POLYMER 2
CLASE 2

Restricciones

Nombre del documento:
Clasificación de la Información: Interno
Restricciones

› Los contenidos de este documento son propiedad de Softtek y son internos. Queda estrictamente prohibido cualquier reproducción total o parcial sin la autorización escrita por parte de Softtek.

› Este documento está sujeto a cambios. Los comentarios, correcciones o dudas deberán ser enviados al autor.

Audiencia	Propósito
Desarroladores	Capacitación

Tabla de Revisión

› La siguiente tabla enlista las revisiones realizadas a este documento. Debe utilizarse para describir los cambios y adiciones cada vez que este documento vuelva a ser publicado. La descripción debe ser detallada e incluir el nombre de quien solicita los cambios.

Núm. de versión	Fecha de versión	Tipo de cambios	Dueño / Autor	Fecha de revisión / Expiración
1.1	5/10/2018	Creación del documento	Carlos Montero	12/10/2018

Notify

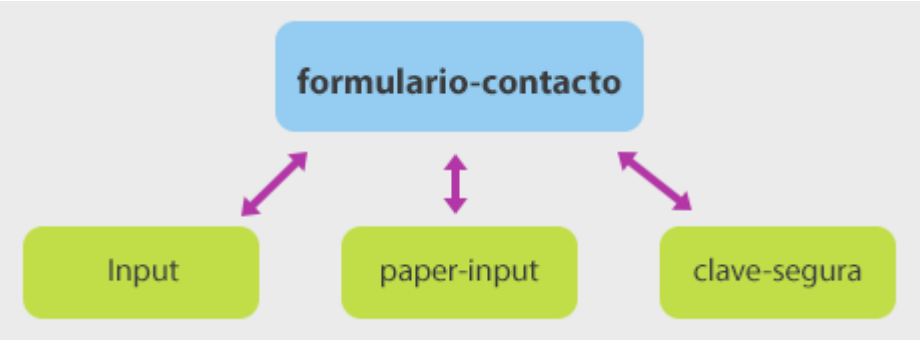
Cuándo usar notify

Notify es una configuración que no se debería usar, a no ser que realmente sea necesaria. Si el componente no necesita comunicarse con el padre para nada, entonces no tiene sentido usar notify. Solo en el caso que el componente necesite comunicar con el padre tendría sentido configurar alguna de sus propiedades con notify a true, pero aún así no siempre será la mejor opción.

Imagina un componente que es un banner por el que rotan varias imágenes, o unos textos que parpadean alternando el mensaje que se visualiza. Ese banner lo colocas en la página, para adornar y quizás no necesita decirle nada al padre. Va cambiando y punto. Si se hace clic sobre el banner se podría accionar un enlace, usando una simple etiqueta A, pero no se necesita avisar al padre de nada en concreto. Entonces, obviamente, no tendrá sentido usar notify.

Ahora piensa en un componente algo diferente, por ejemplo un campo de formulario para escribir una clave, que te indica la fortaleza de esa clave. Ese campo de formulario lo puedes hacer mediante un componente que podría llamarse "clave-segura". Al usar ese componente comúnmente se colocará dentro de un formulario, que puede tener otra serie de campos creando una jerarquía de elementos como podría ser la que tienes en la imagen:

Notify



En este esquema tenemos un componente, que es un formulario de contacto completo. Ese componente tiene a su vez varios hijos, que podrían ser campos INPUT (el elemento caja de texto nativo del HTML), campos paper-input (una caja de texto con el estilo de Material Design) o aquel componente que nos referíamos antes que permite escribir una clave y te informa sobre su fortaleza.

Notify

Sintaxis:

```
static get properties() {
  return {
    datoNotificable: {
      type: String,
      notify: true
    },
  };
}
```


Observers

Softtek®

Polymer contiene un juego muy potente de herramientas para trabajo con propiedades declaradas. De entre todas ellas, hoy vamos a hablar de los “Observers”.

Nos centraremos entonces en cómo observar sus cambios por medio de los mencionados observers.

Solo por aclarar, para quien venga de AngularJS, los observers vienen a ser como los “watcher”, pero algo más sencillos de manejar. Seguramente otras librerías o frameworks ofrecen herramientas semejantes, dado que son muy útiles en proyectos de desarrollo.

Básicamente porque nos permiten centralizar en un único sitio todas las cosas que deben ocurrir cuando hay un cambio en un dato. Cambie por el motivo que cambie ese dato (en Polymer será una propiedad), se invocará al correspondiente observer cuando esto ocurra, si es que se ha definido alguno.

