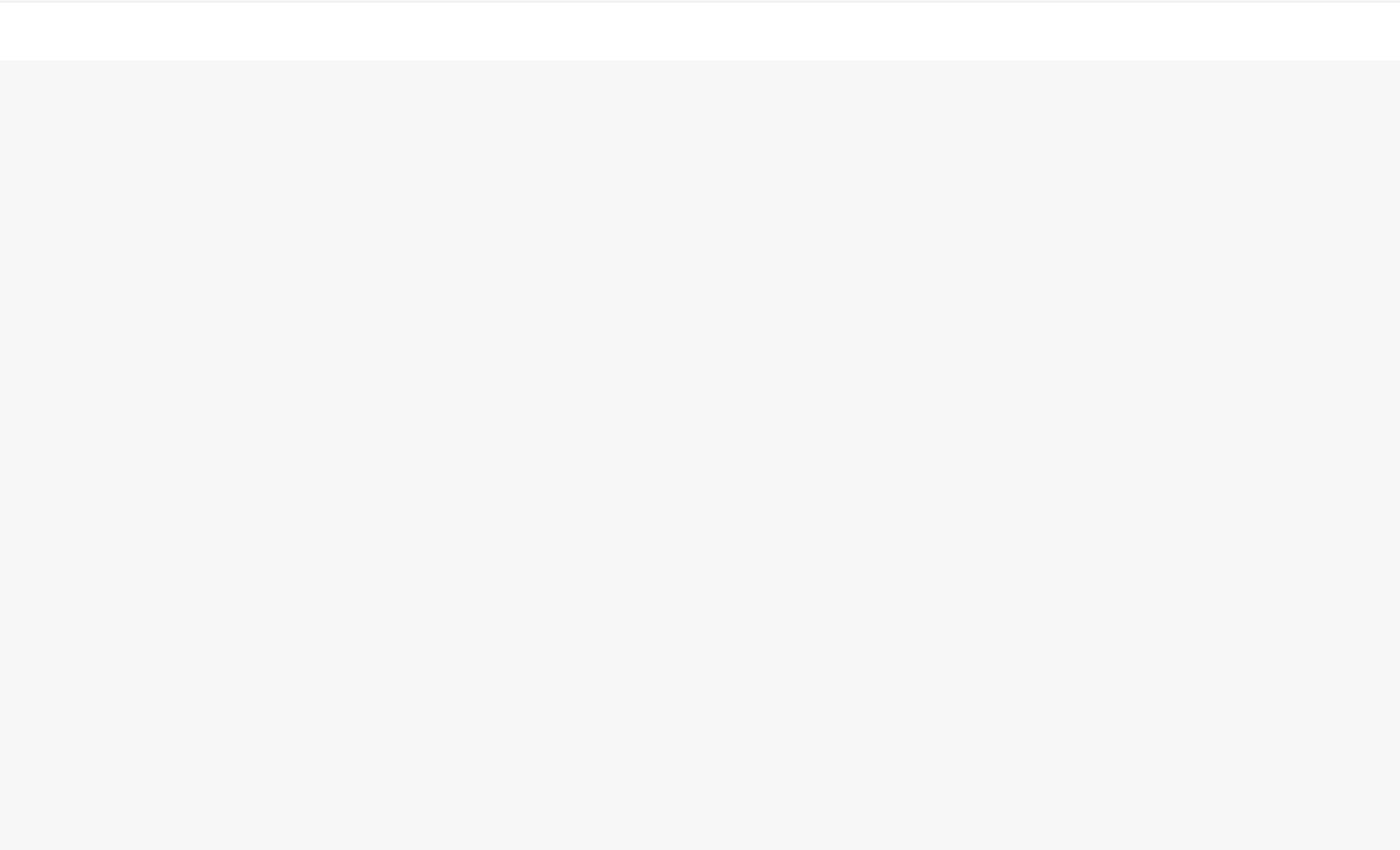
</Router>

), document.body)



98

React Router



* configuration definition const routes = {

path: '/',

component: App,

indexRoute: { component: Home },

childRoutes: [

{ path: 'about', component: About },

{

component: Product,

childRoutes: [{

path: 'product/:id', component: Product

}]

}

]

}

render(<Router routes={routes} />, document.body)

**Params, Nested Components, Links**

Any url (i.e. product/:id ) parameters can be accessed from the rendered component via

this.props.params . Use this.props.children to render nested components. A simple example is maintaining the navigation across the app where the pages are rendered below the navigation.

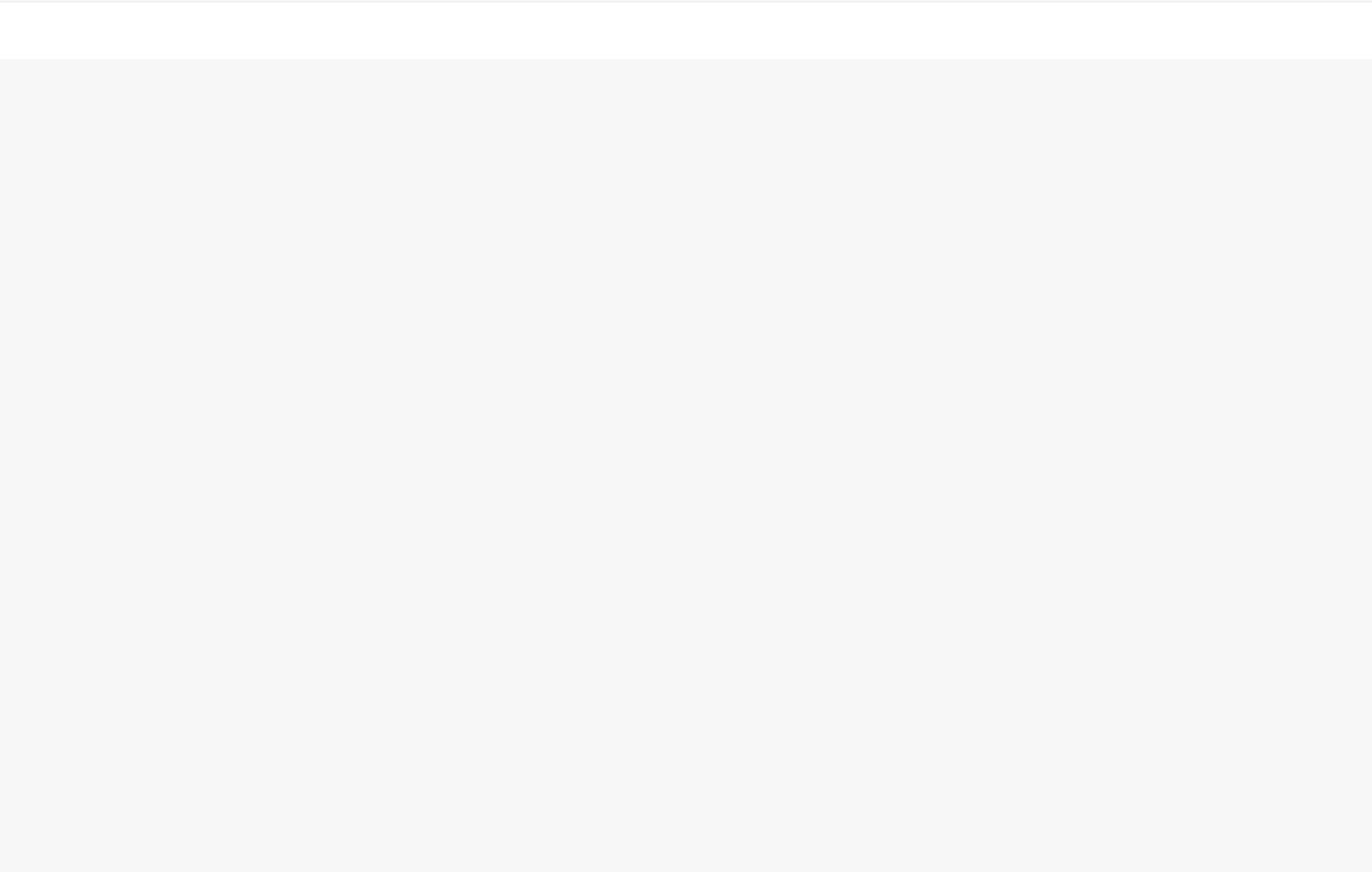


The link component is to create links that use the router configuration (like <a/> tags). To determine the active link, you can pass activeClassName or activeStyle props to link . git st



