Calabash es un framework open source que permite el testing automatizado de aplicaciones moviles desarrollado por Xamarin; para ello utiliza Cucumber, software escrito en Ruby pensado para el testing de software, aunque es independiente del lenguaje utilizado. También podemos utilizarlo para el desarrollo web usando Selenium, aunque en este articulo veremos como usarlo para las aplicaciones Android.

Calabash ejecuta una serie de pruebas funcionales automatizadas contra la interfaz de usuario de una aplicación. Está pensado para ser usado en un modelo de Testing BDD.

Instalación y funcionamiento

Para empezar a instalarlo, tienes que tener instalado Ruby, si no lo tienes lo puedes instalar desde aquí: RubyInstaller.org

También, tienes que tener instalado el JDK (Java Development Kit), que lo puedes descargar de aqui y el SDK de Android, lo puedes descargar desde aqui

Una vez instalado todo esto tienes que ejecutar el siguiente comando para instalarlo:

gem install calabash-android

Si ya has instalado calabash, puedes generar un esqueleto Cucumber con el siguiente comando:

calabash-android gen

Automáticamente generará las carpetas con los archivos .feature y .rb

En el archivo con extension .feature se especifica lo que queremos que Calabash ejecute automáticamente. Por ejemplo un feature con un escenario para el login muy sencillo, de una aplicación quedaría de esta manera:

```
1  Feature: Login Feature
2  Scenario: As a valid user I can log into my app
3  Given I enter text "luisca_jl" into field with id "txtLogin" //pasamos
       los id de los elementos como string
4  And I enter "1234" into input field number 2
5  And I go back //Vuelve atras para cerrar el teclado
6  When I press "btnLogin"
7  Then I see "Bienvenido de nuevo"
```

Puedes ejecutar el testing en la maquina virtual de Android que viene con el SDK o directamenete en nuestro dispositivo si esta conectado a nuestro ordenador.

Por último ejecutamos el comando para que empiece automáticamente a testear la aplicación:

calabash-android run <apk&gt

Y automáticamente abrirá la aplicación en la maquina virtual o en nuestro dispositivo si lo tenemos conectado.

Saludos, buen señor. En esta ocasión traigo un tutorial sobre la creación de paginas web SPA con Angular2 y TypeScript. Para empezar, hay que definir que es SPA. SPA son las siglas de «Single-page application», se trata de una web compuesta de una solá página, esto permite dar una experiencia mas fluida al usuario, ya que tarda mucho menos en cargar. Para comunicarse con el servidor utiliza API REST. Angular2 es el framework que facilita la creación de paginas webs SPA, utilizando para ello TypeScript, que es un lenguaje con una sintaxis muy parecida a Java.

Instalación de Angular2

Lo primero que tenemos que descargar es NodeJs, lo puedes descargar de aqui: https://nodejs.org/en/. Una vez instalado, abrimos su consola de comandos y ejecutamos los siguientes comandos:

npm install -g angular-cli@0.0.24

ng new main

cd main

Hecho todo esto simplemente ejecutamos ng serve para que nos ejecute el servidor en local, en la direccion localhost:4200, abrimos esta direccion en nuestro navegador y comprobamos que funciona correctamente.

Ejemplo de aplicación básica

Angular2 utiliza componentes. Un componente es un archivo con extension .ts, que contiene un template (nuestro archivo html con la estructura que se muestra en la web), y la lógica de esa página escrita en TypeScript. Veamos este ejemplo:

```
1  import {Component} from 'angular2/core';
2
3  @Component({
4  selector: 'app',
5  templateUrl: 'app/app.component.html'
6  })
7  export class AppComponent {
8   name = 'Tutorial';
9   setNombre(nombre:string){
10  this.nombre= nombre;
11  }
12 }
```

Como vemos en el código, le pasamos el archivo html en templateUrl y dentro de export class, va la lógica de nuestra página. En este caso, simplemente he creado una variable name, con un string

por defecto. A continuación creo una función setNombre para cambiar la variable. Veamos ahora el código html de nuestra página:

```
<h1>Hello {{name}}!</h1>
<button (click)=”setName('Frost')”>
 Tu nombre
</button>
```

Simplemente añadimos la etiqueta {{nombre}}, para que coja el valor de la variable del componente. Con la etiqueta (click)=«setName(“Frost”)» lo que hace es llamar a la función de nuestro componente con el valor que le pasamos. Ejecutando este ejemplo nos da como resultado:

Cuando hacemos clic en el botón nos cambia el titulo con el string «Frost».

Otro ejemplo

En este ejemplo voy a demostrar cómo podemos utilizar observables para implementar un ejemplo básico. Para esto nos vamos a basar en dos conceptos, bbservador y observable. Los nombres son similares, pero un observable es algo que emite eventos que pueden ser observados por un observador. En resumen el observador observa y lo observable es la producción de actos que se observan. Primero creamos nuestra clase utilizando un servicio como se muestra a continuación:

```
import {Subject } from 'rxjs/Subject';
import {Customer} from './customer';
export class CustomerEventEmitter extends Subject&lt;Customer&gt;{
    constructor() {
        super();
    }
    emit(value) { super.next(value); }
}

import {CustomerEventEmitter} from './customer-event-emitter';
export class PubSubService{
   Stream:CustomerEventEmitter;
   constructor(){
       this.Stream = new CustomerEventEmitter();
   }
}
```

Con este servicio muy sencillo cogemos una cadena de objetos de timo Customer. El siguiente paso es hacer nuestra clase para crear Customers, como se muestra:

```
import {Component,Input} from '@angular/core';
import {PubSubService} from './pub-sub-service';
```

```
import {Customer} from './customer';
@Component({
    selector: 'producer',
    templateUrl: './components/pub-sub/producer.html'
})
export class Producer {
    @Input() firstName = '';
    @Input() lastName = '';
    constructor(private pubSubService:PubSubService){
    }
    createCustomer(){
        let customer = new Customer();
        customer.firstName = this.firstName;
        customer.lastName = this.lastName;
        this.pubSubService.Stream.emit(customer);
    }
}
```

Por ultimo creamos nuestra clase para subscribirnos a la cadena de Customers, simplemente llamando a la función «susbscribe()»

```
import {Component,OnInit} from '@angular/core';
import {PubSubService} from './pub-sub-service';
import {Customer} from './customer';
@Component({
    selector: 'consumer',
    templateUrl: './components/pub-sub/consumer.html'
})
export class Consumer implements OnInit{
    processed = [];
    subscription = null;
    constructor(private pubSubService:PubSubService){
    }
    ngOnInit(){
        this.subscription = this.pubSubService.Stream.subscribe(
            customer => this.processCustomer(customer));
    }
    processCustomer(customer){
        this.processed.push(customer);
    }
    stopProcessing(){
        this.subscription.unsubscribe();
    }
```

```
22 }
```

Al introducir los datos y pulsar el boton «Create Costumer», se añade una fila a la tabla que aparece abajo con los datos introducidos.

DEMO

Saludos, buen señor. Hoy vamos a aprender a hacer un widget del tiempo usando RxJS, pero antes hay que saber que es RxJs y que es la programación reactiva. RxJs es una librería para JavaScript para la creación de programas asíncronos (No esperamos, el programa no pregunta si una operación en concreto se ha completado, sólo comprueba si ha llegado algún mensaje) basados en eventos. La programación reactiva es un paradigma de la programación que se basa en los flujos de datos y a la propagación de los datos. La programación reactiva aprovecha las ventajas del patrón Iterator y las ventajas del patrón Observer. Es posible escribir sentencias del estilo filtrar, ordenar, seleccionar, etc; sobre eventos. La diferencia es que la consulta es evaluada conforme llegan los datos. Uno puede evaluar datos en tiempo real.

Bien, vamos a aprovechar las ventajas de la programación reactiva para el desarrollo web.

Creación del widget

Para la creación de nuestro widget vamos a utilizar la API de OpenWheaterMap, para ello lo primero que tenemos que hacer es registrarnos en su página oficial:

http://openweathermap.org/current

En la sección API Keys es donde encontraremos nuestra clave para nuestro widget. Lo primero que tenemos que hacer es añadir la declaración de nuestra API Key y la url de la API, para ello añadimos lo siguiente dentro de la sección script de nuestro código html:

```
1 const API_KEY = ”SUSTITUYE_ESTO_POR_TU_API_KEY”;
2 const API_URL = `http://api.openweathermap.org/data/2.5/weather?appid=$
     {API_KEY}`;
3 appid=${API_KEY}`;
```

El siguiente paso es seleccionar los dos elementos DOM de nuestro html con los que vamos a interactuar, el texto de la temperatura y el input donde introducimos la localización:

```
1 var weatherText = document.querySelector('#weather');
2 var searchInput = document.querySelector('#location-input');
```

Una vez hecho esto toca recoger la información y mandársela a la API con una petición HTTP, para ello utilizamos una función RxJS llamada Rx.Observable.fromEvent() en conjunción de un debounce de 50mms, que se encargará de registrar un nuevo evento 500ms después del evento anterior. Tras esto mapeamos la información y se la enviamos a la API como sigue:

```
var searchInputSource = Rx.Observable.fromEvent(searchInput, 'keyup').
    debounce(500);

var requestOnFindStream = searchInputSource.map(ev =&gt; {
    return API_URL + ”&amp;q=” + ev.target.value;
});


function convertToCelsius(kelvin) {
 return (kelvin - 273.15).toFixed(1);
}
```

Invocamos la función flatMap() para crear un nuevo stream a partir de una cadena de strings. Por último queda subscribirse al observable y convertir la temperatura a grados Celsius de esta manera:

```
var responseStream = requestOnFindStream
            .flatMap(requestUrl =&gt; {
                return Rx.Observable.fromPromise($.getJSON(requestUrl))
                    ;
            })
            .map(response =&gt; convertToCelsius(response.main.temp))
            .startWith(0);

responseStream.subscribe(temp =&gt; {
            weatherText.innerHTML = `${temp} °C`;
  });
```

Para terminar puedes consultar la DEMO del resultado final.

Saludos, buen señor. Antes de empezar a ver como se instala y como se configura Jenkins es importante saber qué es la integración continua, ya que Jenkins es un software open source de integración continua escrito en Java.

Integración continua

La integración continua es una práctica de desarrollo de software y del desarrollo web, en la cual los desarrolladores de software suben su código a un repositorio central donde automáticamente pasan las pruebas métricas y de calidad. Esta técnica se suele realizar regularmente para detectar fallos cuanto antes y así mantener el código siempre actualizado.

Jenkins es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta control de versiones como Git y tiene la posibilidad de instalar variedad de plugins para aumentar su funcionalidad.

Instalación y configuración de Jenkins

Para instalar Jenkins lo primero que tenemos que hacer es descargar el archivo .war desde su página oficial:

https://jenkins.io/index.html

Nos dirigimos a la direccion en la que hemos guardado el archivo .war mediante linea de comandos o cmd y ejecutamos el siguiente comando:

java -jar jenkins*.war

Otra opción es meter el archivo en nuestro server Tomcat en la carpeta webapps.

Nos dirigimos a la dirección http://localhost:8080/ y vemos que todo funciona correctamente.

Si queremos que Jenkins se pueda utilizar para aplicaciones Java lo que tenemos que hacer es dirigirnos, en el menú de la izquierda, a Administrar Jenkins y despues a Configurar el Sistema para añadir nuestro JDK de Java. Al añadir un JDK podemos hacerlo añadiendo la ruta donde tenemos instalado el JDK o podemos hacer que Jenkins lo instale automáticamente.

En mi caso quiero que coja el código de Git, pero no viene por defecto, aunque podemos instalarlo mediante plugin. Para ello, volvemos a pulsar en el menú «Administrar Jenkins» y pulsamos en la opción «Administrar Plugins». Se nos mostrará la información de los plugins, pulsando en la pestaña «Todos los plugins», nos aparece una lista con todos los plugins que podemos instalar, así que filtramos por «git plugin» y lo seleccionamos para su instalación, y pulsamos el botón «Instalar sin reiniciar».

Creación de una tarea

Pulsamos en el menu de la izquierda en el botón «Crear nueva tarea», introducimos el nombre y marcamos la opción «Crear un proyecto de estilo libre», en la siguiente pantalla añadimos una descripción y en la seccion «Configurar el origen del código fuente» seleccionamos Git, e introducimos la URL de nuestro repositorio de GitHub

Mas abajo en la sección «Disparadores de ejecuciones» podemos decidir cada cuanto tiempo se va a integrar el codigo de Git, en mi caso voy a seleccionar la opcion de «Consultar repositorio (SCM)» con un programador de H/20 * * * * para que pregunte a Git si tiene algún cambio y los integre automáticamente.

Por último puedes darle al botón «Construir ahora» en el menú de la izquierda para que integre nuestro proyecto, dejado del menú de la izquierda aparecerá el resultado de la integración.

JavaScript es un lenguaje que surgió para el desarrollo web, para poder interaccionar con el usuario y hacer ciertas acciones básicas. En la actualidad JavaScript se ha ampliado ofreciendo nuevas funcionalidades. Se utiliza principalmente en su forma del lado del cliente, aunque existe una versión para el lado del servidor. JavaScript puede tener todo lo que un lenguaje orientado a objetos, tiene que ofrecer.

Para entender como funciona la orientación a objetos en JavaScript, primero hay que empezar definiendo que es la orientación a objetos.

Programación orientada a objetos (POO) es un modelo de lenguaje de programación organizada alrededor de los objetos en lugar de «acciones» y datos en lugar de la lógica. Nos interesa mas saber como vamos a relacionar los objetos y la lógica del programa que los propios datos.