Observers

Existen dos maneras de trabajar con los observers. Una muy sencilla en la declaración de la propiedad y otra un poco más compleja por medio de un array “observers” donde podemos declarar todos los observadores que queramos. La manera sencilla la podemos usar siempre que lo que queramos observar una única propiedad. La manera compleja la usaremos cuando queramos observar dos o más propiedades a la vez.

Observer en una propiedad de un componente

En una propiedad de un componente Polymer podemos agregar el atributo “observer”, asignando como valor el nombre de una función. Así estamos creando un observer básico (de los sencillos), que se ejecutará cada vez que dicha propiedad cambie de valor.

Las funciones que asignas como observadores las podrías entender como una especie de manejador de evento asociado al valor de la propiedad. Es solo un símil a nivel didáctico, porque un observer no es un evento, pero podrías decir que es algo parecido a un evento que se dispara cuando cambia un valor, llamando a la función observadora que se haya declarado.

Su uso es muy elemental, como verás a continuación. Vamos a ver primeramente la declaración de una propiedad donde hemos definido un observer.

Observers

```
cuenta: {
  type: Number,
  value: 10,
  observer: "vigilarFinCuenta"
}
```

Ahora veamos el código de nuestra función observadora, que no es más que un método del propio componente, que recibe como ves dos parámetros.

```
vigilarFinCuenta: function(valorActual, valorAnterior) {
  console.log(valorActual, valorAnterior);
  if(valorActual == 0){
    console.log(“llegamos al final de la cuenta”)
  }
}
```


Observers

Cuando cambia la propiedad se envía a la función como parámetro el nuevo valor, así como el valor que anteriormente hubiera en esa propiedad, por si nos hiciera falta para algo.

Nota: Es interesante mencionar que, en el código de la función anterior, this.cuenta sería equivalente al parámetro valorActual. Podremos acceder a uno u otro indistintamente para acceder al valor actual de la propiedad.

Declaración de observadores mediante el array "observers“

El array de “observers” (fíjate que en este caso está en plural) nos permite observar una o más propiedades. Es el mecanismo sofisticado para definir observadores, que complementa al mecanismo explicado en el punto anterior.

Se debe indicar en el array, también como cadena, cada uno de los nombres de las funciones observadoras, con el juego de parámetros equivalente a los nombres de propiedades que se están observando.

Observers

Las funciones observadoras son capaces de recibir los nuevos valores de las propiedades, una vez producido el cambio observado. Pero a diferencia Igual del método anterior de definición de observers (aquellas funciones observadoras que hemos calificado de simples, declaradas en en el objeto “properties”), en este caso no se nos informa del valor anterior de la propiedad.

Este tipo de observer, declarado en el array "observers", tiene otra característica que lo diferencia y es que la funcion observadora no se ejecutará hasta que el valor de ambas propiedades sea distinto de null o undefined. Dicho de otro modo, el observer no empieza a estar operativo hasta que ambas propiedades tengan un valor definido.

Así se define el array de observers.

```
observers: ['cambiaTask(task)', 'cambiaTaskBoom(task, boom)'],
```


Observers

Luego nos quedaría definir estas funciones observadoras, como métodos en el componente. Es similar a lo que ya hemos conocido,

```
cambiaTask: function(task) {
  console.log("cambiaTask", task, k);
},
cambiaTaskBoom(task, boom) {
  console.log("cambiaTaskBoom", task, boom);
}
```

Como te habrás dado cuenta, en el método de definición de observers no tenemos acceso al valor anterior de la propiedad. Solo nos facilitan como parámetro el valor actual.

Observers ejemplo:

```
<link rel="import" href="../../bower_components/polymer/polymer.html">
<link rel="import" href="../../bower_components/paper-button/paper-button.html">
<dom-module id="cuenta-atras">
  <template>
    <div class="contador" hidden="{{boom}}">{{cuenta}}</div>
    <div class="boom" hidden="{{!boom}}">BOOOOOM!!!</div>
    <p>
      <paper-button on-tap="disparar" hidden="{{boom}}">Dispara</paper-button>
    </p>
  </template>
  <script>
   Polymer({
     is: 'cuenta-atras',
     properties: {
       cuenta: {
         type: Number,
         value: 10,
         observer: "vigilarFinCuenta"
       },
       task: Number,
        boom: {
         type: Boolean,
         value: false
       }
     },

     observers: ['cambiaTask(task)', 'cambiaTaskBoom(task, boom)'],
```

```
cambiaTask: function(task) {
      console.log("cambiaTask", task);
    },
     cambiaTaskBoom(task, boom) {
      console.log("cambiaTaskBoom", task, boom);
    },
    vigilarFinCuenta: function(valorActual, valorAnterior) {
      console.log(valorActual, valorAnterior);
      if(this.cuenta == 0){
        this.cancelAsync(this.task);
        this.boom = true;
      }
    },
    decrementar: function() {
      this.task = this.async(this.decrementar, 1000);
      this.cuenta --;
    },
    ready: function() {
      this.task = this.async(this.decrementar, 1000);
    },
    disparar: function() {
      this.cuenta = 0;
    }
   });
  </script>
</dom-module>
```