99

React Router



import { IndexLink, Link } from 'react-router'

... // import components and React dependencies

class App extends Component {

render() {

return (

<div>

<h1>App</h1>

<ul>

<li><IndexLink to="/" activeStyle={{ color: green }}>Home</Link></li> <li><Link to="/about" activeClassName="active">About</Link></li>

</ul>

{/\* nested components, i.e. <Home/>, rendered below \*/}

{ this.props.children } </div>

)

}

}



100

Router Redux

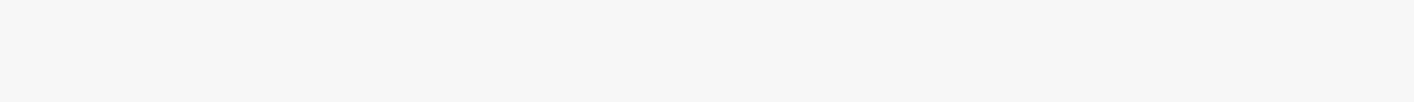


**Redux and React Router**

React Router and Redux work fine together, however, debugging features such as replaying actions with [Redux DevTools](https://github.com/gaearon/redux-devtools) will not work.

Since React Router is controlling the components being rendered via the url, we need to store the history within the application state. We can use [react-router-redux](https://github.com/reactjs/react-router-redux) to do this.

**Setup**



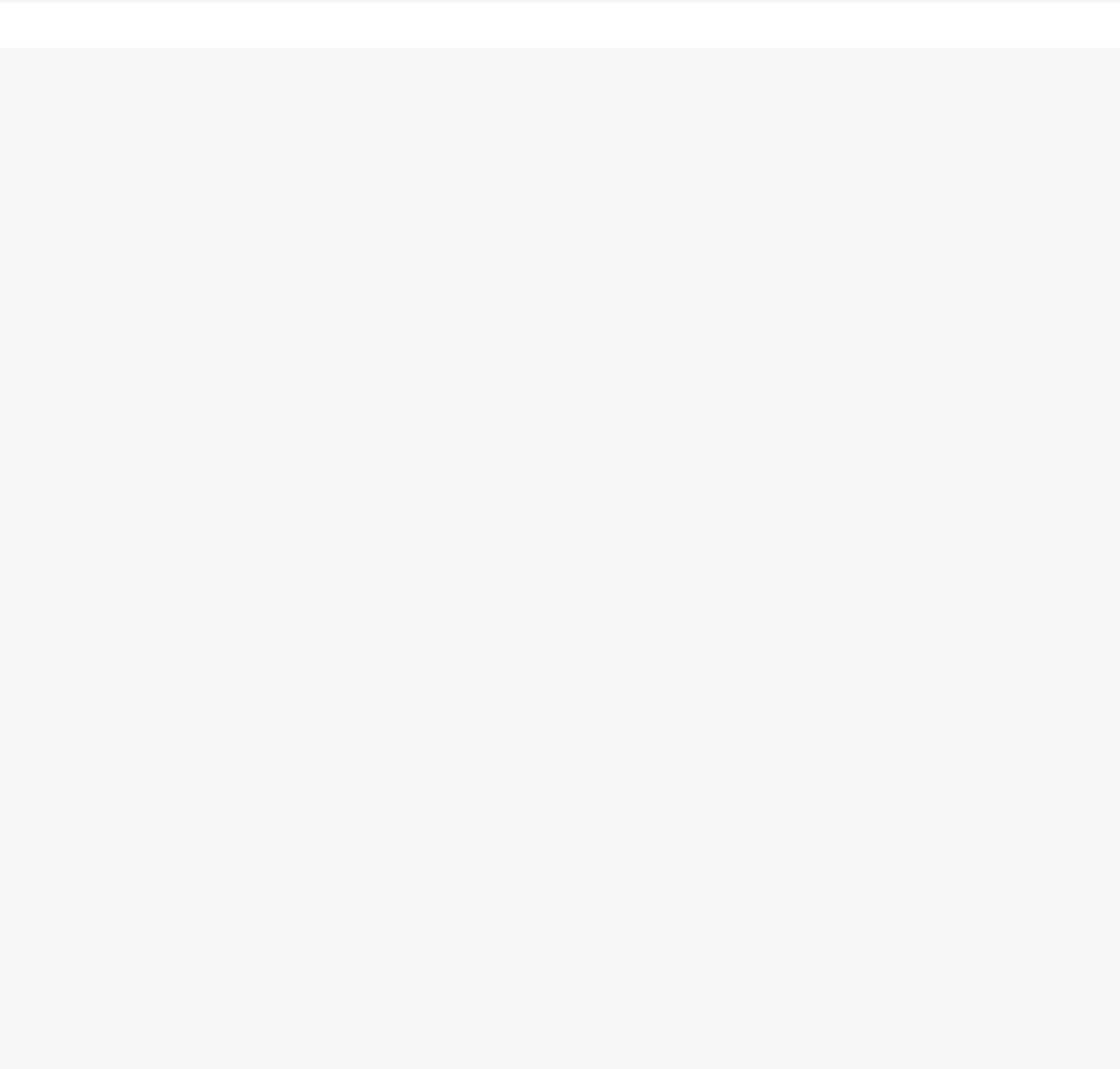
npm install --save react-router-redux

**Example**



101

Router Redux



import { createStore, combindReducers } from 'redux' import { Provider } from 'react-redux'

import { Router, Route, browserHistory, IndexRoute } from 'react-router' import { syncHistoryWithStore, routerReducer } from 'react-router-redux'

... // import components, reducers and React dependencies

const store = createStore(

combineReducers({

...reducers,

* add routerReducer on `routing` key routing: routerReducer

})

)

// sync the history with the store

const history = syncHistoryWithStore(browserHistory, store);

render((

<Provider store={store}>

<Router history={history}>

<Route path="/" component={App}>

<IndexRoute component={Home} />

<Route path="about" component={About} />

<Route path="/products" component={Products}>

<Route path="products/:id" component={Product} /> </Route>

</Route>

</Router>

</Provider>

), document.body)



102

Forms



**Forms in React**

This section will look at setting up [Redux Form](http://redux-form.com/).