Tenemos que saber algunos términos asociados a la orientación de objetos:

Clase: Definimos como va a ser nuestro objeto.

```
<li><strong>Objeto</strong>: Una instancia de una clase</li>
    <li><strong>Métodos y propiedades</strong>: Características
       concretas del objeto y acciones que puede realizar.</li>
    <li><strong>Constructor</strong>: Con este método inicializamos las
        propiedades con los valores que le pasamos. Este método se
       ejecuta al crear un objeto.</li>
    <li><strong>Herencia</strong>: Con la herencia podemos hacer que
       una clase adquiera las características de otra.</li>
    <li><strong>Polimorfismo</strong>: Las clases podrían definir el
       mismo método o propiedad.</li>
</quote>
```

Programación orientada a objetos en JavaScript

Crear una clase con JavaScript tiene esta pinta:

```
function Car(){}
```

Con esto definimos que nuestros coches tienen una serie de caracteristicas (4 ruedas, 1 motor, 5 asientos).

En JavaScript no necesitamos crear un método específicamente para el constructor sino que podemos usar la función que creamos antes para inicializar las características de nuestra clase.

```
function Car() {
  this.wheels = 4;
    this.engines = 1;
      this.seats = 1;
      };
```

Para crear una instancia de la clase que acabamos de crear simplemente llamamos a la palabra new que indica a JavaScript que estas creando un objeto nuevo.

Pero si lo que queremos es que cada instancia tenga diferentes características pero con la misma estructura, tenemos que añadir paramentos a nuestro constructor

```
function Car(wheels, seats, engines) {
    this.wheels = wheels;
    this.seats = seats;
    this.engines = engines;
};
var myCar = new Car(6, 3, 1);
```

De esta forma, al crear el objeto nuevo, le pasamos los valores que queramos, siguiendo la estructura definida en el constructor.

Para que una clase herede de otra, tenemos que usar su prototipo. Cada objeto tiene un prototipo. De este prototipo hereda todas las propiedades y métodos, por eso si queremos hacer que una clase herede de otra lo que tenemos que hacer es cambiar el prototipo de la clase hija por el de la clase del padre.

```
function Dad() {
    this.atributoDad=4;
}

function Son() {
    this.atributoSon=5;
}
Son.prototype = new Dad;
```

Este blog se encuentra alojado en GitHub pages, por lo que para realizarlo he decidido utilizar Jekyll para su programación, un generador de páginas estaticas. A continuación voy a explicar los pasos que di para su realización.

Lo primero que hay que hacer es crear un repositorio en GitHub para alojar los archivos de la página web, mi repositorio se encuentra en https://github.com/Frostqui/my-blog.

A continuación creamos una rama con el nombre «gh-pages», y en los ajustes de GitHub, le indicamos que utilice este repositorio para la página web.

Lo sieguiente que hay que hacer es clonar nuestro repositorio en local y poner el archivo de configuración, para ello creamos un archivo en la raíz de nuestro proyecto llamadado “_conf.yml” con la siguiente configuración:

De esta forma, le indicamos a Jekyll que no utilice el archivo README, y cual es la ruta base de nuestro repositorio. Lo siguiente será inicializar JeyKill para ello, descargamos Ruby y ejecutamos el comando:

gem install jekyll bundler

Tras esto ejecutamos JeKyll para que genere un servdior en local para poder ver los cambios al desarrollar:

jekyll serve –watch –baseurl «»

Cuando se ejecuta, jekyll crea una carpeta llamada _sites, hay que acordarse de añadir esta carpeta al archivo .gitignore para no añadirla al repositorio.

Jekyll utiliza un sistema de plantillas, para utilizarlo, tenemos que crear una carpeta llamada "_layouts«, dentro de esta carpeta creamos el archivo»default.html". En este archivo vamos a incluir todas las librerias y archivos css y el navbar (la barra de navegacion de arriba) para que Jekyll lo utilice en todas las páginas. Para ello añadimos el código html de forma normal, pero añadiendo la etiqueta

{% raw %} {{content}} {% endraw %}

para que se cargue en ese lugar el contenido que creemos.

Creamos archivo index.html y añadimos al principio la plantilla que estamos utilizando:

layout: default

Para añadir posts, tenemos que crear una carpeta llamada "_posts". Estos posts, tienen que ser archivos con extensión .md y con el siguiente nombre: 2016/09/16-nombre-del-post Como veis el nombre tiene que estar formado por la fecha separada mediante barras y el nombre del post separado por guiones. El siguiente paso será añadir los pots a la pagina de inicio, mediante el uso de un bucle for recorriendo los posts, para ello añadimos el siguiente código:

{% raw %}

{% for post in site.posts %}

{{post.title}}

{{ post.description }}

{% endfor %}

{% endraw %}

Con esto ya tenemos la funcionalidad muy básica de nuestro blog alojado en GitHub con Jekyll, a partir de aqui la página se puede extender y personalizar a gusto de cada uno.

Empiezo nuevo proyecto, esta vez , como excusa para indagar y aprender sobre el diseño y la programación de una web. Esta web va a ser construida desde 0, pasado por todas las fases del desarrollo, es decir, desarrollo Full-Stack.

La idea

Para empezar describiré el proyecto de desarrollo web que quiero hacer. Creavelas, como su propio nombre indica, es un espacio donde la gente crea y comparte velas virtuales. Al registrarse, puede

encender una vela, en la que tiene que seleccionar un tipo de vela (vela de la suerte, vela del amor, vela del dinero, etc), escribir la leyenda de la vela, y, opcionalmente, elegir hacia personas esta destinada la vela. Al crear la vela, esta se queda encendida durante un día, tras esto automáticamente se elimina. Mediante un botón en la vela, otros usuarios pueden aumentar el tiempo de vida de las velas. Es decir quiero que la web tenga un enfoque de red social.

Bien, una vez definido lo que quiero hacer es hora de pensar los primeros diseños y logotipos para tener una base hacia la que empezar a desarrollar. Lo primero que hice fue diseñar el logotipo.

Quiero que mi página sea muy sencilla por lo tanto el logotipo tiene que ir en consonancia. Un logotipo tiene que ser legible independientemente del tamaño que elijamos por eso decidí hacerlo con formas muy sencillas de forma que se apreciara perfectamente que se trata de una vela. Mas que un logotipo, he diseñado un isotipo ya que únicamente tiene un icono. Decidí hacerlo redondo porque el círculo representa la propia luz que emite la vela. Quiero que la web tenga colores pastel, colores suaves que transmitan tranquilidad.

Diseño UX & UI

Toca hacer los mockups, una parte muy importante del desarrollo web, para tener una idea de como va a ser el diseño de la web. Para hacer los mockups yo utilizo Photoshop aunque también hay otras herramientas mas especializadas.

Para el navbar, he decidido poner el logotipo junto con las letras de Creavelas, la barra de búsqueda, el botón para crear velas, y la imagen de nuestro perfil si hemos iniciado sesión. Como veis es un diseño muy simple e intuitivo , sin recargar el navbar.

Para la página principal he decidido mostrar las velas. Para facilitar la navegación en puesto un pequeño menú con tres opciones, inicio, para ver las velas populares, suscripciones, para ver las velas de las personas a las que sigues y cercanas, como su nombre indica, para ver las velas de la gente vive cerca nuestra.

A la derecha he puesto un sidebar, o menú lateral para dirigirse a nuestro perfil, o a los ajustes, entre otros. Este menú es provisional y estoy pensando en quitarlo ya que ocupa demasiado espacio.

En cuanto al diseño de cada vela, una imagen a la derecha, representando el tipo de vela que es, ya que cada vela tiene asociada una imagen, y a la derecha el texto de la vela con el autor. Todavía falta por incorporar un boton para aumentar la duraccion de la vela o el tiempo que queda hasta que la vela se apague, como digo no es el diseño final ni mucho menos, simplemente lo voy a utilizar de referencia.

Para la página personal para cada usuario, he decidido meter una imagen de cabecera la cual se puede personalizar para cada usuario. La página, por ahora, simplemente muestra las velas de cada usuario.

Por último he diseñado la página de registro, simplemente un formulario para rellenar con los datos y una imagen de fondo.

Quedan por diseñar mas páginas pero con lo que tengo me sirve para empezar.

Esto es todo por este episodio espero tener las ganas y la inspiración suficientes para seguir adelante con el proyecto. Para cualquier duda o sugerencia puedes contactar conmigo desde la sección contacto. Un saludo.

DEMO

En este post voy a explicar como hacer una página que muestre citas célebres aleatorias, para ello voy a utilizar HTM,CSS y JavaScript para el diseño de la página, y la API de andruxnet para generar las citas célebres. Para realizar peticiones a la API voy a usar AJAX.

AJAX es de carácter reactivo, es decir, puede cambiar la web, o realizar consultas a un servidor, después de cargar la página y sin interferir en su visualización, es decir, podemos cambiar un dato del html de la página sin tener que recargarla. Para el desarrollo web, nos interesa ofrecer la mejor experiencia al usuario, y para ello usando AJAX, vamos a hacer una llamada a la API y a cambiar la página sin tener que recargarla.

Creación de la página

Para hacer la página voy a usar Boostrap, y para ello hay que importarlo, ya que personalmente, me parece una librería que facilita mucho el diseño, sobre todo para el diseño responsive. Puedes descargar Bootstrap desde la página oficial (http://getbootstrap.com/), o directamente importarlo en tu proyecto usando CDN, añadiendo lo siguiente dentro del

```
1 <!-- Latest compiled and minified CSS -->
2 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap
    /3.3.7/css/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous">
3
4 <!-- Optional theme -->
5 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap
    /3.3.7/css/bootstrap-theme.min.css" integrity="sha384-
    rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
    crossorigin="anonymous">
6
7 <!-- Latest compiled and minified JavaScript -->
8 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/
    bootstrap.min.js" integrity="sha384-
    Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
    crossorigin="anonymous"></script>
```

Lo siguiente que hago es descargar JQuery desde su página oficial (https://jquery.com/). Jquery es

una librería para JavaScript que me va a permitir seleccionar de una manera mas fácil los elementos html para cambiarlos de forma dinámica, aparte de que lo voy a usar para hacer peticiones a la API usando AJAX (no recarga toda la página cuando hace la petición).

En cuanto al código HTML del quote generator, lo que hago simplemente es crear un elemento

en el que voy a meter otros 3

, uno para el texto de la cita, otro para el autor de la cita y otro en el que lo voy a dividir en dos columnas (mediante el uso de Bootstrap) para meter un botón para compartir en twitter y un botón para generar otra cita.

```
1
2  <div class="titulo">Random quote by Frostq </h2>
3  <div class="container-fluid">
4      <div class="row">
5          <div class="col-sm-6 main-content col-centered">
6              <div class="centered_content">
7                  </br>
8                  <div class="quote">
9                      <em><span id="text"></span></em>
10                 </div>
11                 </br>
12                 <div class="quote-autor">
13                     <span id="author"></span>
14                 </div>
15                 </br>
16                 <div class="row">
17                     <div class="col-sm-6 tweet">
18
19                         <a class="twitter-share" id="twitterlink" data-
                               size="large" target="_blank"></a>
20
21                     </div>
22                     <div class="col-sm-6">
23                         <button class="boton btn green">Random quote</
                               button>
24
25                     </div>
26                 </div>
27
28
29             </div>
30         </div>
```

```
    </div>

</div>
```

Respecto al css lo único que hago es añadir unos estilos muy simples para dejarlo mas bonito, esta parte es a gusto de cada uno.

```
body {
    background-color: #dae0e3;
    font-family: 'Droid Sans', sans-serif;
}


.main-content{
       float: none;
    margin: 0 auto;
    background-color:white;
    margin-top:10px;
    height:30%;
  -webkit-box-shadow: 5px 5px 16px 1px rgba(0,0,0,0.24);
-moz-box-shadow: 5px 5px 16px 1px rgba(0,0,0,0.24);
box-shadow: 5px 5px 16px 1px rgba(0,0,0,0.24);

}

.titulo{
    text-align: center;
  margin-top:9%;
  color:grey;

}

.centered_content{
    text-align:center;

}

.btn {
  border-radius: 5px;
  padding: 10px 25px;
  float:right;
  font-size: 22px;
  text-decoration: none;
```

```
37    margin-top: 15px;
38      font-weight: bold;
39      margin-right: 15px;
40    color: #fff;
41    position: relative;
42    display: inline-block;
43  }
44
45  .btn:active {
46    transform: translate(0px, 5px);
47    -webkit-transform: translate(0px, 5px);
48    box-shadow: 0px 1px 0px 0px;
49  }
50
51
52  .green {
53    background-color: #2ecc71;
54    box-shadow: 0px 5px 0px 0px #15B358;
55  }
56
57  .green:hover {
58    background-color: #48E68B;
59  }
60
61  .twitter-share{
62    display:block;
63      background: url(”https://static.addtoany.com/images/blog/tweet-
          button-2015.png”);
64    color: #000000;
65    cursor: pointer;
66    font-weight: bold;
67    height: 82px;
68    padding-bottom: 2px;
69    width: 200px;
70  }
71
72  #text{
73
74    font-size:1.8em;
75
76  }
77
78  #author{
```

```
79
80    font-size:1.4em;
81
82  }
```

Para finalizar el quote generator, voy a explicar las funciones que he añadido en la parte de JavaScript con Jquery.