Observers ejemplo 2:

Softtek®

Ejemplo de Observers múltiples con comodines.

```
calificacion: {
        type: Object,
        value: function() {
          return {
            matematicas: 5,
            lenguaje: 5,
            ciencias: 5
          };
        }
      }
    };
  }

  static get observers() {
    return [
      'cambiosCalificacion(calificacion.*)'
    ];
  }
```

```
cambiosCalificacion(changeRecord) {
      let suma = 0;
      let materias = 0;
      let estaAprobado = true;
      for (let i in this.calificacion) {
        materias++;
        let calificacionActual = parseInt(this.calificacion[i]);
        suma += calificacionActual;
        if(calificacionActual < 5) {
          estaAprobado = false;
        }
      }
      this.media = suma / materias;
      this.media = Math.round(this.media * 10) / 10;
      this.aprobado = estaAprobado;
    }
    constructor() {
      super();
    }
  }
  window.customElements.define(CalificacionesEstudiante.is,
CalificacionesEstudiante);
  </script>
</dom-module>
```


Ejemplo notify y observers

Genera el ejemplo de notify y observers adaptando los siguientes componentes en tu equipo, y te sirva para tu biblioteca de código.

index.html

web component 1

web component 2

EVENTOS PERSONALIZADOS BUBBLES Y COMPOSED.

Softtek®

Para entender correctamente que son los eventos personalizados, deberíamos observar los eventos de javascript que ya conocemos, como mousemove, keypress, keydown, etc…

En cualquiera de ellos cuando se lanza el evento se ejecuta una función asociada y ejecuta su código.

En polymer podemos usar todos los eventos de javascript nativo, pero puede ser que necesitemos algún evento que no este definido en javascript.

Por ejemplo, imaginemos una caja de texto, y la vamos a usar para realizar una llamada a un API REST, pero necesitamos que esa llamada solo se realice cuando pulsemos dos veces la tecla enter.

Como es de suponer en javascript no existe un evento ‘cuando pulses una tecla dos veces y que además se la tecla enter ejecuta este código‘.

En el ejemplo anterior hemos creado un evento personalizado que se llama tecla, y que su función es capturar la tecla pulsada.

EVENTOS PERSONALIZADOS BUBBLES Y COMPOSED.

Softtek®

Cuando se ejecutan los eventos en javaScript, estas llamadas se hacen en forma de burbuja desde el elemento más interno hasta el más externo.

A este comportamiento se le denomina bubbling y es un comportamiento nativo de javascript, que solo se comporta así en los elementos que no tienen shadowDOM.

Cuando los componentes tienen shadowDOM los eventos suben de manera predeterminada y se quedan en el padre, no continúan propagándose.

Si queremos que los eventos de los componentes que tienen shadowDOM se comporten igual que en javascript nativo, deberemos añadir dos parámetros dentro del dispatchEvent. Los parámetros son **bubbles y composed** y debemos cambiar su valor a true.

EVENTOS PERSONALIZADOS BUBBLES Y COMPOSED.

Softtek®

Genera el ejemplo de Bubbles y Composed adaptando los siguientes componentes en tu equipo, y que te sirva para tu biblioteca.

index.html

web component 1

web component 2

web component 3

Eventos táctiles

Softtek®

Para finalizar el tema de eventos voy a explicar con algún ejemplo los eventos táctiles disponibles en polymer, ya que según las estadísticas desde 2017 la navegación móvil ha superado a la navegación de escritorio.

Estos son los siguientes evento:

- **down: Cuando se presiona.**
- **up: Cuando se levanta.**
- **tap: Cuando presionamos y levantamos.**
- **track: Cuando arrastramos o movemos el ratón.**

Para poder usar estos eventos necesitamos importar un fichero, ya que no esta incluido en el core de polymer.

Eventos táctiles (ejemplo).