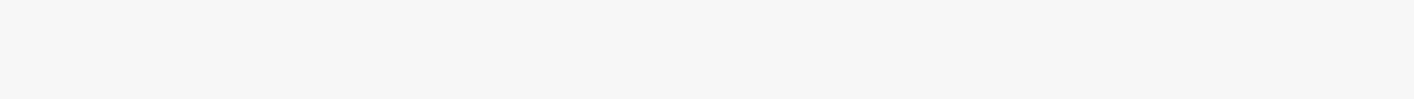


103

Redux Form



**Setup**



npm install redux-form

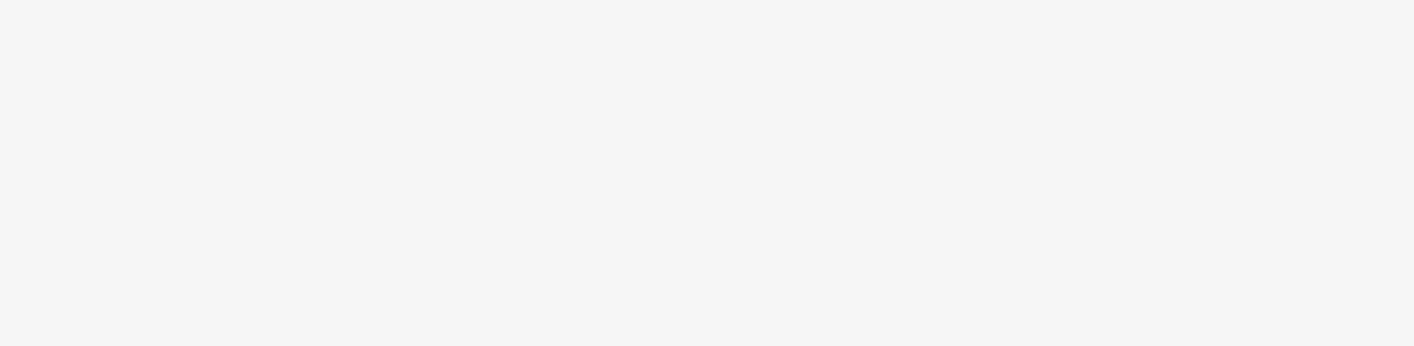
**Reducer**

Redux-form provides a reducer that manages the form's state. Add this to the

combineReducers . **It is important to specify the state reducer key as** **form** **for redux-**



**form to work**



import {reducer as formReducer} from 'redux-form'

const reducer = combineReducers(Object.assign({}, reducers, {

...

routing: routeReducer,

form: formReducer,

}))

**Wrapper**

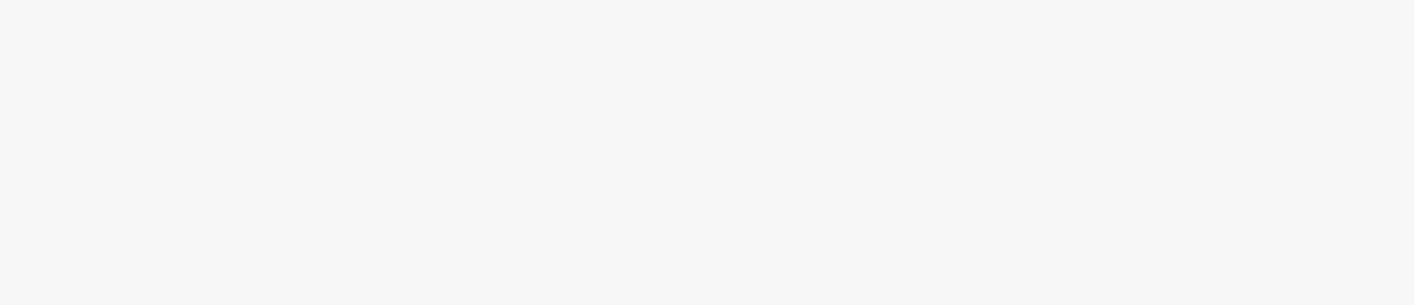
Redux-form provides a redux-form wrapper to pass your component props as callbacks



* resetForm , handleSubmit ) and form data ( error , dirty , fields , submitting ). View



the [full list of props](http://redux-form.com/5.2.5/#/api/props).



export default reduxForm({

form: 'formKey', // form key must be unique

fields: ['name', 'email'] // form fields

...

validate // custom form validation function

})(Form)

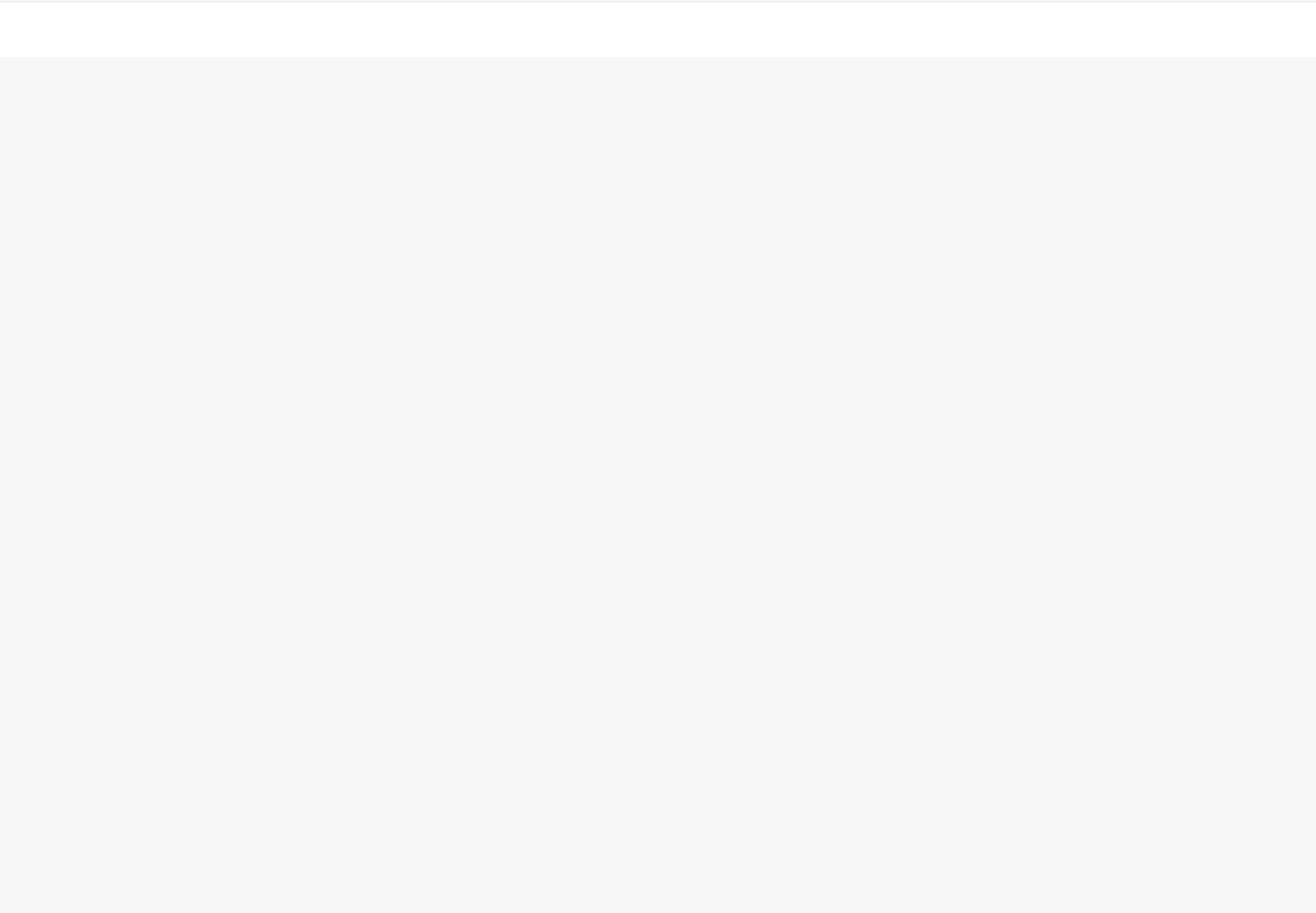
**Form**

The fields props contains the field values (i.e. name, email) and several event listeners for each field, so these must be passed into the input tag for the specific field via {...name} .