La primera función nos va a permitir generar la cita y sustituirla en nuestro código html. Lo primero es usar AJAX para llamar a la API , en caso de que la petición sea correcta nos devolverá un JSON con la cita y el autor, que acto seguido las guardaremos en variables. Después lo que hago es cambiar el atributo «href» del botón para compartir en twitter, para que el usuario pueda twittear la frase que ha generado. Por último selecciono con Jquery el texto de la cita y el autor y lo cambio por el que ha generado la API.

```
1  function getQuote() {
2      $.ajax({
3          headers: {
4              ”X-Mashape-Key”: ”
                   dITqRwBOt6mshm55nVGnfBU8bAVLp1MqSdRjsn3G3wFvdesZxZ”
5              , Accept: ”application/json”
6              , ”Content-Type”: ”application/x-www-form-urlencoded”
7          }
8          , url: 'https://andruxnet-random-famous-quotes.p.mashape.com/
               cat='
9          , success: function (response) {
10             var r = JSON.parse(response);
11             currentQuote = r.quote;
12             currentAuthor = r.author;
13
14
15             if (inIframe()) {
16                 $('#twitterlink').attr('href', 'https://twitter.com/
                       intent/tweet?&text=' + encodeURIComponent('”' +
                       currentQuote + '” ' + currentAuthor + ”   /   ” + ”
                       http://codepen.io/Frostq/pen/VjEZqm” + ” @DiegoLopGr
                       ”));
17
18
19             }
20             $(”#text”).html(r.quote);
21             $(”#author”).html(” - ” + r.author + ” - ”);
22
```

```
23
24
25          }
26      });
27  }
```

Solo queda hacer un par de funciones, una para que al pulsar el botón de compartir en Twitter, se abra el enlace en una pestaña nueva del navegador, y la función principal que se ejecuta al cargar la página web donde hacemos las llamadas a las funciones anteriores.

```
1  $(”document”).ready(function () {
2      getQuote();
3      $('.boton').on('click', getQuote);
4
5  });
6
7
8  function inIframe() {
9      try {
10         return window.self !== window.top;
11     } catch (e) {
12         return true;
13     }
14 }
```

Empecemos este post viendo qué es VIM. VIM es un editor de texto incorporado en todos los sistemas UNIX. Proviene de otro editor, VI, aunque con el paso del tiempo se han ido implementando mejoras. Si estas utilizando Windows no te preocupes porque puedes descargarte VIM desde su página oficial http://www.vim.org/download.php.

Ventajas y desventajas de este editor

Muy bien pero, ¿Cuáles son las ventajas de VIM?.

A diferencia de otros editores de texto, este esta pensado para ser utilizado en la terminal del ordenador, por tanto no vas a necesitar utilizar el ratón. Cuando estamos programando, perdemos demasiado tiempo cambiando la mano del teclado al ratón y viceversa, pero con VIM, nos acostumbramos a no usarlo, disminuyendo así el tiempo que perdido.

Pero, la principal ventaja de VIM, es la productividad. Este editor de texto cuenta con 3 modos, a los que accedemos pulsando una sola tecla de nuestro teclado:

- Modo normal: Al pulsar la tecla escape (por defecto) accedemos a este modo, en el cual, las teclas de nuestro teclado cambiaran su funcionalidad, es decir, ahora al pulsar sobre las letras ejecutaremos

un comando directamente.

- Modo insertar: Se accede al pulsar la tecla «I» cuando estamos en modo normal. En este modo es donde escribiremos de forma natural nuestro código.

- Modo visual: Accedemos pulsando la letra «V». Este modo lo usaremos para seleccionar, de forma visual, el texto.

Estando en modo normal, tenemos a nuestra disposición toda una serie de comandos para movernos por el código, editar, copiar, borrar y un largo etc. Estos comandos los podemos combinar entre sí, de modo que, con práctica, seremos capaces de aumentar nuestra productividad abismalmente.

A todas estas ventajas hay que sumarle la posibilidad de añadir plugins creados por la comunidad, con lo cual podemos aumentar sus funcionalidades.

¿Y qué hay de las deventajas?

Este editor no tiene interfaz gráfica, de modo que puede ser un caos. Otra desventaja es que todos los comandos vistos anteriormente te los tienes que aprender de memoria si realmente los quieres sacar provecho. VIM tiene una curba de dificultad muy alta de tal forma que si es la primera vez que lo utilizas, te va a costar mucho aprender como funciona todo.

Atajos y comandos básicos de VIM

Recuerdo que para poder utilizar estos comandos, debemos estar en modo normal.

Letra	Uso
h, j ,k, l	Para desplazarnos, en lugar de movernos con las flechas del teclado, nos movemos con estas letras (h: ← j: ↓ k: ↑ l: →)
w, b	La w para desplazar el cursor una palabra hacia adelante y la b para desplazarnos una palabra hacia atrás
0, $	El 0 sirve para desplazarnos hacia el inicio de la linea en la que nos encontremos y $ para movernos al final
^	Para movernos hasta el primer carácter no vacío de la linea
gg, G	gg para movernos al principio del documento y G para movernos al final

Letra	Uso
d	Este comando se utiliza después de usar uno de los anteriores, elimina desde la posicion de nuestro cursor hasta el desplazamiento indicado
dd	Elimina toda la linea sobre la que se encuentre el cursor
y,p	y para copiar el texto hasta el desplazamiento que le indiquemos con los comandos anteriores, p para pegar
. (punto)	Con este comando repetiremos el comando ejecutado anteriormente

Hay muchisímos más comandos, pero he querido poner algunos de los principales, te animo a que sigas descubriendo nuevos comandos. Decir que los comandos anteriores se pueden combinar con números, es decir, si pulsamos sobre el 7 y luego sobre la w, nos moveremos 7 lineas hacia adelante. Aparte de estos atajos, hay que sumar, otras instrucciones que podemos escribimos:

:q Cierra vim.

:w Guardamos los cambios.

:e + «Nombre de la ruta» Para abrir el archivo que le indiquemos con la ruta.

:split Parte la pantalla verticalmente o horizontalmente para editar varios archivos a la vez.

:Ex Explorador de archivos.

:help Para ver el archivo de ayuda de Vim con todos los comandos.

Cómo editar el archivo de configuración de vim

A VIM le podemos cambiar muchos de sus parámetros y opciones por defecto, para ello, teniendo VIM abierto, ejecutamos el comando:

```
1 :e ~/.vimrc
```

A continuación dejo algunas de las configuraciones que tengo yo puestas en mi archivo:

```
1     " Vim por defecto crea archivos de backup pero son muy molestos con
          estas opciones los podemos quitar
2
3     set nobackup
```

```
4     set nowritebackup
5     set noswapfile
6
7     " Para hacer que VIM tabule automaticamente
8     set autoindent
9
10    "Para hacer que VIM utilize por defecto 2 espacios para lenguajes
          que se ven mejor así
11    autocmd FileType html,css,sass,scss,javascript setlocal sw=2 sts=2
12    autocmd FileType json setlocal sw=2 sts=2
13    autocmd FileType ruby,eruby setlocal sw=2 sts=2
14    autocmd FileType yaml setlocal sw=2 sts=2
15
16    "Para que muestre a la izquierda los números de linea relativos,
          muy útil para cuando introducimos comandos
17    set relativenumber
18
19    "Para que resalte los paréntesis y los corchetes
20    set showmatch
```

Como siempre, hay muchas más configuraciones, dependiendo de los gustos de cada uno.

Plugins para VIM

Para instalar plugins en VIM, lo mejor es utilizar un gestor de plugins, yo recomiendo vim-plug: https://github.com/junegunn/vim-plug. Una vez instalado, simplemente añades en tu archivo .vimrc:

```
1     call plug#begin('~/.vim/plugged')
2
3         "Entre estas dos líneas añades Plug + 'nombre del plugin que
              quieres instalar'
4
5     call plug#end()
```

Plugins que yo utilizo:

- NERDTree (https://github.com/scrooloose/nerdtree) Explorador de arhivos para vim, aunque últimamente lo utilizo menos
- Ctrlp (https://github.com/kien/ctrlp.vim) Permite buscar archivos en nuestro proyecto mediante el nombre que le indiquemos.
- Buftabline (https://github.com/ap/vim-buftabline) Añade una barra arriba con los buffers que tienes abiertos.

- Emmet-vim (https://github.com/mattn/emmet-vim) Este plugin viene muy bien para los desabolladores web, permite crear varios elementos del HTML de golpe.

Conclusiones

VIM es muy pontente en el sentido de la productividad, aunque es muy dificil de aprender. Tiene muchísimos plugins y configuraciones, pero no es un IDE, para algunos lenguajes se nos queda corto incluso con plugins. Habrá a muchas personas a las que no compese el esfuerzo de aprender a manejar solo el editor de texto, pero si lo dominas los resultados son increíbles.

WebGL es una API basada en OpenGL mediante la cual podemos crear gráficos o pequeños juegos en 3D para el desarrollo web. Para ello, necesitamos que el navegador que estemos utilizando soporte WebGL (actualmente la mayoria lo soportan). Si quieres saber más información puedes consultar la wikipedia (https://es.wikipedia.org/wiki/WebGL), o puedes echar un vistazo a estos ejemplos, por si te sirven de inspiración: https://www.chromeexperiments.com/webgl.

Para esta guia, vamos a utilizar Dart para la programación de un simple escenario en 3D, que nos puede servir como base para futuros proyectos. WebGL también se puede utilizar con JavaScript o con otros lenguajes pensados para la web.

Primeros pasos con Dart y WebGL

Lo primero es descargar Dart y su sdk para poder empezar. Vamos a la página de Dart y lo instalamos siguiendo los pasos que aparecen para cada sistema operativo https://www.dartlang.org/install

Una vez lo tenemos instalado, tenemos que crear un proyecto Dart, afortunadamente el IDE que estoy usando (WebStorm), puede generar proyectos Dart, en caso de usar otro IDE tendríamos que generarlo manualmente. Ahora, abrimos nuestro archivo index.html y creamos el canvas en el que se dibujaran los elementos, para ello añadimos la siguiente linea dentro del body:

```
1 <canvas id=”game” width=”900px” height=”500px”></canvas>
```

Notar que he añadido dentro de la etiqueta del canvas, el id para poder identificarlo en nuestro código en dart, así como, la anchura (width) y la altura (height), aunque estas últimas se pueden añadir a posteriori.

El siguiente paso es añadir código al archivo dart. Lo primero es importar lo que vamos a utilizar:

```
1 import 'dart:html' as html;
2 import 'dart:web_gl' as GL;
3
4 GL.RenderingContext gl;
```

Añadimos la funcion main() si no la tenemos ya. Dentro vamos a inicializar WebGL en el canvas que creamos anteriormente, en caso de que el usuario no disponga de WebGL, hacemos una alerta. En

caso contrario, llamamos a la funciona start() que es la que vamos a utilizar para empezar a cargar WebGL.

```
void main() {
 var canvas = html.querySelector("#game");

  gl = canvas.getContext("webgl");

  if(gl == null) gl = canvas.getContext("experimental-webgl");

  if(gl != null){
    start();
  }else{
    html.window.alert("Tu navegador no soporta WebGL");
  }
}
```

Tras cargar WebGL en la variable gl, comprobamos si es null, en este caso intentamos cargar experimental-webgl (para navegadores Chrome).

Ahora toca implementar la funcion start().

```
void start(){
  gl.clearColor(0.3,0,3.0,1.0);
  gl.enable(GL.RenderingContext.DEPTH_TEST);
  gl.clear(GL.COLOR_BUFFER_BIT|GL.DEPTH_BUFFER_BIT);
}
```

Con la funcion clearColor especificamos que color se va a cargar cuando llamemos a gl.clear(). La siguiente instruccion sirve para habilitar la prueba de WebGL de profundidad. Por último limpiamos el buffer de bits. Si queremos cambiar la resolucion de renderizado tenemos que añadir la siguiente instruccion:

```
gl.viewport(0, 0, 900, 500);
```

Esto ajustará la resolucion al ancho y el alto que le especifiquemos, en este caso 900x500, nuestro ancho y alto del lienzo. Lo mejor, en este caso, es añadir una variable para el ancho y el alto de nuestro lienzo para usarlo directamente en esta instrucción.

El código al completo del archivo main.dart, quedaría de la siguiente manera

```
import 'dart:html' as html;
import 'dart:web_gl' as GL;

```

```
4  GL.RenderingContext gl;
5
6  void main() {
7   var canvas = html.querySelector("#game");
8
9    gl = canvas.getContext("webgl");
10
11   if(gl == null){
12     gl = canvas.getContext("experimental-webgl");
13
14   }
15
16   if(gl != null){
17     start();
18   }else{
19     html.window.alert("Tu navegador no soporta WebGL");
20   }
21 }
22
23 void start(){
24   gl.clearColor(0.3,0,3.0,1.0);
25   gl.enable(GL.RenderingContext.DEPTH_TEST);
26   gl.clear(GL.COLOR_BUFFER_BIT|GL.DEPTH_BUFFER_BIT);
27   gl.viewport(0, 0, 900, 500);
28 }
```

Si ahora ejecutas el proyecto y vas al navegador, si puedes ver un cuadrado de color azul, es que lo has hecho bien.

Si queremos subir nuestro proyecto Dart a un servidor, tenemos que compilarlo en archivos con extensión .js (JavaScript). Dart pone a nuestra disposición un comando para generar estos archivos. Si tenemos instalado correctamente Dart en nuestro ordenador, tendremos disponibles los comandos de Dart. Solo tenemos que escribir en la consola el siguiente comando

```
1  dart2js --out=test.js test.dart
```

Donde test.js es el nombre del archivo JavaScript que queremos generar y test.dart es nuestro archivo Dart, en nuestro caso main.dart.

Para más información sobre estos comandos visita la web de Dart https://webdev.dartlang.org/tools/dart2js

De momento no tenemos nada, simplemente el lienzo, en articulos posteriores cargaremos los sha-

ders, y empezaremos a dibujar las figuras.

Introducción y primeros pasos

Bosstrap es un framework creado con la finalidad de crear páginas webs reponsive, es decir, páginas en las cuales el diseño se adapta en función del dispositivo desde el cual abrimos la página web.

¿Por qué usar Bootstrap?

- . Facilidad de uso. Simplemente usando sus clases
- . Responsive. Con las nuevas tecnologías queremos que nuestra página web se vea bien en todos los dispositivos.
- . Personalizable. Al descargarlo podremos elegir que elementos queremos dependiendo de nuestras necesidades
- . Gran comunidad. Este framework está muy extendido y si tenemos un problema podremos encontrar mucha información en Internet.