Softtek®

index.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, minimum-scale=1, initial-scale=1, user-scalable=yes">
  <title>Curso01</title>
  <meta name="description" content="Curso01 description">
  <!-- See https://goo.gl/OOhYW5 -->
  <link rel="manifest" href="./manifest.json">
  <script src="./bower_components/webcomponentsjs/webcomponents-loader.js"></script>
  <link rel="import" href="./src/sandbox/comp-gestos.html">
</head>
<body>
  <comp-gestos></comp-gestos>
</body>
</html>
```


Eventos táctiles

comp-gestos.html:

```
<link rel="import" href="../../bower_components/polymer/polymer-element.html">
<link rel="import" href="../../bower_components/polymer/lib/mixins/gesture-event-listeners.html">
<dom-module id="comp-gestos">
    <template>
        <style>
            h1:nth-child(2) {
                user-select: none;
                cursor: pointer;
                border: 1px solid;
                padding: 40px;
                text-align: center;
            }
        </style>
        <h1 on-track='gesto'>Gestos</h1>
        <h1>Nº total de arrastres de izquierda a derecha...[[arrastrar]]</h1>
    </template>
```


Eventos táctiles

```
<script>
    class ComponenteGestos extends Polymer.GestureEventListeners(Polymer.Element) {
        static get is() {
            return 'comp-gestos';
        }
        static get properties() {
            return {
                arrastrar: {
                    type: Number,
                    value: 0
                }
            };
        }
        gesto(evt) {
            if (evt.detail.state === 'end' && evt.detail.dx > 150) {
                this.arrastrar++;
            }
        }
    }
    window.customElements.define(ComponenteGestos.is, ComponenteGestos);
  </script>
</dom-module>
```

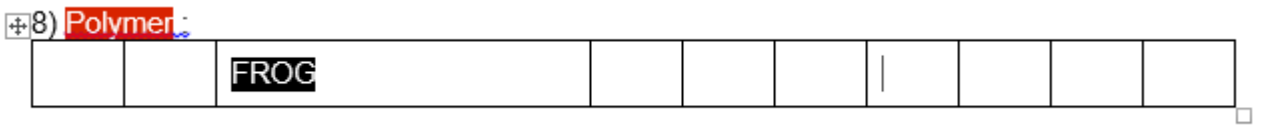

Ejercicio individual

Genera un componente que reciba un numero entero que indicara el tamaño de un arreglo, la primera posición del arreglo tendra el valor de “frog” y las vacias de “null”.

Deberás simular el array en tu pagina con estilos.

Deberás tener una caja de texto donde ingresaras el numero de saltos que dará la rana, si el numero de saltos es mayor al tamaño del arreglo deberá moverse al principio del arreglo.

Para los parámetros de salida “frogindex”, indica la ubicación de la rana en el arreglo.
Y cada que se mueva la rana se debe lanzar un evento que se llame “frog-move”, que deberá Contener from: index[num] y to index[num].



Web component iron-ajax

Softtek®

Dentro del catálogo de componentes de Polymer encontramos la clasificación "Iron", que incluyen elementos que teóricamente se deberán usar en conjunto con otros, para resolver problemas mayores. El componente iron-ajax es el que se usará para realizar solicitudes Ajax, siendo un elemento genérico para hacer cualquier tipo de llamada, que generalmente usarás en el marco de otros componentes para producir elementos más complejos que resuelvan tareas específicas.

Lo bueno de iron-ajax, así como los componentes de Polymer y Web Components en general, es que permiten implementar el comportamiento de manera declarativa. Es decir, mediante el propio HTML, usando el custom element iron-ajax y definiendo su configuración por medio de atributos en la etiqueta, podemos implementar cualquier solicitud asíncrona al servidor y obtener la respuesta de varias maneras distintas.

Lo primero que tenemos que hacer si queremos implementar una solicitud Ajax con iron-ajax es hacer el import del componente. Para ello lo habremos de instalar mediante Bower, con el comando que encontramos en la documentación de iron-ajax.

bower install --save PolymerElements/iron-ajax

Web component iron-ajax

Una vez instalado en el proyecto, podemos hacer el correspondiente import:

```
<link rel="import" href="../bower_components/iron-ajax/iron-ajax.html">
```

Nota: Solo ten cuidado con la URL del href, que tendrás que colocarla atendiendo a tu propia estructura de carpetas.

Una vez hecho el import, podemos usar el componente, colocando la correspondiente etiqueta del custom element.