104

Redux Form



import React, {Component} from 'react';

import {reduxForm} from 'redux-form';

const submit = (formValues, dispatch) => {

...

}

class Form extends Component {

render() {

const {fields: {name, email}, handleSubmit} = this.props; return (

<form onSubmit={handleSubmit(submit)}>

<label>First Name</label>

<input type="text" placeholder="Name" {...name}/> <label>Last Name</label>

<input type="text" placeholder="Email" {...email}/> <button type="submit">Submit</button>

</form>

)

}

}

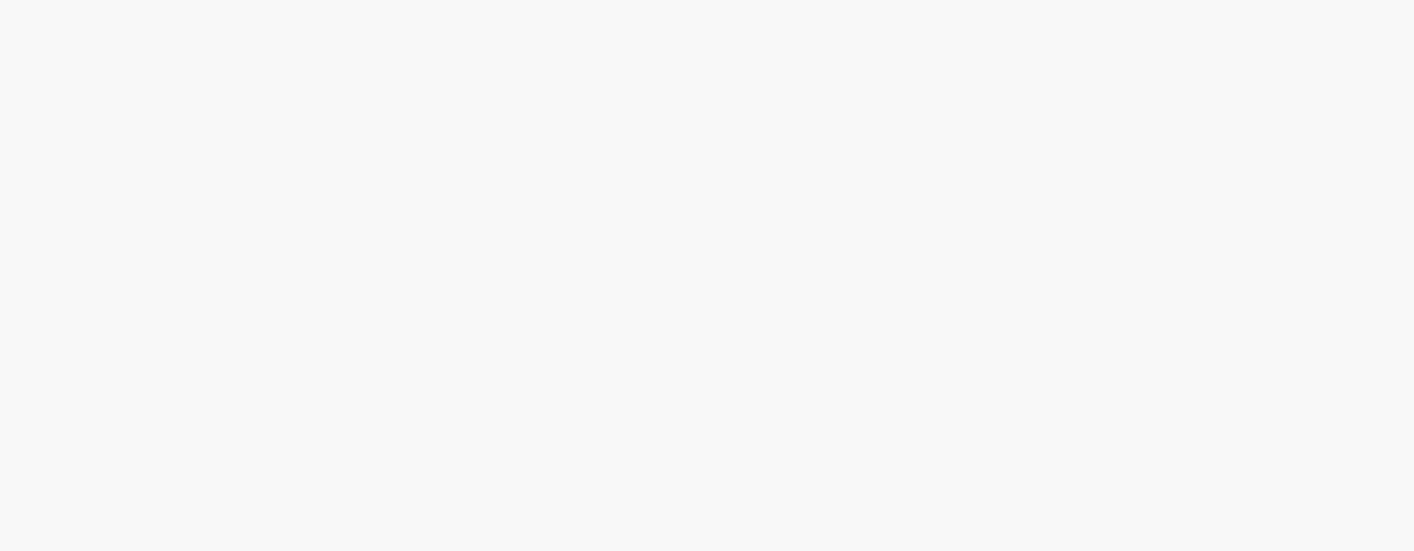
For submission of form data, the handleSubmit prop is passed to the onSubmit or onClick and will complete any validations before calling this.props.onSubmit(data) . You can also pass a function into handleSubmit to be called, so handleSubmit(submit) .



Check out [Redux Form](http://redux-form.com/5.2.5/#/examples?_k=f0v1eh) for more form examples.

**Validation**

The validate function will be called on each render cycle and will be passed the a form values object where the function must return an errors object with the specific error message for each field.



const validate = fields => {

const errors = {}

if (!fields.name) {

errors.name = 'Required'

}

if (!fields.email) {

errors.email = 'Required'

} else if (!/^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/i.test(fields.email)) { errors.email = 'Invalid email address'