Descarga y configuración de Bootstrap

Para usar Bootstrap en nuestro proyecto tenemos dos formas:

. Primera opcion: Descargamos Bootstrap desde su página oficial http://getbootstrap.com/ y descargamos JQuery https://jquery.com/. Metemos estos archivos que acabamos de descargar en una carpeta dentro del proyecto en el cual queremos usar Bootstrap. Ahora tenemos que llamar a Bootstrap desde el archivo index.html o en su defecto el archivo donde tenemos todos los estilos por defecto.

```
1  <head>
2
3     <link href="css/bootstrap.min.css" rel="stylesheet" media="screen">
          <! Inicializamos Bootstrap con la ruta en la que se encuentra-.
         >
4     <meta name="viewport" content="width=device-width, initial-scale=1"
         > <-! Con esto garantizamos que se vea correctamente en todos
         los dispositivos móviles ->
5
6     <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/
         jquery.min.js"></script> <-! Llamamos antes a JQuery ->
7        <script src="js/bootstrap.min.js"></script> <-! Llamamos al
            JavaScript de Bootstrap ->
8  </head>
```

. Segunda opción: CDN. Con este método no tenemos que descargar Bootstrap. Simplemente inclimos el enlace del CDN para que se descargue solo.

```
1  <head>
```

```
2       <!-- Latest compiled and minified CSS -->
3     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
          bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-
          BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u
          " crossorigin="anonymous">
4
5     <!-- Optional theme -->
6     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
          bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-
          rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp
          " crossorigin="anonymous">
7
8     <!-- JQuery -->
9      <script type="text/javascript" src="https://code.jquery.com/jquery
          -2.1.1.min.js"></script>
10
11    <!-- Latest compiled and minified JavaScript -->
12    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/
          bootstrap.min.js" integrity="sha384-
          Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa
          " crossorigin="anonymous"></script>
13
14  </head>
```

Elementos de Bootstrap

Para usar los elementos de Bootstrap tenemos que usar sus clases dentro de los elementos del html. Aquí los elementos más importantes de Bootstrap

Grid

Estos son los elementos que, personalmente, me parecen más utilies y son los que más utilizo. Bootstrap tiene un sistema para crear columnas, las cuales adaptan su ancho dependiendo del dispositivo. Para usar este sistema tenemos que ponerle la clase «row» a un elemento div del html. Acabamos de crear la fila de una columna, pero esta vacía. Para meter columnas dentro de las filas tenemos que meter elementos div con la clase «col»

```
1  <div class="container"><-! Este elemento es un contenedor. Podemos
       meter elementos dentro de este contenedor para que se les aplique un
        margen y se centren ->
2    <div class="row">
3      <div class="col">
4        1 of 2
5      </div>
```

```
    <div class="col">
      1 of 2
    </div>
  </div>
  <div class="row">
    <div class="col">
      1 of 3
    </div>
    <div class="col">
      1 of 3
    </div>
    <div class="col">
      1 of 3
    </div>
  </div>
</div>
```

```
<div class="col m6 red lighten-4">
  1 of 2
</div>
<div class="col m6 purple lighten-4">
  1 of 2
</div>
```

```
<div class="col m4 amber lighten-4">
  1 of 3
</div>
<div class="col m4 cyan lighten-4">
  1 of 3
</div>
<div class="col m4 teal lighten-4">
  1 of 3
</div>
```

El primer row es una fila con 2 columnas de igual tamaño y el segundo row es una fila con 3 columnas iguales. Como ves si queremos crear columnas de igual tamaño basta con meter elementos col dentro de las filas. ¿Qué pasa si queremos crear columnas con un tamaño diferente? Para crearlas tenemos que especificar el tamaño que queremos. Importante: Las filas de Bootstrap tienen un tamaño de 12 unidades, es decir, si creamos columnas más pequeñas no llegarán a cubrir el ancho de la fila, y si las creamos más grandes saltarán a la siguiente fila.

```
<div class="container">
```

```
  <div class="row">

    <div class="col-3">
      1 of 3
    </div>
    <div class="col-6"> <!- Esta columna será más ancha que las otras
       dos ->
      2 of 3 (más grande)
    </div>
    <div class="col-3">
      3 of 3
    </div>

  </div>
</div>
```

```
<div class="col m3 amber lighten-4">
  1 of 3
</div>
<div class="col m6 cyan lighten-4">
  2 of 3 (más grande)
</div>
<div class="col m3 teal lighten-4">
  3 of 3
</div>
```

También podemos definir especificamente que tamaño queremos que tengan las columnas dependiendo del dispositivo. Para ello existen varias clases en Bootstrap:

- - xs (para móviles)
- - sm (para tablets)
- - md (para ordenadores)
- - lg (para ordenadores con pantalla más grande)

Por ejemplo, si queremos que una determinada columna ocupe todo el ancho en dispositivos moviles, pero queremos que ocupe menos en ordenadores, tenemos que usar la clase «col-xs-12 col-md-6»

Botones

Para añadir un botón en Bootstrap tenemos que usar la clase «btn». Podemos usar varios tipos dependiendo del uso que le vayamos a dar. El boton por defecto es «default»

```
<!-- Botón standart -->
<button type="button" class="btn btn-default">Default</button>
```

```
3
4  <!-- Botón primario -->
5  <button type="button" class="btn btn-primary">Primary</button>
6
7  <!-- Botón de exito -->
8  <button type="button" class="btn btn-success">Success</button>
9
10 <!-- Botón de información  -->
11 <button type="button" class="btn btn-info">Info</button>
12
13 <!-- Botón de aviso -->
14 <button type="button" class="btn btn-warning">Warning</button>
15
16 <!-- Botón de peligro  -->
17 <button type="button" class="btn btn-danger">Danger</button>
18
19 <!-- Botón con énfasis en un link  -->
20 <button type="button" class="btn btn-link">Link</button>
```

Para aumentar o disminuir el tamaño del botón tenemos que añadir la clase .btn-lg, .btn-sm, o .btn-xs dependiendo del tamaño que queramos.

Imágenes

Para conseguir que las imágenes sean responsive solo tenemos que añadir la clase «.img-responsive». Bootstrap se encargará de aumentar o disminuir el tamaño de la imagen según lo necesite.

Otros elementos

Añadir colores de fondo a un elemento directamente con una clase de Bootstrap

```
1  <p class="bg-primary">...</p>
2  <p class="bg-success">...</p>
3  <p class="bg-info">...</p>
4  <p class="bg-warning">...</p>
5  <p class="bg-danger">...</p>
```

...

...

...

...

...

Forzar a elementos a que se visualicen o no de una forma muy comoda:

```
<div class="show">...</div>
<div class="hidden">...</div>
```

Centrar contenido

```
<div class="center-block">...</div>
```

Conclusiones

Bootstrap es un framework muy potente y muy extendido en la actualidad. Tiene multitud de elementos (muchos no los he puesto en este artículo) a nuestra disposición que nos facilitarán enormemente la vida para desarrollar webs responsive adatadas a las nuevas tecnologías. Si quieres seguir aprendiedo acerca de este framework te invito a que busques más información en su página web http://getbootstrap.com/css/

Antes de empezar con este artículo, es recomendable leer el episodio anterior, lo puedes encontrar aqui: http://frostqui.github.io/tutorial-dart-webgl Bien, entendido este podemos empezar. Este tutorial está basado en los tutoriales de NeHe OpenGL.

Creando el proyecto

Lo primero que tenemos que saber es que aunque queramos dibujar renderizar en 2D, no tenemos que olvidar que WebGL dibuja los objetos en un espacio 3D.

Como en el anterior tutorial, empezamos creando el proyecto dart con nuestro IDE, en mi caso con WebStorm. Abrimos el achivo index.html y añadimos el canvas donde dibujaremos las figuras.

```
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale
        =1.0">
    <meta name="scaffolded-by" content="https://github.com/google/
        stagehand">
    <title>untitled</title>
    <link rel="stylesheet" href="styles.css">
    <script defer src="main.dart" type="application/dart"></script>
    <script defer src="packages/browser/dart.js"></script>
</head>

<body>

```

```
    <canvas id="game" width="900px" height="500px"></canvas>

</body>
</html>
```

Tras esto, abrimos el archivo main.dart, inicializamos importamos webgl y lo incializamos con el canvas. Además, he importado la librería para tener datos tipados, y la librería para crear vectores.

```
import 'dart:html' as html;
import 'dart:web_gl' as GL;
import 'dart:typed_data';
import 'package:vector_math/vector_math.dart';

void main() {
 var canvas = html.querySelector("#game");

  gl = canvas.getContext("webgl");

  if(gl == null){
    gl = canvas.getContext("experimental-webgl");
    _initShaders();
    _initBuffers();
  }

  if(gl != null){
    start();
  }else{
    html.window.alert("Tu navegador no soporta WebGL");
  }
}

void start(){
  gl.clearColor(1, 1, 1, 1.0);
  gl.enable(GL.RenderingContext.DEPTH_TEST);

}
```

Para importar la librería de los vectores tienes que añadir esto en tu archivo pubspec.yaml y ejecutar el comando pub get

```
dependencies:
  vector_math: any
```

Creando los buffers

Ahora vamos a escribir nuestra función para crear los buffers que sean necesarios. Vamos a utilizar los buffers para almacenar las figuras que tienen que ser dibujadas. Para ello declaramos dos variables de tipo buffers

```
GL.Buffer _triangleVertexPositionBuffer;
GL.Buffer _squareVertexPositionBuffer;
```

Toca crear la funcion para inicializar los buffers. Para el triángulo, vamos a almacenar la posición de sus vertices, de esta forma, si hace falta que se vuelva a renderizar, será más eficiente porque ya tiene su posición. Hacemos lo mismo para el cuadrado.

```
void _initBuffers() {
  // variable to store verticies
  List<double> vertices;

  // Creamos el triangulo
  _triangleVertexPositionBuffer = gl.createBuffer();
  gl.bindBuffer(GL.RenderingContext.ARRAY_BUFFER,
      _triangleVertexPositionBuffer);

  // Rellenamos el buffer con los vértices
  vertices = [
    0.0,  1.0,  0.0,
    -1.0, -1.0,  0.0,
    1.0, -1.0,  0.0
  ];
  gl.bufferDataTyped(GL.RenderingContext.ARRAY_BUFFER, new Float32List.
      fromList(vertices), GL.RenderingContext.STATIC_DRAW);

  // Creamos el cuadrado
  _squareVertexPositionBuffer = gl.createBuffer();
  gl.bindBuffer(GL.RenderingContext.ARRAY_BUFFER,
      _squareVertexPositionBuffer);

  // Rellenamos el buffer con los vértices
  vertices = [
    1.0,  1.0,  0.0,
    -1.0,  1.0,  0.0,
    1.0, -1.0,  0.0,
    -1.0, -1.0,  0.0
  ];
```

```
28    gl.bufferDataTyped(GL.RenderingContext.ARRAY_BUFFER, new Float32List.
          fromList(vertices), GL.RenderingContext.STATIC_DRAW);
29
30  }
```

Dibujando las figuras

Ahora, vamos a definir la función que dibujará la escena. Lo primero, es usar la funcion clear de WebGl para limpiar el área que vamos a dibujar. Para nuestra escena vamos a necesitar una Matriz para almacenar las posiciones y las transformaciones, así que creamos una llamada _mvMatrix, y le aplicamos una transformación para movernos al centro de la escena. El siguiente paso es dibujar las figuras que hay en el buffer. Por último, una vez dibujada cada figura, llamamos al método _setMatrixUniforms() para que se almacene en la tarjeta gráfica el modelo de la vista. La función quedaría de la siguiente manera:

```
1  render([double time = 0.0]) {
2    gl.clear(GL.RenderingContext.COLOR_BUFFER_BIT | GL.RenderingContext.
         DEPTH_BUFFER_BIT);
3
4    _mvMatrix = new Matrix4.identity();
5    _mvMatrix.translate(new Vector3(-1.5, 0.0, -7.0));
6
7    // Dibujamos el triángulo
8    gl.bindBuffer(GL.RenderingContext.ARRAY_BUFFER,
         _triangleVertexPositionBuffer);
9    gl.vertexAttribPointer(_aVertexPosition, _dimensions, GL.
         RenderingContext.FLOAT, false, 0, 0);
10   _setMatrixUniforms();
11   gl.drawArrays(GL.RenderingContext.TRIANGLES, 0, 3); // Triángulo.
         Empieza en 0. 3 en total
12
13   // Dibujamos el cuadrado
14   _mvMatrix.translate(new Vector3(3.0, 0.0, 0.0));
15
16   gl.bindBuffer(GL.RenderingContext.ARRAY_BUFFER,
         _squareVertexPositionBuffer);
17   gl.vertexAttribPointer(_aVertexPosition, _dimensions, GL.
         RenderingContext.FLOAT, false, 0, 0);
18   _setMatrixUniforms();
19   gl.drawArrays(GL.RenderingContext.TRIANGLE_STRIP, 0, 4); // Cuadrado.
          Empieza en 0. 4 en total
20 }
21
```

```
22 void _setMatrixUniforms() {
23   Float32List tmpList = new Float32List(16);
24
25   _pMatrix.copyIntoArray(tmpList);
26   gl.uniformMatrix4fv(_uPMatrix, false, tmpList);
27
28   _mvMatrix.copyIntoArray(tmpList);
29   gl.uniformMatrix4fv(_uMVMatrix, false, tmpList);
30 }
```

Creación de los shaders

Antes de crear los shaders vamos a entender qué son. Los shader son una forma de añadir color e iluminación mediante código a figuras o píxeles. Esto se calcula atomáticamente antes de dibujar las figuras. Estos shaders también permiten otros efectos como animaciones, distorisiones o desenfoques. Los shaders también son útiles para hacer los cálculos necesarios con la velocidad de la GPU.