```
<iron-ajax
  url="http://jsonplaceholder.typicode.com/users/2"
  auto
  handle-as="json"
  last-response="{{data}}"
></iron-ajax>
```


Web component iron-ajax

Softtek®

En esta primera aproximación tenemos el componente configurado con diversos atributos que detallamos:

- **url:** indicamos la ruta a la que queremos conectar con la llamada Ajax.

- **auto:** hace la conexión automáticamente, según se carga la página y el propio componente, al inicializarse, se realizará la llamada sin que tengamos que intervenir manual o imperativamente.

- **handle-as:** esto nos permite indicarle cómo debe tratar la respuesta. Existen varias configuraciones posibles, en este caso se indica que la respuesta la tratará como un JSON, de modo que lo que recibiremos será un objeto Javascript resultado de parsear el JSON recibido.

- **last-response:** este parámetro nos sirve para decirle que la respuesta la tiene que bindear directamente a una propiedad de nuestro componente. Como vemos, el valor que se asigna a last-response está bindeado con "2-way-binding" a la propiedad "data". Es una de las maneras, la más sencilla, de recuperar datos de una solicitud Ajax. Cada vez que la URL cambie, se realizará la conexión y los datos serán asignados a la propiedad bindeada "data", si la URL cambia muchas veces esa propiedad data irá cambiando también y en ella siempre tendremos el valor de la última operación Ajax recibida.

Web component iron-ajax

Softtek®

Algo típico que querrás hacer cuando se realiza una solicitud ajax es mostrar la típica rueda que gira para indicar al usuario que hay un dato que está cargando. Esto es muy sencillo de realizar en un componente iron-ajax porque el propio componente expone hacia afuera una propiedad llamada "loading", que es un boleano que indica si está o no esperando una respuesta del servidor.

Lo general es que tengas ese atributo "loading" bindeado a una propiedad que usarás dentro del componente para expresar la carga.

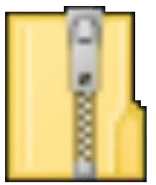
```
<iron-ajax
  url="http://jsonplaceholder.typicode.com/users/2"
  auto
  handle-as="json"
  last-response="{{data}}"
  loading="{{cargando}}"
></iron-ajax>
```

Ahora tenemos en la propiedad "cargando" el true/false dependiendo de si está o no esperando a recibir la respuesta. Algo muy sencillo sería usar esa propiedad dentro de un atributo hidden de una etiqueta que muestre el un mensaje de carga:

```
<div hidden$="[[!cargando]]">Estamos cargando datos...</div>
```


Web component iron-ajax

Ejemplo de componente que realiza Ajax



iron ajax

Para los “bower_components”, se manda por correo carpeta comprimida, sobrescribir en la carpeta donde van a crear el componente.

Web iron-request

Softtek®

Este componente del catálogo de los elementos de Polymer baja a un nivel inferior que el iron-ajax, permitiendo ver otros datos sobre la solicitud recibida, que no te expone directamente el componente de más alto nivel iron-ajax.

Realmente, aunque podrías usarlo a él, sin necesidad de implementar iron-ajax, para hacer una solicitud, no es algo habitual. Sueles trabajar con iron-ajax o iron-form y, cuando tienes que examinar más a fondo la respuesta de la solicitud usarás iron-request. Realmente, como veremos a continuación, la operativa es bastante sencilla y el elemento iron-request aparecerá en tu vida de manera natural: Polymer te lo proporcionará automáticamente y nosotros lo usaremos para recibir datos que necesitemos de la solicitud.

Básicamente, Polymer te entrega el componente iron-request como parámetro al implementar un evento del componente iron-ajax o iron-form llamado "response". Es decir, dentro de un iron-ajax o iron-form podemos detectar eventos "response" y al escribir sus manejadores de eventos recibiremos como parámetro el propio evento y el componente iron-request.

Web iron-request

Softtek®

Ejemplo de uso de iron-request

Este componente se entenderá mejor con un vistazo a un ejemplo más complejo de solicitud Ajax. En este ejemplo enviaremos datos por POST a un servicio web, que lo que hace es recibirlos e insertarlos en un recurso de un API REST, devolviendo un código de HTTP y un cuerpo con el dato recién insertado.

Nuestro componente iron-ajax ahora tendrá una serie mayor de propiedades.