}

return errors

}



105

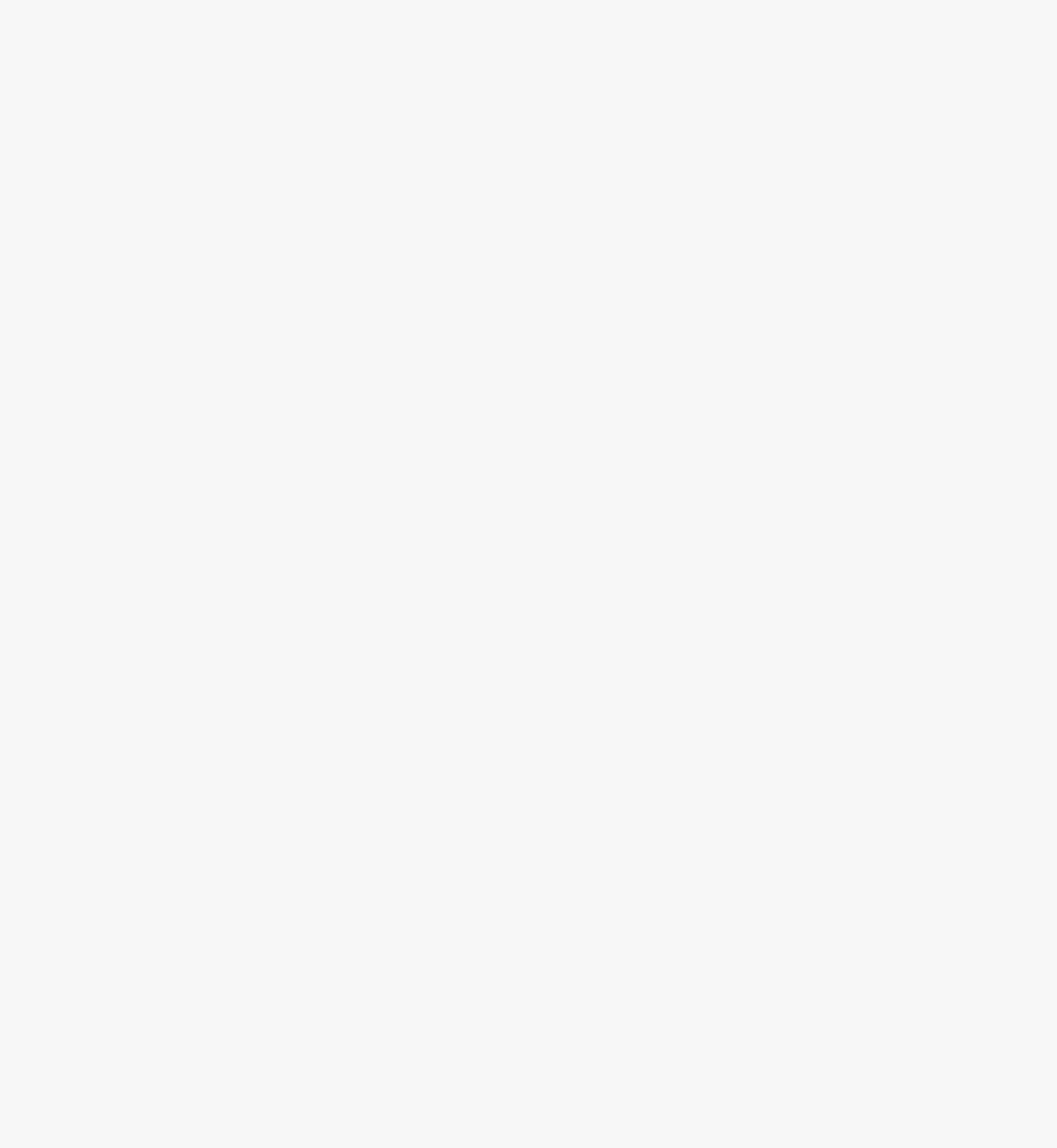
Redux Form



The keys for the input field values and output errors objects must match the form fields specified (i.e. name , and email ).



**Full Example**



import React, {Component} from 'react';

const validate = values => {

const errors = {};

if (!values.name) {

errors.name = 'Required';

}

if (!values.email) {

errors.email = 'Required';

} else if (!/^[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i.test(values.email)) {

errors.email = 'Invalid email';

}

return errors;

}

class Form extends Component {

render() {

const {fields: {name, email}, handleSubmit} = this.props; return (

<form onSubmit={handleSubmit(submit)}>

<label>First Name</label>

<input type="text" placeholder="Name" {...name}/> <label>Last Name</label>

<input type="text" placeholder="Email" {...email}/> <button type="submit">Submit</button>

</form>

)

}

}

export default reduxForm({

form: 'formKey', // form key must be unique

fields: ['name', 'email'] // form fields

...

validate // custom form validation function })(Form)



106

Testing



**Testing**

We will be using [Enzyme](https://github.com/airbnb/enzyme) to render and test React components.



107

.find

Setup



**Setup**

To get started, we need to do a little bit of setup. Install the following dependencies.

babel-eslint



babel-preset-es2015



babel-preset-react



babel-preset-stage-0



babel-register



chai



enzyme



jsdom



mock-localstorage



react-addons-test-utils



redux-mock-store



sinon



mocha



nock



**Enzyme**

[Enzyme](https://github.com/airbnb/enzyme) allows for rendering of React Components in three different ways: render (static component), shallow (isolated component), mount (full DOM). Most components can be shallow rendered and tested in isolation. Enzyme allows for searching of the rendered



component for testing (i.e. ).

We will be using Mocha as the test runner. View the [API Documentation](http://airbnb.io/enzyme/) for further details.

**JSDOM**

When using Enzyme to render component via mount (full DOM), you can use [jsdom](https://github.com/airbnb/enzyme/blob/master/docs/guides/jsdom.md) as a headless browser since mount requires a DOM environment.



108

Components



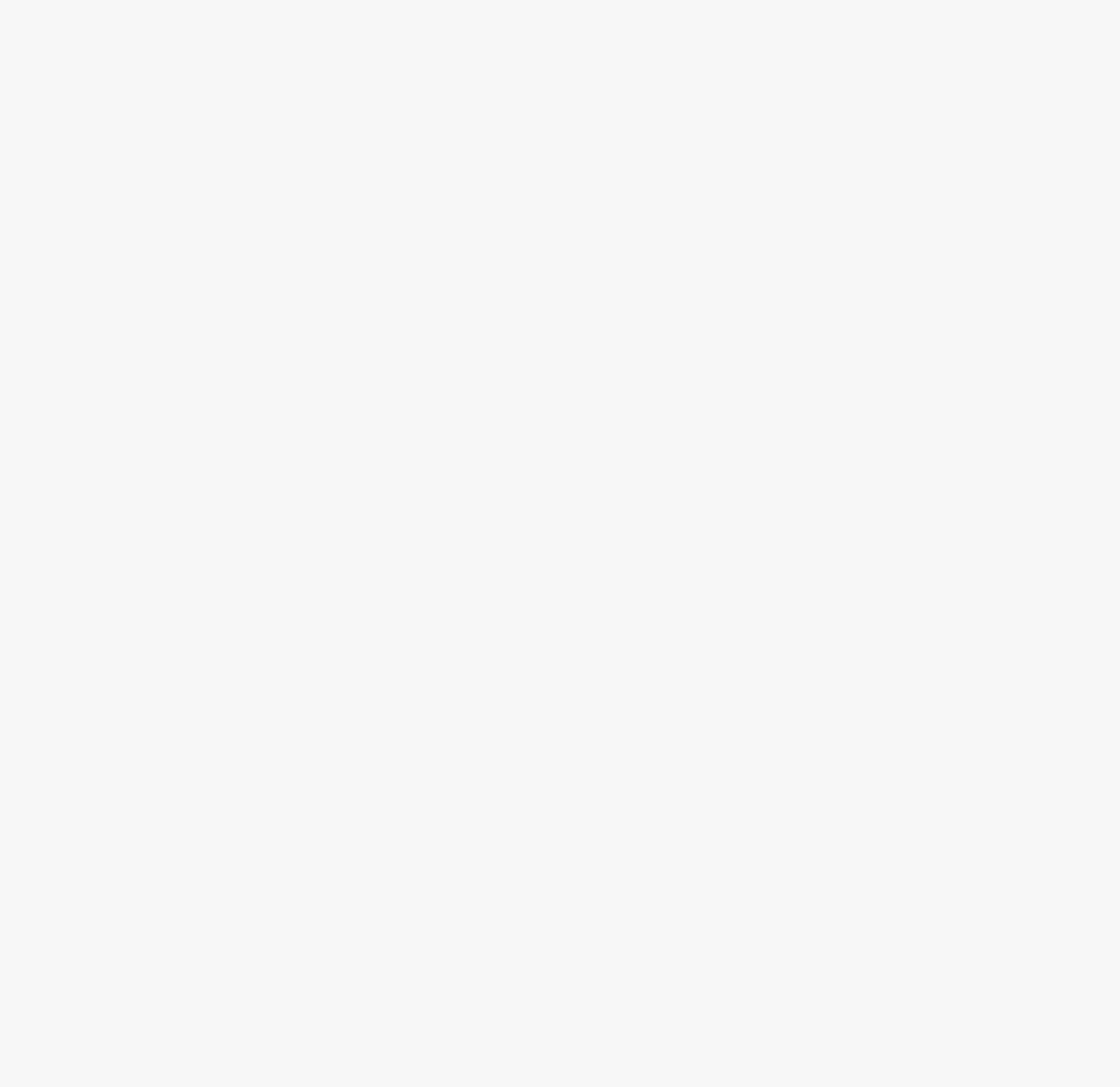
**Components**

Enzyme is used to output React components and manipulate or transverse them. Using the

chai assertion library, we can make assertions on the component.