Lo primero que hacemos es inicializar un par de Strings. Uno lo usaremos para definir la animación y el otro para definir el color. Lo siguiente que hacemos es compilar los shaders y aplicarlos a la escena. Por último comprobamos que el proceso ha salido bien.

```
1  void _initShaders() {
2    // VertexShader para la creación de la animación
3
4    String vsSource = """
5      attribute vec3 aVertexPosition;
6      uniform mat4 uMVMatrix;
7      uniform mat4 uPMatrix;
8      void main(void) {
9          gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0)
               ;
10     }
11     """;
12
13   // Fragment shader para la creación de la animacion de los colores
14
15
16   String fsSource = """
17     precision mediump float;
18     void main(void) {
19         gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0);
20     }
21     """;
```

```
22
23    // Compilacion de los dos shaders
24    _vs = gl.createShader(GL.RenderingContext.VERTEX_SHADER);
25    gl.shaderSource(_vs, vsSource);
26    gl.compileShader(_vs);
27
28
29    _fs = gl.createShader(GL.RenderingContext.FRAGMENT_SHADER);
30    gl.shaderSource(_fs, fsSource);
31    gl.compileShader(_fs);
32
33    // Metiendo los shaders en el programa
34    _shaderProgram = gl.createProgram();
35    gl.attachShader(_shaderProgram, _vs);
36    gl.attachShader(_shaderProgram, _fs);
37    gl.linkProgram(_shaderProgram);
38    gl.useProgram(_shaderProgram);
39
40    //Comprobamos que los shaders han sido añadidos correctamente
41
42    if (!gl.getShaderParameter(_vs, GL.RenderingContext.COMPILE_STATUS))
          {
43      print(gl.getShaderInfoLog(_vs));
44    }
45
46    if (!gl.getShaderParameter(_fs, GL.RenderingContext.COMPILE_STATUS))
          {
47      print(gl.getShaderInfoLog(_fs));
48    }
49
50    if (!gl.getProgramParameter(_shaderProgram, GL.RenderingContext.
          LINK_STATUS)) {
51      print(gl.getProgramInfoLog(_shaderProgram));
52    }
53
54    _aVertexPosition = gl.getAttribLocation(_shaderProgram, "
          aVertexPosition");
55    gl.enableVertexAttribArray(_aVertexPosition);
56
57    _uPMatrix = gl.getUniformLocation(_shaderProgram, "uPMatrix");
58    _uMVMatrix = gl.getUniformLocation(_shaderProgram, "uMVMatrix");
59  }
```

Si todo ha salido bien, al ejecutar nuestro código nos encontraremos con las dos figuras

El código de todo el archivo main.dart lo puedes encontrar en este github:

https://github.com/martinsik/dart-webgl-tutorials/blob/master/lib/lesson-01/lesson-01.dart

¡Saludos! En el día de hoy vamos a ver en que consiste Electron, y para ello vamos a desarrollar una app muy sencilla para la gestión de notas y recordatorios usando NodeJS y MongoDB.

Introducción

Antes de empezar con el ejemplo que vamos a desarrollar vamos a ver que es Electron. Electron es un framework desarrollado con la intención de facilitar la vida enormemente al programador. Entre su ventajas, destaca la de poder crear apps de escritorio multiplataforma, es decir, para distintos sistemas operativos, usando tecnologías web. Si manejas bien los lenguajes del desarrollo web como HTML, CSS y JavaScript, con este framework, podrás crear apps sin tener que aprender un lenguaje nuevo. Al utilizar HTML y CSS para representar los datos, es más sencillo crear apps responsive. Una vez entendidas las características de Electron vamos a crear un ejemplo sencillo para ver mejor como funciona.

Instalación de Electron y de todos los elementos que vamos a necesitar

Antes de instalar Electron, tenemos que instalar Node y NPM en nuestro equipo. Nos dirigimos a su página, lo descargamos y lo instalamos: https://nodejs.org/en/. NPM lo pudes bajar e instalar desde aquí: https://www.npmjs.com/ Para instalar Electron, tenemos que abrir la terminal para introduccir el siguiente comando:

```
1 npm install electron --save-dev
```

Bien, para manejar las notas y recordatorios de nuestra app, vamos a utilizar ExpressJS junto con MongoDB como base de datos. Para el frontend, vamos a utilizar AngularJS, es decir, vamos a utilizar el stack MEAN. Para el que no lo conozca, el stack MEAN consiste en desarrollar páginas webs, utilizando solo el lenguaje JavaScript, es decir, AngularJS, NodeJS y MongoDB, todos estos frameworks utilizan la sintaxis de JavaScript. Ahora vamos a descargar o clonar el siguiente proyecto:

https://github.com/theallmightyjohnmanning/electron-express

Una vez descargado, ejecutamos el comando npm install para que instale todas las dependencias. Si hasta aquí todo ha ido bien, si ejecutamos el comando npm start nos debería abrir la aplicación.

Desarrollo de la aplicación

Para insertar los datos en la base de datos MongoDB vamos a utilizar Moongose para gestionarlo de una manera más sencilla. Para ello tenemos que añadir una dependencia nueva en el archivo packcage.json

```
”mongoose”: ”^4.4.12”
```

Si ahora, vuelves a hacer npm install, debería descargarse Moongose correctamente. Para conectar nuestra app con la base de datos en MongoDB tenemos que modificar el fichero app.js para dejarlo parecido a esto:

```
module.exports = () => {
  // Load The FS Module & The Config File
  fs = require('fs');

  // Load The Path Module
  path = require('path');
  mongoose = require('mongoose');


 mongoose.connect('AQUI METE LA URL DE LA BASE DE DATOS MONGODB');
    // connect to mongoDB database on modulus.io


  config = JSON.parse(fs.readFileSync('config.json'));

  // Load Express Module
  express = require('express');
  app   = express();

  // Load Body Parser Module
  bodyParser = require('body-parser');
  app.use(bodyParser.json());
  app.use(bodyParser.urlencoded({extended: false}));

  // Load Express Handlebars Module & Setup Express View Engine
  expressHandlebars = require('express-handlebars');
  app.set('views', __dirname+'/views/'); // Set The Views Directory
  app.engine('html', expressHandlebars({ // Setup View Engine
     Middleware

    layoutsDir:__dirname + '/views/layouts',
    defaultLayout: 'main',
    extname: '.html',
    helpers: {
        section: function(name, options) {

```

```
35              if(!this._sections) {
36                  this._sections = {};
37              }
38
39              this._sections[name] = options.fn(this);
40              return null;
41          }
42      }
43  }));
44  app.set('view engine', 'html');
45
46  // Load Express Session Module
47  session = require('express-session');
48  app.use(session({ // Setup Session Middleware
49
50    secret: config.session.secret,
51    saveUninitialized: true,
52    resave: true
53  }));
54
55  // Load Connect Flash Module
56  flash = require('connect-flash');
57  app.use(flash());
58  app.use(function(req, res, next) { // Setup Global Flash Message
        Middleware
59
60    res.locals.success_msg  = req.flash('success_msg');
61    res.locals.error_msg    = req.flash('error_msg');
62    res.locals.error            = req.flash('error');
63    res.locals.user             = req.user || null;
64    next();
65  });
66
67  // Load The Bcrypt Module
68  bcrypt = require('bcryptjs');
69
70  // Setup Globally Included Directories
71  app.use(express.static(path.join(__dirname, '/../bower_components/'))
        );
72  app.use(express.static(path.join(__dirname, '/../node_modules/')));
73  app.use(express.static(path.join(__dirname, '/../controllers/')));
74  app.use(express.static(path.join(__dirname, '/../public/')));
75
```

```
76    // Load Available Modules For Dependancy Injection Into Models &
          Routes
77    modules = {
78      app: app,
79      bcrypt: bcrypt,
80      bodyParser: bodyParser,
81      config: config,
82      express: express,
83      expressHandlebars: expressHandlebars,
84      flash: flash,
85      fs: fs,
86      path: path,
87      session: session
88    };
89
90    // Setup Globally Included Routes
91    fs.readdirSync(path.join(__dirname, 'routes')).forEach(function(
          filename) {
92
93      if(~filename.indexOf('.js'))
94          require(path.join(__dirname, 'routes/'+filename))(modules);
95    });
96
97    // Start The HTTP Server
98    app.listen(config.server.port, config.server.host);
99  }
```

Ahora vamos a crear nuestro modelo para las notas y recordatorios. Este modelo lo puedes personalizar con los parámetros que necesites, en mi caso he creado un archivo dentro de la carpeta app/models/ llamado todo.js

```
1  var mongoose = require('mongoose');
2
3  module.exports = mongoose.model('Todo', {
4      text : String,
5      done: { type: Boolean, default: false },
6      created: { type: Date, default: Date.now },
7      type: { type: String, default:"Class" },
8
9   });
```

Ahora vamos a modificar el archivo pages.js que está situado dentro de la carpeta app/routes. En este

archivo es donde gestionaremos las llamadas a la API Rest que estamos creando.

```
1
2   = require('../models/todo');
3  module.exports = function() {
4
5      app.get('/api/todos', function(req, res) {
6
7          // use mongoose to get all todos in the database
8          Todo.find(function(err, todos) {
9
10             // if there is an error retrieving, send the error. nothing
                   after res.send(err) will execute
11             if (err)
12                 res.send(err)
13
14             res.json(todos); // return all todos in JSON format
15         });
16     });
17
18     app.post('/api/todos', function(req, res) {
19
20         // create a todo, information comes from AJAX request from
               Angular
21         Todo.create({
22             text : req.body.text,
23             done : false
24         }, function(err, todo) {
25             if (err)
26                 res.send(err);
27
28             // get and return all the todos after you create another
29             Todo.find(function(err, todos) {
30                 if (err)
31                     res.send(err)
32                 res.json(todos);
33             });
34         });
35
36     });
37
38
39     app.delete('/api/todos/:todo_id', function(req, res) {
```

```
40          Todo.remove({
41              _id : req.params.todo_id
42          }, function(err, todo) {
43              if (err)
44                  res.send(err);
45
46              // get and return all the todos after you create another
47              Todo.find(function(err, todos) {
48                  if (err)
49                      res.send(err)
50                  res.json(todos);
51              });
52          });
53      });
54
55      app.patch('/api/todos/:todo_id', function(req, res) {
56
57          Todo.findById(req.params.todo_id, function (err, todo) {
58      // Handle any possible database errors
59      if (err) {
60          res.status(500).send(err);
61      } else {
62          // Update each attribute with any possible attribute that may
                have been submitted in the body of the request
63          // If that attribute isn't in the request body, default back to
                whatever it was before.
64
65
66          todo.done = !todo.done;
67
68          // Save the updated document back to the database
69          todo.save(function (err, todo) {
70              if (err) {
71                  res.status(500).send(err)
72              }
73
74
75              Todo.find(function(err, todos) {
76                  if (err)
77                      res.send(err)
78
79                  res.json(todos);
80              });
```

```
81
82
83          });
84
85
86      }
87  });
88
89
90      });
91
92
93  app.get('*', function(req, res) {
94          res.sendfile('./public/index.html'); // load the single view
                file (angular will handle the page changes on the front-end)
95      });
96
97  }
```

Básicamente con esta API podemos gestionar las llamadas de GET, DELETE, y POST para crear, modificar y eliminar notas.

Ahora vamos a gestionar la parte del frontend, es decir, la representación de los datos. En el archivo core.js creamos las siguientes funciones:

```
1  var scotchTodo = angular.module('scotchTodo', []);
2
3  function mainController($scope, $http) {
4      $scope.formData = {};
5
6      // when landing on the page, get all todos and show them
7      $http.get('/api/todos')
8          .success(function (data) {
9
10             angular.forEach(data, function (value, key) {
11
12                 value.created = moment(value.created).fromNow();
13
14
15
16
17             });
18
```

```
19              $scope.todos = data;
20              console.log(data);
21
22          })
23          .error(function (data) {
24              console.log('Error: ' + data);
25          });
26
27      // when submitting the add form, send the text to the node API
28      $scope.createTodo = function () {
29          $http.post('/api/todos', $scope.formData)
30              .success(function (data) {
31                  $scope.formData = {}; // clear the form so our user is
                        ready to enter another
32                  angular.forEach(data, function (value, key) {
33
34
35                      value.created = moment(value.created).fromNow();
36
37
38
39
40                  });
41
42                  $scope.todos = data;
43                  console.log(data);
44              })
45              .error(function (data) {
46                  console.log('Error: ' + data);
47              });
48      };
49
50      // delete a todo after checking it
51      $scope.deleteTodo = function (id) {
52          $http.delete('/api/todos/' + id)
53              .success(function (data) {
54
55                    angular.forEach(data, function (value, key) {
56
57                  value.created = moment(value.created).fromNow();
58
59
60
```

```
61
62              });
63
64
65                  $scope.todos = data;
66                  console.log(data);
67              })
68              .error(function (data) {
69                  console.log('Error: ' + data);
70              });
71      };
72
73
74
75      $scope.doneTodo = function (id) {
76          $http.patch('/api/todos/' + id)
77
78              .success(function (data) {
79
80                    angular.forEach(data, function (value, key) {
81
82                  value.created = moment(value.created).fromNow();
83
84
85
86
87              });
88
89
90                  $scope.todos = data;
91                  console.log(data);
92              })
93              .error(function (data) {
94                  console.log('Error: ' + data);
95              });
96      };
97
98  }
```

Lo que hago es crear 4 funciones. La primera funcion, lo que hace es llamar a la API para coger todas las notas. La segunda función se encarga de coger el texto que metemos con el teclado en el input de la app, y crea la nota, al crearla, utilizando la librería de moment.js lo que hago es añadir la fecha

actual en la nota que acabo de crear. Posteriormente vuelvo a listar todas las notas. La tercera función se encarga de eliminar una determinada nota. La última función sirve para marcar una nota o tarea como completada.