```
<iron-ajax
  id="elajax"
  url="http://jsonplaceholder.typicode.com/posts"
  handle-as="json"
  loading="{{cargando}}"
  method="POST"
  content-type="application/json"
  on-response="respuestaRecibida"
></iron-ajax>
```


Web iron-request

Los atributos nuevos que estamos usando son los siguientes:

method: indica el método de la solicitud (POST, GET, PUT…).

content-type: permite indicar el tipo de información que estamos incluyendo en el cuerpo de la solicitud. En este caso enviaremos en el cuerpo un JSON del dato que deseamos insertar.

on-response: esta es la declaración de un manejador de eventos que vamos a ejecutar cuando se reciba la respuesta del servidor.

body: este atributo no lo estamos usando en el anterior código, pero la propiedad en si la vamos a usar en nuestro ejemplo. En ella indicamos el cuerpo de la solicidut HTTP, que se usa por ejemplo en el caso de enviar por post un dato al servidor. Ese dato lo vamos a cargar de manera imperativa, desde Javascript, más tarde en este ejemplo.

Nota: Observa que estamos usando un servicio web que trabaja como un API REST. Vamos a insertar datos en el recurso http://jsonplaceholder.typicode.com/posts

Web iron-request

Softtek®

Tendremos por otra parte un elemento en la página, que si lo pulsamos llamará al método que ejecutará la solicitud Ajax. Para el ejemplo no se uso con la etiqueta , se llamo con el Evento on-click de un button.

```
<div on-tap="guardar">
  Guardar post!
</div>
```

Nuestro método guardar será encargado de ejecutarse al hacer tap sobre el elemento DIV anterior. Aquí realizamos un uso del componente iron-ajax, en el que se cargará un dato en el cuerpo de la solicitud HTTP, usando la propiedad "body" que hemos comentado antes. Además luego lanzamos la solicitud con el método generateRequest() del componente iron-ajax.

Web iron-request

```
guardar: function() {
  var obj = {
    title: 'Título de un post',
    body: 'Este es el cuerpo de un post',
    userId: 1
  };
  this.$.elajax.body = obj;
  this.$.elajax.generateRequest();
},
```

Esa solicitud se ejecutará entonces, gracias al comportamiento del iron-ajax. Finalmente, cuando el servidor devuelva el resultado de la solicitud, se producirá el evento "response" del iron-ajax, para el cual se definió un manejador declarado en la etiqueta iron-ajax. Allí es donde, por fin, podremos usar el componente iron-request.

```
respuestaRecibida: function(e, request) {
  //el parámetro "request" es el iron-request de esta solicitud
  console.log(request.response)
}
```


Web iron-request

Softtek®

Algunas propiedades útiles de iron-request
Por medio de las propiedades de iron-request podemos observar con mucho detalle el estado de nuestra solicitud. Algunas de sus propiedades más útiles son:

response: con el cuerpo de la respuesta. Algo normal es que se nos devuelva un JSON, entonces recibiríamos el objeto Javascript una vez parseado el JSON.

status: el número del status de la solicitud HTTP: 200, 404, 201... esos códigos de respuesta dependen del API que estemos usando, aunque seguramente sepas que existen unos que son más estándar como 200 cuando la solicitud tuvo éxito, 201 cuando una inserción tuvo éxito, 404 cuando el recurso no fue encontrado...

succeeded: es un boleano que indica si la solicitud tuvo éxito (si el status es 200, mayor que 200 y menor que 300. O bien cero como status).

errored: el contrario de succeeded.

timedOut: Un boleano que indica si la solicitud terminó por un "time out".
Con todos esos datos, y otros, será muy fácil interpretar la solicitud y mostrar la información correcta al usuario. Recuerda que puedes encontrar otras propiedades y métodos en la documentación del componente iron-request.

Web iron-request

Este sería un ejemplo sencillo de manejador de evento donde recibimos el iron-request.

```
respuestaRecibida: function(e, request) {
  if(request.succeeded) {
    this.mensaje = 'la solicitud se resolvió correctamente con código ' + request.status
  } else {
    this.mensaje = 'la solicitud nos ofreció resultados incorrectos, con código ' +
request.status
  }
},
```


Generar API REST FALSO.

PRE-REQUISITOS PARA UTILIZAR JSON-SERVER

El único requisito es tener instalado Node.JS.

CREANDO NUESTRO API REST FALSO

Primero creamos una carpeta para el API REST falso e ingresamos.

```
mkdir api-fake && cd api-fake
```

Luego daremos por iniciado nuestro proyecto creando interactivamente nuestro archivo `package.json`, con la siguiente instrucción.