**Example**



* ./counter/index.js import React from 'react'; import Button from '../button';

function Counter({ counter, increment, decrement, ...props }) { return (

<div className="flex" data-testid={ props.testid }>

<Button data-ref="decrementButton" className="bg-black col-2"

onClick={ decrement }>

-

</Button>

<div data-ref="result" className="flex-auto center h1">

{ counter } </div>

<Button data-ref="incrementButton" className="col-2"

onClick={ increment }>

+

</Button>

</div>

);

}

Counter.propTypes = {

counter: React.PropTypes.number,

increment: React.PropTypes.func,

decrement: React.PropTypes.func,

testid: React.PropTypes.func,

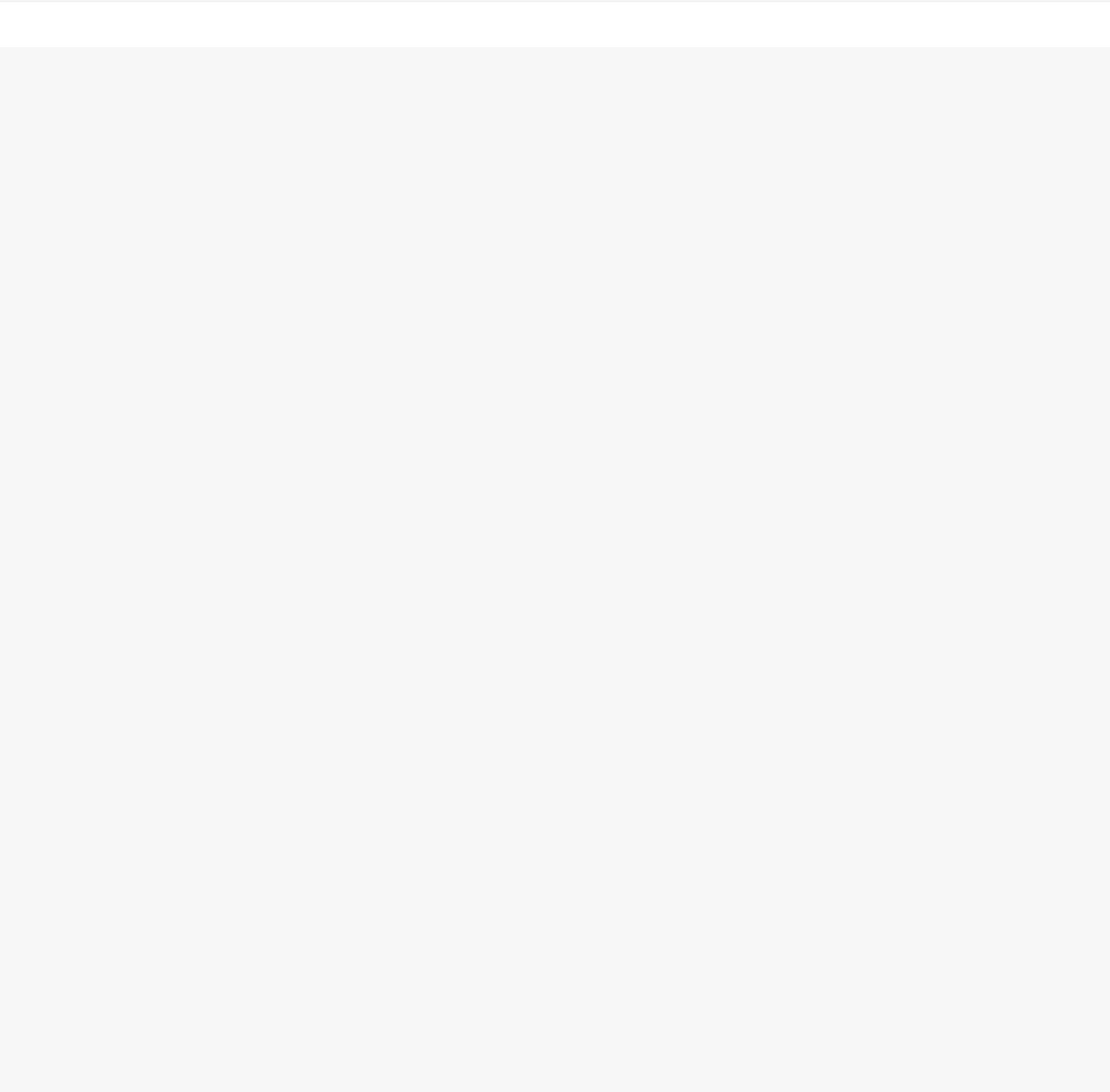
};

export default Counter;



109

Components



* ./counter/index.test.js import { assert } from 'chai'; import React from 'react';

import { shallow, render } from 'enzyme'; import sinon from 'sinon';

import Counter from './index';

describe('counter', () => {

it('should create a counter', () => {

const wrapper = render(<Counter counter={5} />);

assert.isOk(wrapper.children().length,

'Counter not found');

assert.strictEqual(wrapper.find('[data-ref="result"]').text(), '5', 'Counter not showing its value');

});

it('should respond to click events', () => {

const onIncrement = sinon.spy();

const onDecrement = sinon.spy();

const wrapper = shallow(

<Counter increment={onIncrement} decrement={onDecrement} /> );

wrapper.find('[data-ref="incrementButton"]').simulate('click'); assert.isTrue(onIncrement.calledOnce, 'increment not called');

wrapper.find('[data-ref="decrementButton"]').simulate('click'); assert.isTrue(onIncrement.calledOnce, 'decrement not called');

});

});