Ahora vamos a crear el archivo HTML, para ello modificamos el HTML de la carpeta public:

```
<!-- index.html -->
<!doctype html>
<!-- ASSIGN OUR ANGULAR MODULE -->
<html ng-app="scotchTodo">

<head>
    <!-- META -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Optimize mobile viewport -->
    <title>Task Box</title>
    <!-- SCROLLS -->
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">
    <!-- load bootstrap -->
    <link rel="stylesheet" href="css/index.css">
    <!-- load bootstrap -->
    <!-- SPELLS -->
    <script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
    <!-- load jquery -->
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.32/angular.min.js"></script>
    <!-- load angular -->
    <script src="core.js"></script>
    <script src="js/moment.js"></script>
</head>
<!-- SET THE CONTROLLER AND GET ALL TODOS -->

<body ng-controller="mainController">
    <div class="container">
        <br>
        <!-- HEADER AND TODO COUNT -->
        <h2 class="green">TO-DO <span class="label label-info">{{ todos.length }}</span></h2>
        <hr class="green_hr">
```

```
33          <!-- TODO LIST -->
34          <div id="todo-list">
35              <!-- LOOP OVER THE TODOS IN $scope.todos -->
36              <h4>
37                  <div ng-if="!todo.done" class="checkbox row" ng-repeat=
                        "todo in todos">
38                      <div class="col-xs-4"><label for="checkbox1">
39                        <div> {{ todo.text }} </div>
40
41                      </label>
42                      </div>
43                      <div class="col-xs-8">
44                          <div class="left-column">
45                              <small class="date" id="date">{{ todo.
                                    created }}</small>
46                              <buttom id="checkbox1" class="btn
                                    btn_delete" ng-click="doneTodo(todo._id)
                                    "> <span class="glyphicon glyphicon-ok
                                    green delete" aria-hidden="true"></span>
                                    </buttom>
47                              <buttom id="checkbox1" class="btn
                                    btn_delete" ng-click="deleteTodo(todo.
                                    _id)"> <span class="glyphicon glyphicon-
                                    remove red delete" aria-hidden="true"></
                                    span></buttom>
48                          </div>
49                      </div>
50                  </div>
51                  <hr class="green_hr">
52                  <div ng-if="todo.done" class="checkbox row" ng-repeat="
                        todo in todos">
53                      <div class="col-xs-4"><label for="checkbox1">
54
55                        <div  class="done"> {{ todo.text }} </div>
56                      </label>
57                      </div>
58                      <div class="col-xs-8">
59                          <div class="left-column">
60                              <small class="date" id="date">{{ todo.
                                    created }} {{ todo.type }}</small>
61                              <buttom id="checkbox1" class="btn
                                    btn_delete" ng-click="doneTodo(todo._id)
                                    "> <span class="glyphicon glyphicon-ok
```

```
                                    grey delete" aria-hidden="true"></span><
                                    /buttom>
62                                <buttom id="checkbox1" class="btn
                                    btn_delete" ng-click="deleteTodo(todo.
                                    _id)"> <span class="glyphicon glyphicon-
                                    remove red delete" aria-hidden="true"></
                                    span></buttom>
63                            </div>
64                        </div>
65                    </div>
66                </h4>
67            </div>
68        </div>
69        <!-- FORM TO CREATE TODOS -->
70        <div id="todo-form" class="row">
71            <form>
72                <div class="input-group">
73                    <input type="text" class="form-control input-lg text-
                        center input1" placeholder="TO-DO task" ng-model="
                        formData.text">
74                    <span class="input-group-btn">
75                            <button type="submit" class="btn btn-lg button1
                            " ng-click="createTodo()"><span class="
                            glyphicon glyphicon-plus" aria-hidden="true"
                            ></span></button>
76
77                    </span>
78                </div>
79                <!-- /input-group -->
80            </form>
81            <script src="js/index.js"></script>
82    </body>
83
84    </html>
```

Lo que hago en primer lugar, es mostrar una lista con las notas y tareas sin realizar. Para ello con Angular hago un bucle recorriendo todas las notas, si la nota no esta completada crea un div con todos sus parámetros. A la derecha de cada nota añado un botón para poder marcarla como completada con ng-click=«doneTodo(todo._id) y otro botón para eliminarla completamente de la base de datos ng-click=»deleteTodo(todo._id) Debajo, añado una lista con las tareas que ya hayan sido completadas, el mecanismo es el mismo, simplemente que si la nota ha sido completada se muestra. Por último, en la parte inferior un input para añadir el texto de la nota y el botón para añadir. También podemos añadir

CSS a nuestra app para darle estilos, en mi caso me he decantado por los siguiente estilos:

```
body{
        background-color: #1E1A25;
        color: white;



}

.green{

    color: #39FAB4;
}

.label-info{
    background-color: #39FAB4;
    border-radius: 50%;
    color:#1E1A25;

}

.green_hr{
    border-top: 1px solid #39FAB4;
}

#todo-form{

    position: fixed;
    width: 90%;
    bottom: 0;
    left: 50%;
    margin-left: -43.5%;

}

.button1{
    background-color: transparent;
    color: #39FAB4;
    border: none;
}

.button1:hover{
```

```
42      background-color: #39FAB4;
43      color: #1E1A25;
44  }
45
46  .input1{
47      background: transparent;
48      border-color: #39FAB4;
49      color: white;
50  }
51
52
53  .red{
54      color: #f45c42;
55
56  }
57
58  .grey{
59      color: #999;
60  }
61
62  .left-column{
63      margin-top: -6px;
64  }
65
66  .row{
67      margin-bottom: 30px;
68  }
69
70  .done{
71      color: #999;
72  }
```

Si todo ha ido bien, al ejecutar la app con npm start debería aparecer algo parecido a esto:

Y hasta aquí el artículo de hoy, te animo a que sigas investigando acerca de Electron ya que es una tecnología con mucho potencial, la cual puedes utilizar para hacer apps muy chulas en poco tiempo y de una manera muy sencilla.

¡Saludos! Hoy os traigo una lista seleccionada personalmente por mí, con las que considero que son las mejores o más utilies librerías Javascript para este 2017. Decir que no he incluido JQuery por ejemplo al igual que tantas otras porque ya son conocidas por todos.

Jquery FlexDataList

Empezamos con Jquery FlexDataList, una librería para incluir autocompletado en un página web. Para usarla tienes que seguir estos pasos:

Incluimos la ruta de la librería en la seccion HEAD de nuestro archivo HTML y sustituimos /path/to/ por la ruta donde descargemos flexdatalist:

```
1 <link href="/path/to/jquery.flexdatalist.min.css" rel="stylesheet" type
    ="text/css">
```

Inclimos el archivo JS de la librería también en la sección HEAD sustituyendo /path/to/ por la ruta donde tengamos flexdatalist:

```
1 <input type="text" class="flexdatalist" >
```

Ahora solo tenemos que poner el input en el html donde queramos que se autocomplete:

```
1 <input type="text" class="flexdatalist" >
```

Para cargar los datos tenemos que hacer una llamda a la función flexdatalist() seleccionando el input con JQuery.

Página oficial

AOS - Animate on scroll library

Se trata de una librería con la que, de una manera muy sencilla, podremos hacer animaciones muy chulas cuando los usuarios de nuestra página web hagan scroll en la página. La herramienta incluye muchas animaciones por lo sirve de gran ayuda para dejar impresionados a los visitantes de nuestra sitio web.

Para usarla podremos hacer uso del CDN, es decir, añadiendo estas líneas en la sección HEAD, será suficiente para hacer uso de estas animaciones.

```
1 <link href="https://cdn.rawgit.com/michalsnik/aos/2.1.1/dist/aos.css"
    rel="stylesheet">
2 <script src="https://cdn.rawgit.com/michalsnik/aos/2.1.1/dist/aos.js"><
    /script>
```

Ahora, para animar un elemento, tan solo tenemos que añadir el nombre de la animación en la propiedad data-aos del elemento que queramos animar, es decir por ejemplo:

```
<div data-aos="fade-up"></div>
```

Página oficial

Popper.js

Una herramienta que permite crear etiquetas con información sobre un determinado elemento de una página tan solo escribiendo una simple línea de código. Tiene muy buen suporte para AngularJS y para React y la librería no pesa mucho.

Para descargarla en nuestro proyecto, lo tenemos que hacer desde su Github

Una vez descargada, podemos crear la etiqueta desde javascript, es decir por ejemplo, para crear una etiqueta situada a la derecha de otro elemento:

```
var popper = new Popper(referenceElement, onPopper, {
    placement: 'right'
});
```

Página oficial

Bideo.js

Si alguna vez has querido añadir un vídeo de fondo a una web, y no sabes como o te parece muy lioso, con esta librería podrás añadirlo de una manera muy sencilla, compatible con todos los navegadores y responsive para todo tipo de pantallas.

Para descargarlo, lo puedes hacer desde su Github

Para usarlo podemos hacerlo con la etiqueta vídeo, por ejemplo:

```
<video id="background_video" loop muted></video>
```

Página oficial

Cleave.js

Hay veces que necesitamos, que cuando el usuario escriba una fecha, automáticamente se añadan barras sin necesidad de que el usuario las introduzca. O por ejemplo, queremos que se añadan espacios cuando un usuario escriba el número de teléfono. Para hacer esto, y tener que evitarnos escribir mucho código, podemos hacer uso de esta librería. Además de los ejemplos que he comentado, también tiene otros tipos de formateo en inputs.

Para descargarlo podemos hacerlo desde su repositorio de Github

Una vez lo metemos en la etiqueta HEAD

```
<script src="cleave.min.js"></script>
<script src="cleave-phone.{country}.js"></script>
```

(Sustituimos country por el páis que qureamos para el formateo de las tarjetas de crédito)

Para usarlo, por ejemplo, añadimos:

```
<input class="input-phone" type="text"/>
```

y luego en el javascript:

```
var cleave = new Cleave('.input-phone', {
    phone: true,
    phoneRegionCode: '{country}'
});
```

Página oficial

Granim.js

Con este sencillo script podremos hacer un fondo con degradado que vaya cambiando de color.

Para descargarlo, desde su Github

Y para usarlo simplemente creamos un elemento **canvas** en la página en la que queramos el fondo y añadimos, el siguiente código javascript:

```
var granimInstance = new Granim({
    element: '#canvas-basic',
    name: 'basic-gradient',
    direction: 'left-right',
```

```
    opacity: [1, 1],
    isPausedWhenNotInView: true,
    states : {
        "default-state": {
            gradients: [
                ['#AA076B', '#61045F'],
                ['#02AAB0', '#00CDAC'],
                ['#DA22FF', '#9733EE']
            ]
        }
    }
});
```

Donde canvas-basic es el id del canvas que añadimos anteriormente.

Página oficial

Elevator.js

Esta divertida librería añade un botón para volver arriba en la página, solo que añade música y sonidos de ascensor, no tiene mucho más.

Si la quieres, la puedes descargar desde Github

Y para usarla, como en el caso de este código, hacemos que el script se ejecute cuando pulsamos sobre el botón que creamos.

```
<div class="elevator-button">Back to Top</div>

<script>
// Elevator script included on the page, already.

window.onload = function() {
  var elevator = new Elevator({
    element: document.querySelector('.elevator-button'),
    mainAudio: '/src/to/audio.mp3',
    endAudio: '/src/to/end-audio.mp3'
  });
}
</script>
```

Página oficial

Jquery Izimodal

Si no quieres perder mucho el tiempo haciendo y diseñando modals, te recomiendo este plugin. Tiene muchos tipos prediseñados, como formularios de login y registro, notificaciones, alertas, etc.

Si te gustaría usarlo, lo mejor es bajartelo usando el CDN https://cdnjs.com/libraries/izimodal

Ahora tienes que añadir el código que necesites, por ejemplo:

```
<button data-izimodal-open="modal-id">Open</button>
<!-- Specifying the opening transition -->
<button data-izimodal-open="modal-id" data-izimodal-transitionin="
    fadeInDown">Open</button>
```

Todos los ejemplos los puedes encontrar en su página web oficial

Página oficial

Moment.js

Esta librería es especialmente útil cuando queremos formatear fechas, es decir, si le pasamos al script una fecha, Moment.js podrá transformarla para mostrala de otro modo. También es capaz de coger una fecha, y devolver el tiempo que ha pasado desde entonces hasta ahora. Por si fuera poco también tiene soporte para muchos lenguajes, entre ellos el español.

Para descargarla podemos hacerlo desde su Página oficial

En cuento al uso, es muy sencillo, simplemente llamamos a la función moment().format() desde el javascript, por ejemplo:

```
moment().format('MMMM Do YYYY, h:mm:ss a'); // April 3rd 2017, 5:09:39
    pm
moment().format('dddd');                    // Monday
moment().format("MMM Do YY");               // Apr 3rd 17
moment().format('YYYY [escaped] YYYY');     // 2017 escaped 2017
moment().format();                          // 2017-04-03T17
    :09:39+02:00
```

O para tiempos relativos a otra fecha:

```
moment(”20111031”, ”YYYYMMDD”).fromNow(); // 5 years ago
moment(”20120620”, ”YYYYMMDD”).fromNow(); // 5 years ago
moment().startOf('day').fromNow();        // 17 hours ago
moment().endOf('day').fromNow();          // in 7 hours
moment().startOf('hour').fromNow();       // 10 minutes ago
```

Página oficial

Anime.js

Con esta herramienta serás capaz de crear tus propias animaciones para ponerlas en múltiples elementos del html. Estas animaciones pueden ser todo lo personalizables que tu quieras, en otras palabras, puedes cambiar la dirección, la velocidad, el tiempo, la elasticidad, etc.

Para usarla puedes descargarla desde su Github

En cuento a su fucionamineto, tienes que crear tu animación en un fichero JS para posteriormente invocarla desde tu código html de la página web, por ejemplo:

```
var cssSelector = anime({
  targets: '#cssSelector .el',
  translateX: 250
});
```

```
<div id=”cssSelector”>
  <div class=”line”>
    <div class=”square el”></div>
  </div>
</div>
```

Página oficial

Date dropper

Si un día necesitas incluir en tu página web un calendario donde el usuario puede introducir la fecha que desee, esta librería te lo pone muy fácil. Podrás crear sencillos y vistosos calendarios, adapatados a todo tipo de pantallas y dispositivos y con una librería que ocupa poquísimo espacio.

Para descargarla puedes hacerlo desde su página oficial

Empezar a usarla es muy sencillo, simplemente añades algo parecido a esto:

```
<input type="text" />
<!-- init dateDropper -->
<script>
    $('input').dateDropper();
</script>
```

Página oficial

Drop.js

Este script ahorra tener que crear dropdowns al hacer clic sobre un elemento, lo hace por tí. Dispone de multiples plantillas como tarjetas, sliders, modals, hovers, etc.

En su Github puedes descargarla.

Ahora para usarla, simplemente importas los dos archivos que vienen, y en el enlace de un link, colocas la etiqueta href con la referencia al drop que quieres crear:

```
<link rel="stylesheet" href="drop-theme-arrows.css" />
<script src="tether.min.js"></script>
<script src="drop.min.js"></script>
```

Página oficial

Premonish

Ofrece una funcionalidad muy curiosa, la libería es capaz de «adivinar» el elemento del html sobre el que va a interaccionar el usuario antes de que lo haga, en otras palabras, predice que elemento va a ser pulsado.