```
npm init -f
```


Generar API REST FALSO.

Softtek®

Mediante la instrucción `npm init -f` o `npm init –force` evitamos todas las preguntas incómodas de `npm init`.

INSTALANDO JSON-SERVER COMO DEPENDENCIA

Instalamos la única dependencia de nuestro proyecto, la cual es: **json-server**

```
npm i -S json-server
```


Generar API REST FALSO.

Softtek®

Mediante la instrucción `npm i -S` podemos instalar las dependencias de nuestro proyecto, esto hará que dentro de nuestro archivo package.json se agregue la llave `dependencies` y allí se almacenen las dependencias principales de nuestro proyecto.

Luego de instalar json-server, nuestro archivo **package.json** debe verse algo así:

```
{
"name": "api-fake",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
"json-server": "^0.8.14"
}
}
```


Generar API REST FALSO.

Softtek®

CREANDO LA BASE DE DATOS PARA JSON-SERVER

Luego simplemente creamos la carpeta `db` y dentro de esta creamos el archivo `database.json`.

```
mkdir -p ./db/ && touch ./db/database.json
```

Ahora procedemos a agregar el contenido a nuestra base de datos, en este caso agregaremos el contenido para la tabla **customers**(clientes), para lo cual editamos el archivo `./db/database.json`.

En nuestra base de datos json simplemente agregamos lo siguiente:

```
{
"customers": []
}
```


Generar API REST FALSO.

CONFIGURANDO Y EJECUTANDO NUESTRO SERVIDOR CON JSON-SERVER

Ahora sólo nos falta configurar nuestro servidor json-server, para lo cual editamos nuestro archivo `./package.json` y agregamos la tarea que llamaremos **server**, de la siguiente manera:

```
{
"name": "api-fake",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"server": "json-server --watch ./db/database.json --port 3004",
"test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
"json-server": "^0.8.14"
}
}
```


Generar API REST FALSO.

Softtek®

Como podemos apreciar sólo hemos agregado la línea:

```
"server": "json-server --watch ./db/database.json --port 3004"
```

En la cual simplemente le indicamos a npm que creé la tarea llamada **server** ejecutando json-server más 2 parámetros **watch** y **port**. Con watch le decimos que esté al tanto de cualquier cambio en nuestra base de datos

y con el parámetro port le indicamos que use el puerto 3004,

básicamente para no tener conflictos con otros servicios.

CORRIENDO NUESTRO SERVIDOR

Para correr el servidor, simplemente escribimos lo siguiente en nuestro terminal:

```
npm run server
```


Generar API REST FALSO.

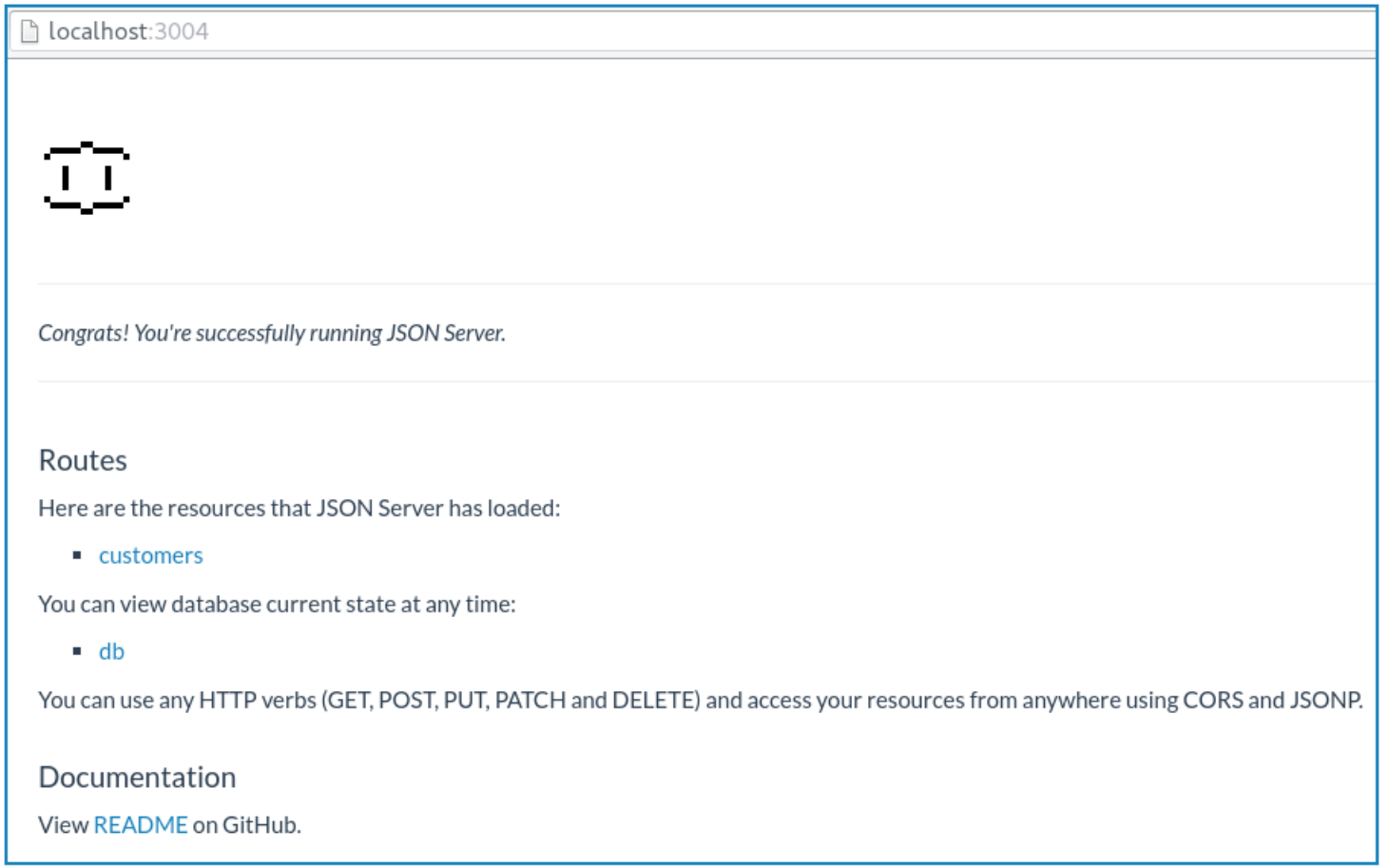
Softtek®

Si todo funcionó correctamente nuestro terminal debe verse así:

```
--- services/api-fake » npm run server

> api-fake@1.0.0 server /home/jan/projects/services/api-fake
> json-server --watch ./db/database.json --port 3004


  \{^_^}/ hi!

  Loading ./db/database.json
  Done

  Resources
  http://localhost:3004/customers

  Home
  http://localhost:3004

  Type s + enter at any time to create a snapshot of the database
  Watching...

GET / 304 9.517 ms - -
GET /stylesheets/style.css 304 1.739 ms - -
GET /images/json.png 304 1.654 ms - -
GET /db 304 7.650 ms - -
```


Generar API REST FALSO.

Ahora abre cualquier navegador e ingresa a la siguiente dirección http://localhost:3004 y listo!, ya tenemos corriendo nuestro servidor. Y en nuestro navegador se debe ver así:



Generar API REST FALSO.

¿CÓMO USO JSON-SERVER?

Ahora que ya tenemos corriendo el servidor, podremos conocer el verdadero poder de json-server insertando, actualizando, consultando y eliminando datos de la tabla customers en nuestra base de datos. Todo eso lo haremos utilizando los verbos HTTP (get, post, put y delete) para comunicarnos con nuestro API REST falso.

Ahora verifica el siguiente web component que se conecta al api rest falso para guardar información con post, y para obtener información puedes ocupar get pero ya en un ejemplo anterior se indico como abrir el archivo json y trabajar con los datos mostrados, para la practica final vas a ocupar Delete para borrar datos y put para actualizar.

servicios rest

Practica Final

Softtek®

Genera en base a desarrollo declarativo, una aplicación responsiva que genere una agenda.
Inicialmente te solicitara usuario y password el cual validara en un archivo json de nombre access.json
Al ingresar a la aplicación mostrara una imagen central referente a una agenda y tendrá los siguientes Botones:

- **Agregar Evento:** Guarda en el archivo agenda.json fecha y hora del evento a si como descripción del evento, debe validar que no se sobreescriba en un evento ingresado posteriormente, cada evento tiene un ID que corresponde a cada usuario.

- **Consultar:** Muestra los eventos por día en base a un filtro por fecha el rango máximo para visualizar son 7 días.

- **Eliminar:** Este evento elimina un evento de la agenda siempre y cuando sea igual o mayor a la fecha actual del sistema y se este visualizando en pantalla el resultado del Botón Consultar.

- **Actualizar:** Este evento actualiza un registro en la agenda solo para el campo descripción y debe la fecha debe ser igual o mayor a la fecha del sistema para poder realizar la actualización.

- **Salir:** cierra la aplicación y muestra una imagen de despedida solo actualizando la pagina volverá a Pedir el login.

Softtek®

Polymer 2 Clase 2

Curso de capacitación para personal asignado a BBVA