110

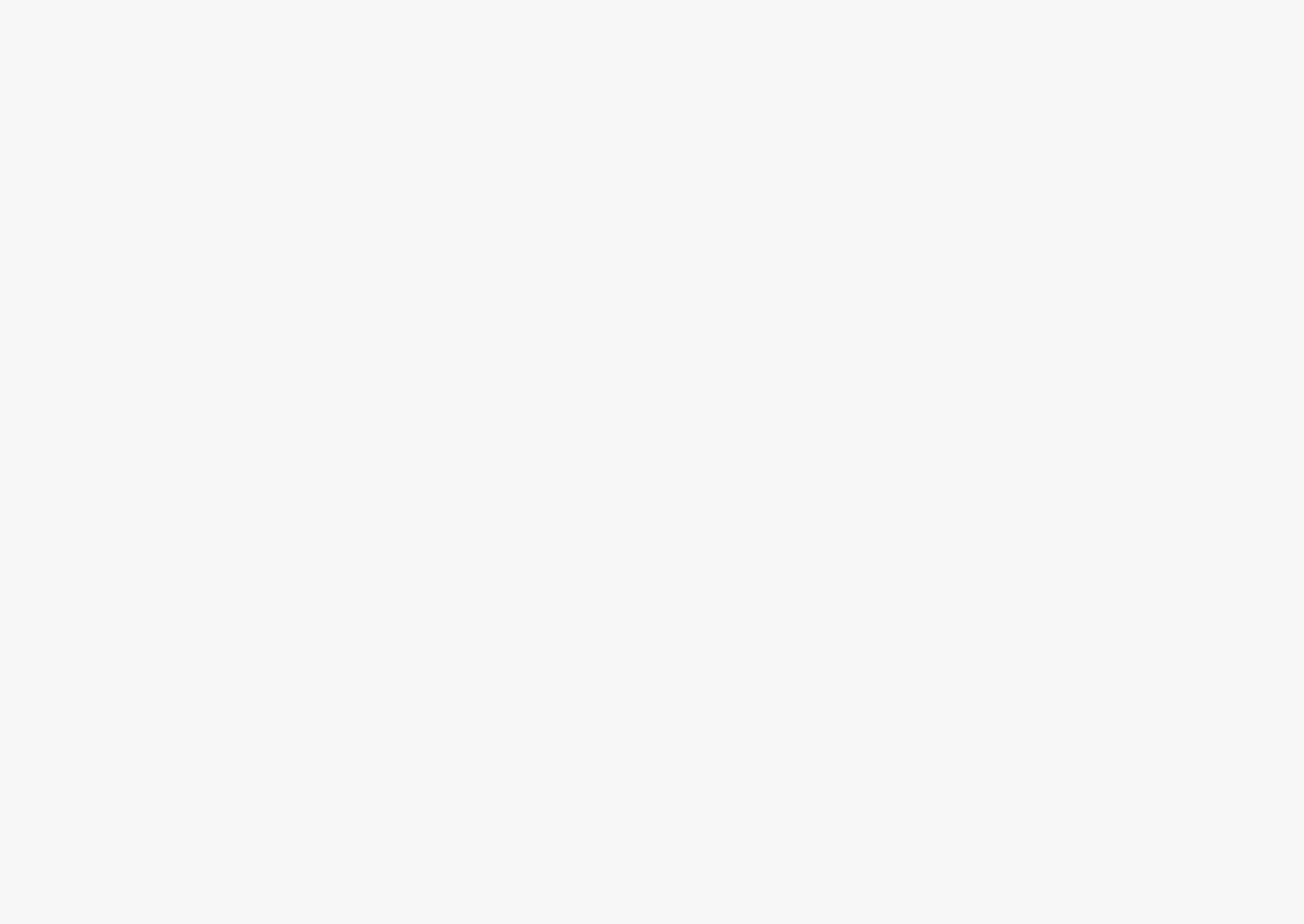
Reducers



**Reducers**

Since reducers are pure functions, they can be tested like a JavaScript function (i.e. given an action, confirm the expected result)

**Example**



// ./reducers/counter.js

import { INCREMENT\_COUNTER, DECREMENT\_COUNTER } from '../constants'; import { fromJS } from 'immutable';

const INITIAL\_STATE = fromJS({

count: 0,

});

function counterReducer(state = INITIAL\_STATE, action = {}) { switch (action.type) {

case INCREMENT\_COUNTER:

return state.update('count', (value) => value + 1);

case DECREMENT\_COUNTER:

return state.update('count', (value) => value - 1);

default:

return state;

}

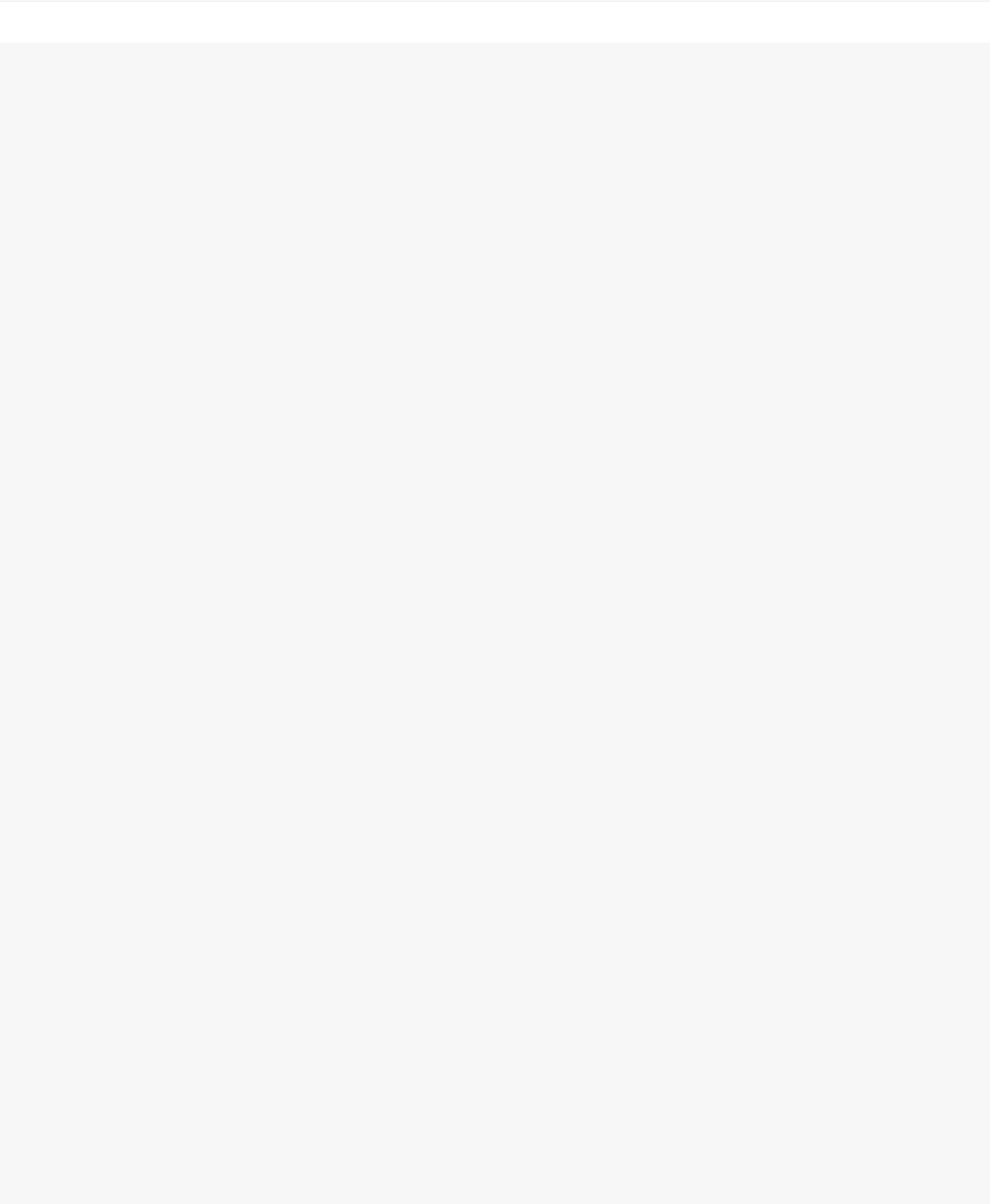
}

export default counterReducer;



111

Reducers



* ./reducers/counter.test.js import { assert } from 'chai';

import counterReducer from './counter';

import { INCREMENT\_COUNTER, DECREMENT\_COUNTER } from '../constants'; import { Map } from 'immutable';

//Used for testing to fire actions against a reducer.

function fireAction(reducer, currentState, type, payload = {}) { return reducer(currentState, {

type,

payload,

});

}

let state = counterReducer(undefined, {});

describe('counter reducer', () => {

describe('inital state', () => {

it('should be a Map', () => {

assert.strictEqual(Map.isMap(state), true);

});

});

describe('on INCREMENT\_COUNTER', () => {

it('should increment state.count', () => {

const previousValue = state.get('count');

state = fireAction(counterReducer, state, INCREMENT\_COUNTER); assert.strictEqual(state.get('count'), previousValue + 1);

});

});

describe('on DECREMENT\_COUNTER', () => {

it('should decrement state.count', () => {

const previousValue = state.get('count');

state = fireAction(counterReducer, state, DECREMENT\_COUNTER); assert.strictEqual(state.get('count'), previousValue - 1);