La puedes descargar desde su Github

Para usarla en una página web simplemente:

```
import Premonish from 'premonish';
const premonish = new Premonish({
```

```
  selectors: ['a', '.list-of' '.selectors', '.to', '#watch'],
  elements: [] // Alternatively, provide a list of DOM elements to
     watch
});

premonish.onIntent(({el, confidence}) => {
  console.log(el); // The DOM node we suspect the user is about to
     interact with.
  console.log(confidence); // How confident are we about the user's
     intention? Scale 0-1
});
```

Aunque esto es una base, lo puedes personalizar todo lo que quieras, te animo a probarla.

Página oficial

Chart.js

Para representar datos, una buena forma es mediante gráficos y tablas. Con esta herramienta resulta muy sencillo la creación de gráficas con JavaScript y con soporte para todo tipo de pantallas y dispositivos. La librería viene con un monton de tipos de gráficas y configuraciones, como por ejemplo, gráficas lineales, de barras, con círculos, con burbujas, etc.

Descarga: http://www.chartjs.org/

Si queremos empezar a disfrutarla, tenemos que crear un elemento canvas en el HTML, y a continuación, en el javascript, cargamos los datos en el canvas:

```
<canvas id="myChart" width="400" height="400"></canvas>
<script>
var ctx = document.getElementById("myChart");
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: ["Red", "Blue", "Yellow", "Green", "Purple", "Orange"],
        datasets: [{
            label: '# of Votes',
            data: [12, 19, 3, 5, 2, 3],
            backgroundColor: [
                'rgba(255, 99, 132, 0.2)',
                'rgba(54, 162, 235, 0.2)',
```

```
                'rgba(255, 206, 86, 0.2)',
                'rgba(75, 192, 192, 0.2)',
                'rgba(153, 102, 255, 0.2)',
                'rgba(255, 159, 64, 0.2)'
            ],
            borderColor: [
                'rgba(255,99,132,1)',
                'rgba(54, 162, 235, 1)',
                'rgba(255, 206, 86, 1)',
                'rgba(75, 192, 192, 1)',
                'rgba(153, 102, 255, 1)',
                'rgba(255, 159, 64, 1)'
            ],
            borderWidth: 1
        }]
    },
    options: {
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero:true
                }
            }]
        }
    }
});
</script>
```

Como ves, cargamos los datos y seteamos los colores que queremos para nuestra gráfica, todo de una manera muy simple

Página oficial

Clusterize.js

Si añadimos muchisimas filas a una tabla, en el HTML, al hacer scroll sobre la tabla, notaremos que la tabla se siente con mucho lag. Para arreglar esto puedes usar este script, se encarga de sustituir todas las filas en el HTML por una sola, reduciendo mucho la lentitud al hacer scroll.

Te la puedes bajar de aquí https://github.com/NeXTs/Clusterize.js

Para usarlo añadimos al HTML algo parecido a esto, e inicializamos mediante JS la tabla:

```
<!--HTML-->
<div class="clusterize">
  <table>
    <thead>
      <tr>
        <th>Headers</th>
      </tr>
    </thead>
  </table>
  <div id="scrollArea" class="clusterize-scroll">
    <table>
      <tbody id="contentArea" class="clusterize-content">
        <tr class="clusterize-no-data">
          <td>Loading …data</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

```
// JavaScript
var data = ['<tr…></tr>', '<tr…></tr>', …];
var clusterize = new Clusterize({
  rows: data,
  scrollId: 'scrollArea',
  contentId: 'contentArea'
});
```

Página oficial

Notie.js

Quizás esta herramienta no te parezca muy útil, ya que solo añade un script para generar notificaciones en pantalla, pero si quieres hacer un proyecto rápido o no tienes mucho tiempo para diseñar las notificaciones, esta librería te puede venir muy bien.

Si la quieres probar la puedes encontrar aquí https://jaredreich.com/projects/notie

Para hacer saltar la notificación, simplemente llamamos al metodo notie, por ejemplo:

```
notie.alert({
  type: Number|String, // optional, default = 4, enum: [1, 2, 3, 4, 5,
     'success', 'warning', 'error', 'info', 'neutral']
  text: String,
  stay: Boolean, // optional, default = false
  time: Number, // optional, default = 3, minimum = 1,
  position: String // optional, default = 'top', enum: ['top', 'bottom
     ']
})
```

Página oficial

Layzr.js

Layzr ofrece una forma de incluir imágenes en una página web con carga perezosa, es decir, éstas imágenes no se renderizaran en pantalla hasta que no aparezcan. Por ejemplo, si tenemos una web con muchas imágenes, y que además, muchas de ellas no se ven hasta que el usuario hace scroll, este sistema hace que la página cargue mucho mas rápido, ofreciendo una mejor experiencia al usuario.

Puedes descargartela de aqui https://github.com/callmecavs/layzr.js/releases

Para inicializar las imágenes tienes que hacerlo de este modo.

```
<img data-normal="normal.jpg">
```

Página oficial

Iconate

Su premisa es muy simple, mediante los conocidos iconos de FontAwesome, esta herramienta es capaz de transformar los iconos en otros al hacerlos pulsar, por ejemplo, al pulsar sobre el icono de desplegar el menú, el icono se transforma, mediante una animación, en el icono de la equis para cerrar el menú. Éste es solo un ejemplo pero hay muchos ejemplos más, te animo a descubrirlos todos.

Para desacargarlo puedes hacerlo desde aquí https://github.com/bitshadow/iconate

Ahora sencillamente ejecutamos el script en el icono para que se transforme:

```
var iconElement = document.getElementById('icon');
var options = {
    from: 'fa-bars',
    to: 'fa-arrow-right',
    animation: 'rubberBand'
};

iconate(iconElement, options);
```

Página oficial

Clipboard.js

Este es el típico icono que aparece para que al pulsar se copie el contenido del input en el portapapeles, esto es especialmente útil cuando queremos ahorrar al usuario tener que seleccionar el texto para copiarlo. Está soportado por todos los navegadores modernos.

Puedes bajartelo de aquí https://github.com/zenorocha/clipboard.js

Un ejemplo de uso:

```
<!-- Target -->
<input id="foo" value="https://github.com/zenorocha/clipboard.js.git">

<!-- Trigger -->
<button class="btn" data-clipboard-target="#foo">
    <img src="assets/clippy.svg" alt="Copy to clipboard">
</button>
```

Página oficial

Bounce.js

Para finalizar, una librería muy liviana para incluir sencillas animaciones de rebote, movimiento, aparición, etc.

Link de descarga: https://github.com/tictail/bounce.js

Por ejemplo para crear una animación

```
var bounce = new Bounce();
bounce.scale({
  from: { x: 0.5, y: 0.5 },
  to: { x: 1, y: 1 }
});
```

Como ves, permite configurar los parámetros a nuestro gusto por lo que es perfecta para cualquier tipo de página web.

Página oficial

Y hasta aquí todas las librerías, espero que os hayan gustado. Por último añadir que esta lista no sigue ningún orden en concreto, son librerías que personalmente me han parecido muy útiles o interesantes para desarrolladores web.

¡Saludos! En el post de hoy veremos algunos de los comandos más utilizados, pero tranquilo, si ya eres un usuario más avanzado de git, no te preocupes, también veremos algunos comandos más menos conocidos y avanzados pero igualmente útiles.

Pero antes de todo esto, no está de más recordar qúe es git y para qué sirve por si hay alguien que se quiere iniciar en git.

Git es una herramienta que sirve para gestionar el control de versiones. Pero, ¿qué es el control de versiones? Pues es simplemente una forma de tener controlados todos los cambios que realizamos sobre cualquier tipo de archivos. Este control se puede efectuar a mano, aunque es recomendable usar una herramienta que nos facilite la vida como por ejemplo git, aunque hay muchas más.

Instalación y configuración de Git.

Lo primero es tener instalado git en nuestra máquina para tener acceso a todos sus comandos. Para descargar e instalar git lo puedes hacer desde su página oficial:

Si estás en Windows, puedes descargarlo directamente desde aquí:

https://git-scm.com/download/win

Si usas otro sistema puedas buscar el tuyo aquí:

https://git-scm.com/downloads

Una vez instalado git, puedes configurar tu identidad ejecutando en la terminal (si estás en windows se te habrá instalado un programa llamado Git bash):

```
git config --global user.name ”Tu nombre aquí”
git config --global user.email ejemplo@example.com
```

Comandos básicos

Bien, pongámonos en la situación de que tenemos un proyecto que estamos dearrollando y queremos cambiar algo drástico. Lo mejor es usar git para llevar el control de estos cambios por si queremos revertirlos.

El primer paso es abrir la terminal de git, o la terminal de nuestro sistema, para dirigirnos a la carpeta donde tengamos guardado el proyecto que estemos desarrollando, y ejecutar el siguiente comando:

```
git init
```

Con este comando lo que estamos haciendo es decirle a git que este pendiente de los cambios que se produzcan en los archivos de ese directorio. Este comando solo lo tenemos que ejecutar una sola vez para cada proyecto que estemos realizando.

Ahora podemos continuar desarrollando nuestro proyecto.

Cuando queramos guardar los cambios con git tenemos que hacer lo siguiente:

```
git status
```

Este comando imprimirá los archivos que van a ser guardados, el siguiente paso es ejecutar:

```
git add .
```

Con esto añadiremos todos los archivos que aparecían anteriormente para ser guardados. Si queremos añadir un archivo o carpeta en concreto lo podemos hacer mediante

```
git add NOMBREDELARCHIVO
```

Si queremos eliminar los archivos que acabamos de añadir para ser guardados lo podemos hacer con

```
git rm .
```

Con estos comandos hemos añadido o hemos quitado archivos pero aún no han sido guardados, para ello:

```
git commit -m ”Nombre descriptivo del cambio que hemos realizado”
```

Acabamos de hacer nuestro primer commit, un commit es un guardado con mensaje de los cambios que hemos realizado en un momento determinado en nuestro proyecto

Para imprimir todos los commits que hemos realizado tenemos el comando:

```
1 git log
```

La cadena de números y letras que aparece al lado de la palabra «commit» es el identificador que podemos usar para revertir los cambios y volver atrás a ese punto. Para ello ejecutamos:

```
1 git reset --hard IDENTIFICADOR
```

Sustituimos IDENTIFICADOR por el identificador que comentaba anteriormente, eliminando así, todos los commits posteriores a ese commit.

Con *git revert* se crea un nuevo commit que revierte los cambios realizados en el último commit, pero no elimina dicho commit.

Administrando proyectos git

Ahora, imaginemos que queremos administrar un proyecto ubicado en otro servidor o en un almacén de repositorios como es Github

Si queremos bajarnos el proyecto para empezar a gestionarlo lo podemos hacer usando:

```
1 git clone git://servidor/ruta/a/los/archivos
```

Por ejemplo, para GitHub:

```
1 git clone https://github.com/proyectogithub
```

Si alguien hace un cambio desde otro sitio a este mismo repositorio, tenemos que bajarnos el cambio a nuestra máquina, para ello:

```
1 git pull
```

Pero, si tenemos cambios sin guardar, tenemos que guardarlos haciendo commit, antes de bajarnos nada.

Tras hacer commit de cambios nuevos, podemos subirlos al repositorio remoto:

```
1 git push origin master
```

Donde origin es la dirección del repositiorio remoto y master es la rama a la que estamos subiendo los cambios.

Para añadir la dirección de un repositorio remoto a origin, lo podemos hacer así:

```
1 git remote add origin https://github.com/user/repo.git
```

De esta forma, podemos usar la palabra origin para referirnos a esa dirección.

Ramas en git

Muy bien, pero antes has dicho que has subido los cambios a una rama, ¿qué es una rama?

Las ramas sirven para llevar un control de cambios independiente en el mismo repositorio, es decir, podemos crear una rama a partir de la rama base, o rama master, con otra serie de cambios, posteriormente podemos subir estos cambios a la rama master haciendo una combinación de ambas ramas:

```
1 git push origin master
```

Si quires ver los cambios que has realizado desde el último commit lo puedes hacer con:

```
1 git diff
```

En verde, saldrán las líneas que has añadido, y en rojo las que has eliminado desde la última vez.

Para posicionarnos en una rama escribe:

```
1 git checkout nombre_de_la_rama
```

Y para crear una rama:

```
1 git checkout -b rama_nueva
```

Comandos avanzados

Alias

Para no tener que estar escribiendo todo el rato «commit» o «checkout» podemos crear alias, los alias sirven para decirle a git que comando tiene que ejecutar para el alias que le hemos indicado, estos son los alias más comunes, aunque puedes crear y configurar más.