});

});

});



112

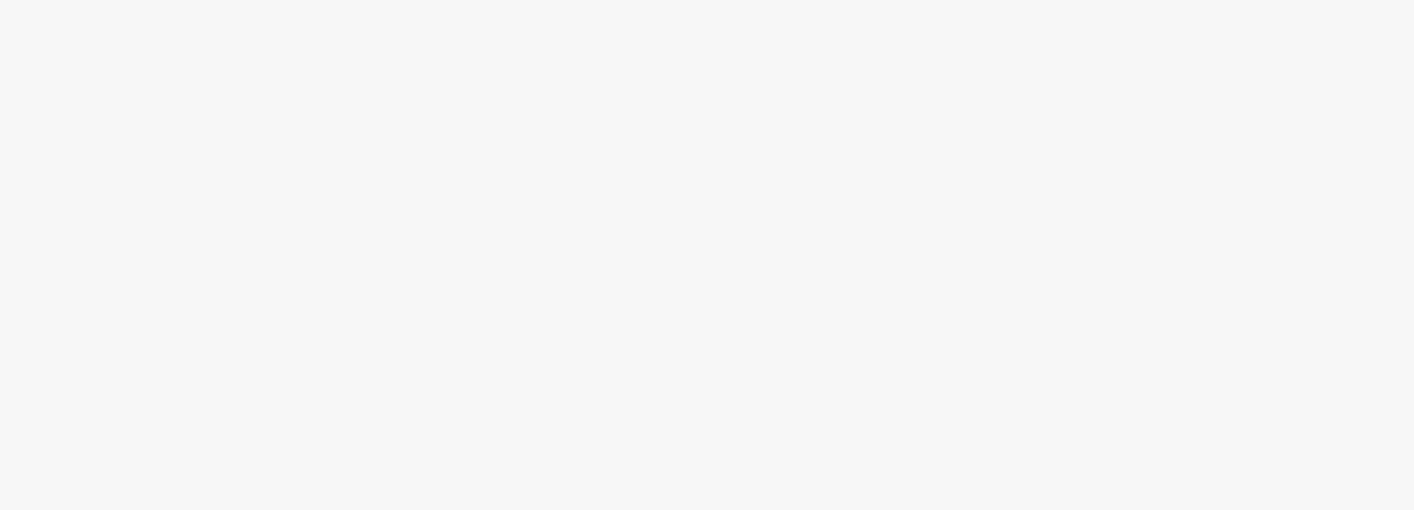
Actions



**Actions**

Simple actions can be tested like reducers, as pure functions (i.e. given params, confirm the expected object/response). The [Redux gitbook](http://redux.js.org/docs/recipes/WritingTests.html#async-action-creators) contains more examples, including handling tricky async actions.

**Example**



// ./actions/counter.js

import { INCREMENT\_COUNTER, DECREMENT\_COUNTER } from '../constants';

export function increment() {

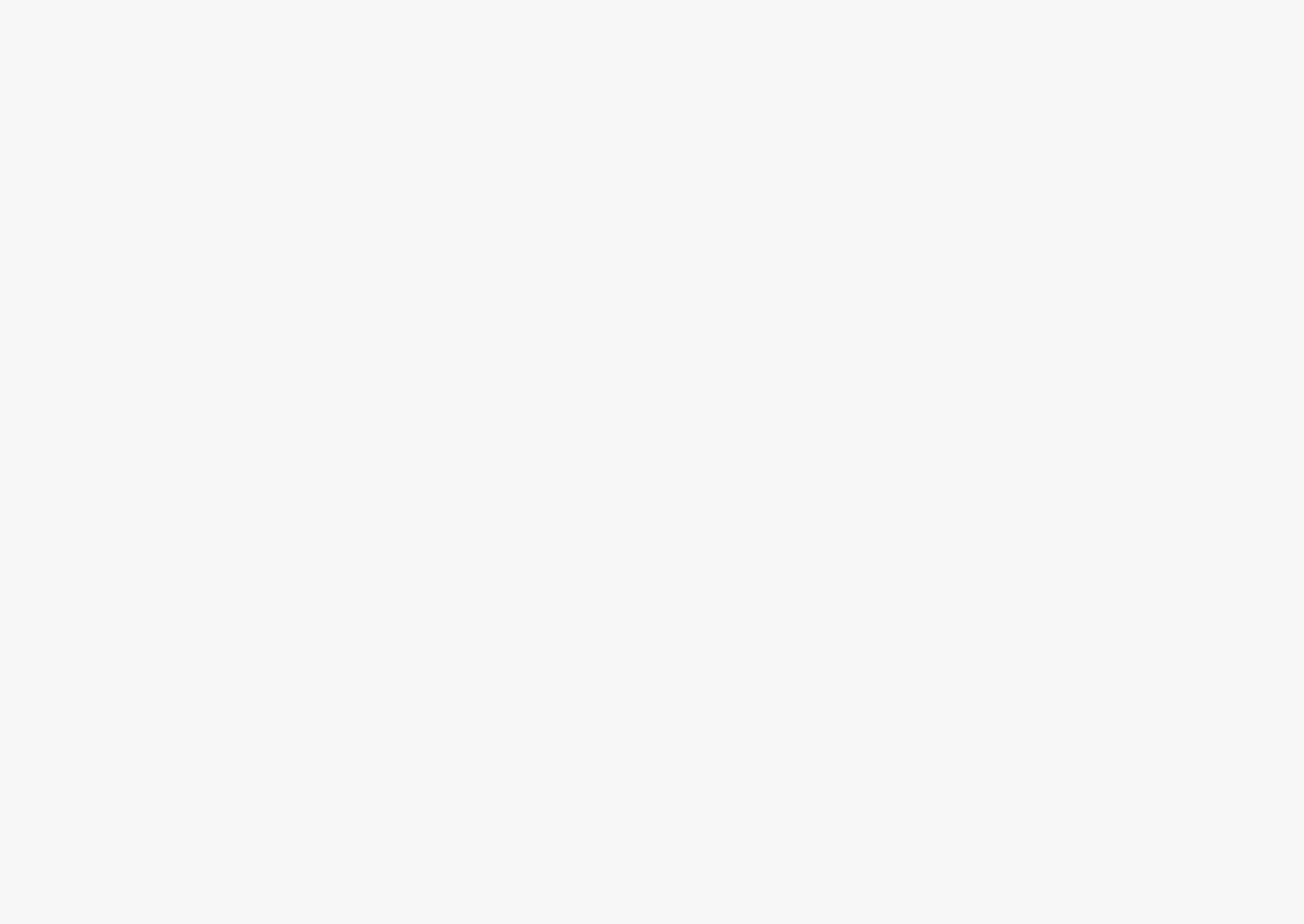
return {

type: INCREMENT\_COUNTER,

};

}

...



// ./actions/counter.test.js

import { assert } from 'chai';

import { INCREMENT\_COUNTER, DECREMENT\_COUNTER } from '../constants'; import { increment } from '../actions/counter';

describe('Counter Actions', () => {

describe('increment', () => {

let obj;

beforeEach(() => {

obj = increment();

});

it('should return an object', () => {

assert.isObject(obj)

});

it('should return an object with correct type property', () => { assert.deepEqual(obj, {type: INCREMENT\_COUNTER});

});

})

});



113

Actions



114