```
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.st status
```

De esta forma, al escribir *git st*, git ejecutará el comando *git status* ahorrándonos mucho tiempo.

Descartar temporalmente cambios

Si estas trabajando en una rama y quieres cambiarte a otra, git no te dejará porque tienes cambios sin guardar, una forma de solucionar esto es haciendo un commit, pero si no queremos hacerlo lo que podemos hacer es descartar los cambios temporalmente, para ello:

```
git stash
```

Posteriormente si los quieres recuperar:

```
git stash pop
```

Pull de un solo commit

Si por cualquier motivo, necesitas hacer un pull pero solo de un determinado commit lo que puedes hacer es usar este comando:

```
git cherry-pick <commitSHA>
```

Git log avanzado

Hay veces en las que el comando git log ofrece demasiada información, pero esto se puede personalizar, por ejemplo:

```
git log --online
```

Se imprimirá en cada linea un commit, con su identificador y el texto del commit.

Otro parámetro bastante útil del log es el de git graph

```
git log --graph --oneline
```

Esto imprimira la lista de commits y mediante caracteres ASCII, representara el árbol con las ramas y los cambios entre ellas.

También podemos filtrar los commits, por ejemplo:

```
git log --author="John"
git log --after="2014-7-1"
git log -- foo.py bar.py
```

Estos comandos filtarán los commits por autor, por fecha y por los archivos que fueron modificados respectivamente.

Conclusiones

Git es una herramienta muy potente que ofrece muchísmos comandos para la gestios de versiónes de nuestros proyectos. Los commmits que he explicado son unos cuantos, me dejo muchos de ellos, pero te animo a que eches un vistazo a la documentación oficial de git para que descubras muchos mas comandos y configuraciones.

¿Qué es React?

React es una librería javascript para crear interfaces de usuario dependiendo de su estado. Las aplicaciones React se construyen mediante componentes, los cuales son elementos independientes y pueden ser reutilizados, además, describen cómo tienen que visualizarse y cómo tienen que comportarse.

React utiliza el llamado HTML virtual DOM, el cual se renderiza mucho más rapido que el HTML tradiccional ya que no se calculan sus estilos CSS. Cuando el virtual DOM cambia se genera uno nuevo y se calcula las diferencias con el anterior virtual DOM. Por último React genera los cambios pertinentes en el HTML.

Instalación y configuración de React

La forma más rápida de instalar React es inluyendo el CDN en un archivo html:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
```

```
5       <script src="https://unpkg.com/react@15/dist/react.min.js"></script
            >
6       <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js">
            </script>
7       <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
            standalone/6.24.0/babel.js"></script>
8
9   </head>
10  <body>
11  </body>
12  </html>
```

A continuación añadimos un script babel, y un elemento root que usará React para renderizar todos los elementos:

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <meta charset="UTF-8">
5       <script src="https://unpkg.com/react@15/dist/react.min.js"></script
            >
6       <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js">
            </script>
7       <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
            standalone/6.24.0/babel.js"></script>
8
9   </head>
10  <body>
11
12      <div id="root"></div>
13
14      <script type="text/babel">
15      </script>
16
17  </body>
18  </html>
```

Por último comprobamos que lo hemos instalado correctamente renderizando un elemento dentro de la página mediante React.render

```
1   <!DOCTYPE html>
2   <html>
3   <head>
```

```
    <meta charset="UTF-8">
    <script src="https://unpkg.com/react@15/dist/react.min.js"></script>
    <script src="https://unpkg.com/react-dom@15/dist/react-dom.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.24.0/babel.js"></script>

</head>
<body>

    <div id="root"></div>

    <script type="text/babel">

    ReactDOM.render(
        <div>Hello World</div>,
        document.getElementById("root")
    )

    </script>

</body>
</html>
```

JSX

JSX es una extensión a la sintaxis de Javascript que permite insertar código HTML. Usando JSX podemos crear elementos React fácilmente:

```
    var element = <h1>Hello World!</h1>
```

En cambio si no tenemos JSX, el proceso es mucho mas lento y menos legible:

```
     var element = React.createElement(
        'h1',
        null,
        'Hello World!'
    )
```

Además, JSX permite escribir expresiones mas complejas:

```
1
2     var item = {
3         name: ”Cheese”,
4         price: 5
5     }
6     var element = <p>{item.name} : ${item.price} </p>
```

Componentes en React

Como decíamos anteriormente, un componente en React es un elemento independiente y reeutilizable. Además existen dos tipos de componentes en React:

- Componentes funcionales: Solo tienen propiedades.
- Componentes de clase: Tienen propiedades, ciclos de vida y propiedades.

Componentes funcionales

Son componentes que generan elementos React. Por convenio se pone el nombre de la función en mayúsculas. Para renderizarlo simplemente se pone una etiqueta con el nombre de la función. Por ejemplo:

```
1   function HelloWorld(){
2         return <h1>Hello World!</h1>
3     }
4
5     ReactDOM.render(
6         <HelloWorld/>,
7         document.getElementById(”root”)
8     )
```

A estas funciones le puedes pasar parámetros, el primero de ellos son las propiedades, por ejemplo:

```
1 function HelloWorld(props){
2         return <h1>Message: {props.message}</h1>
3     }
4
5 ReactDOM.render(
6         <HelloWorld message=”Hello World!”/>,
7         document.getElementById(”root”)
8     )
```

En este ejemplo al renderizar el elemento, pasamos una variable al componente, que se encargará de renderizarla. Puedes pasar como parámetros variables, arrays e incluso otros elementos React.

También se puede configurar la salida de un componente dependiendo de un operador condicional:

```
function Feature(props){
    return <h1>This feature is {props.active? "active" : "not active"
        }</h1>
}
```

Si la variable que llega a la función es true imprimirá «This feature is active» y «This feature is not active» en caso contrario. También podemos revolver null en un componente para que no se renderice.

Ejemplo de aplicación React

Para entender mejor la teoría vamos a construir una aplicavión React para la lista de la compra.

Empezamos creando un componente que represente un único elemento de la lista:

```
function ListItem(props){
        return <li>Test String</li>
}
```

Creamos otro componente para alojar el título y la descripción de la página:

```
function ShoppingTitle(props){
        return (
            <div>
                <h1>Test Title</h1>
                <h2>Test Description</h2>
            </div>

        )
    }
```

También creamos un componente que contenga una lista y su título. Dentro de la lista colocamos el componente con un únco item de la lista, el que creamos anteriormente:

```
function ShoppingList(props){
        return (
            <div>
                <h3>Test Header</h3>
                <ol>
```

```
                    <ListItem/>
                    <ListItem/>
                    <ListItem/>
                </ol>
            </div>
        )
    }
```

Ahora vamos a crear otro componente que contenga la una lista junto con el título de la lista:

```
function ShoppingApp(props){

        return (
            <div>
                <ShoppingTitle/>
                <ShoppingList/>
                <ShoppingList/>
                <ShoppingList/>
            </div>
        )
    }
```

Por el momento, estamos devolviendo títulos de prueba para comprobar que lo que estamos haciendo funciona, es hora de cambiar esto por los datos:

```
    function ShoppingApp(props){

        return (
            <div>
                <ShoppingTitle title = "My Shopping List" numItems = "9
                    "/>
                <ShoppingList header = "Food" items = {[ "Apple","Bread
                    ","Cheese"]}/>
                <ShoppingList header = "Clothes" items = {[ "Shirt","
                    Pants","Hat"]}/>
                <ShoppingList header = "Supplies" items = {[ "Pen","
                    Paper","Glue"]}/>
            </div>
        )
    }
```

En cada ShoppingList añadimos el header y la lista con los items, este componente se encargará de pasar los elementos a cada uno de los items para representarlos. Al componente ShoppingTitle le

pasamos el nombre de la aplicación y el número total de items.

Ahora tenemos que modificar cada uno de los componentes para que rendericen los elementos que llegan desde los parámetros:

```
function ShoppingList(props){
        return (
            <div>
                <h3>{props.header}</h3>
                <ol>
                    <ListItem item = {props.items[0]}/>
                    <ListItem item = {props.items[1]}/>
                    <ListItem item = {props.items[2]}/>
                </ol>
            </div>
        )
    }


function ShoppingTitle(props){
        return (
            <div>
                <h1>{props.title}</h1>
                <h2>Total Number of Items: {props.numItems}</h2>
            </div>

        )
    }

function ListItem(props){
        return <li>{props.item}</li>
     }
```

Para probar nuestra aplicación, usamos React.render para mostrar la aplicación:

```
 ReactDOM.render(
        <ShoppingApp/>,
        document.getElementById(”root”)
    )
```

Si todo ha salido bien el resultado tiene que ser como este:

Por el momento eso es todo, en posteriores artículos seguiremos aprendiendo otras características de React para sacarle todo el potencial.

Componentes de clase

Anteriormente, vimos que React tiene dos tipos de componentes: los funcionales y los de clase. En esta ocasión veremos los componentes de clase, que difiere de los funcionales en los estados y ciclos de vida. Los componentes de clase tienen dos propiedades, this.state y this.props Un ejemplo sencillo de componente de clase:

```
class Welcome extends React.Component{
    render(){
        return <h1>Hello World!</h1>
    }
}
```

Dentro de la función render(), se sitúa el elemento que queremos devolver.

Para renderizarlo podemos hacerlos igual que cuando vimos los componentes funcionales, escribiendo su nombre dentro de una etiqueta HTML:

```
<Welcome/>
```

Para acceder a las propiedades de la clase, podemos hacerlo mediante this.props, por ejemplo:

```
class Welcome extends React.Component{
    render(){
        return <h1>Message: {this.props.message}</h1>
    }
}
```

```
<Welcome message="Hello World!"/>
```

Estados

Dentro de los componentes de clase existe una función constructor que se utiliza para definir el estado inicial del componente. Es muy importante llamar a la función super() dentro del constructor para que los parámetros funcionen adecuadamente:

```
class Welcome extends React.Component{
    constructor(props){
        super(props)
    }
    render(){
```

```
        return <div>Hello World!</div>
    }
}
```

Para definir el estado inicial podemos hacerlo creando un objeto y pasándoselo a la variable state:

```
class Counter extends React.Component{
    constructor(props){
        super(props)
        this.state = {foo:123,bar:456}
    }
    render(){
        return <div>foo:{this.state.foo} bar:{this.state.bar}</div>
    }
}
```

El método updateState se utiliza para actualizar el estado del componente. Recoge un objeto de actualización y actualiza el estado del componente fusionando los atributos del objeto del actualizador con el estado del componente anterior. El método actualiza el estado de forma asíncrona, por lo que hay una opción de callback que se llamará una vez que el estado ha terminado de actualizar completamente. Para usar el método updateState(), debe ser referenciado llamando a this.updateState(). Por ejemplo:

```
class Counter extends React.Component{
    constructor(props){
        super(props)
        //initial state set up
        this.state = {message:"initial message"}
    }
    componentDidMount()
        //updating state
        this.setState({message:"new message"})
    }
    render(){
        return <div>Message:{this.state.message}</div>
    }
}
```

Hay que tener especial cuidado con los estados previos, puesto que cuando llamamos a la función para actualizar el estado actual no se cambia instantáneamente,sino que se añade a una cola esperando a procesarse. Si por ejemplo llamamos 4 veces seguidas para actualizar el estado, solo surtirá efecto la primera llamada.

En este ejemplo podemos ver cómo actualizar el estado teniendo en cuenta estados previos:

```
class Counter extends React.Component{
    constructor(props){
        super(props)
        //initial state set up
        this.state = {message:”initial message”}
    }
    componentDidMount()
        //updating state
        this.setState((prevState, props) => {
            return {message: prevState.message + '!'}
        })
    }
    render(){
        return <div>Message:{this.state.message}</div>
    }
}
```

El estado es inmutable, eso quiere decir que no podemos cambiarlo manualmente actualizando sus variables. Por ejemplo la siguiente sentencia es incorrecta:

```
this.state.message = ”new message”
```

Ciclos de vida

Los componentes de clase siguen unos ciclos de vida, pero sobreescribiendo estos métodos podemos incluir código para ser ejecutado en cada uno de los ciclos.

Fase de montaje

- constructor(props) - Cuando el componente se inicializa. Se ejecuta una sola vez.
- componentWillMount() - Cuando el componente está a punto de ser montado.
- render() - Cuando el componente ya se ha renderizado.
- componentDidMount() - Cuando el componente ya se ha montado.

Fase de actualización

- componentWillReceiveProps(nextProps) - Cuando el componente esta recibiendo nuevos parámetros.

- shouldComponentUpdate(nextProps, nextState) - Cuando el componente ha recibido todos los parámetros y está a punto de actualizarse.
- componentWillUpdate(nextProps, nextState) - Cuando el componente va a actualizarse.
- render() - Cuando el componente va a ser renderizado.
- componentDidUpdate(prevProps, prevState) - Cuando el componente ha terminado de actualizarse.
- componentWillUnmount() - Cuando el componente va a ser desmontado.

Manipuladores de eventos

Crear eventos en ReactJS es similar a crearlos en HTML con la diferencia de que el React se usan llaves:

```
<button onClick = {clickHandler} >Click Me</button>
```

Para manipular estos eventos se pueden crear funciones dentro de las clases de React, por ejemplo:

```
class Counter extends React.Component{
    constructor(props){
        super(props)
        this.state = {count:0}
        //binding is necessary to make `this` point to the correct
           object
        this.clickHandler = this.clickHandler.bind(this)
    }
    clickHandler(){

      this.setState((prevState,props) => {
        return {count: prevState.count + 1}
      })
    }
    render(){

        return <button onClick = {this.clickHandler}>{this.state.count
           }</button>
    }
}
```

La función bind(), que está en el constructor se usa para enlazar la palabra clave a la instancia del componente, si no incluimos este método, apuntará a un objeto incorrecto y por tanto la aplicación no funcionará correctamente.

Elevando los estados hacia arriba

El método setState() sólo permite que los componentes actualicen su propio estado. Si queremos actualizar el estado de un componente a un componente hijo o hermano no lo podemos hacer directamente. Tenemos que elevar el estado hacia la clase principal para que enlace el estado una vez actualizado a los componentes que lo necesiten.

Para ver cómo podemos aplicar esta técnica puedes echar un vistazo a este ejemplo:

https://codepen.io/benjlin/pen/OmaLqE

Como ves, el ejemplo consta de 4 botones con una descripción en la parte inferior que muestra el botón que está activo.

Cuando pulsamos sobre un botón, este componente tendrá que actualizar los estados de los demás botones para que se vuelvan inactivos. Es por ello que tenemos que elevar el estado, es decir al controlador de eventos para que se encargue de coger el array de botones activos para volverlos inactivos.

Introducción

Java no sea uno de los mejores lenguajes para la creación de videojuegos, mucho menos para videojuegos en 3D, pero para muchos (me incluyo), es uno de los lenguajes con los que empezamos a programar. Aunque este artículo este escrito usando Java, se pueden usar las ideas y conceptos que expongo en otros lenguajes consiguiendo resultados parecidos. Este artículo esta basado en los juegos de Notch (creador de minecraft).

Este es el resultado final de la base de motor gráfico que vamos a realizar hoy:

Primeros pasos

Lo primero que hacemos es crear un proyecto Java vacío con nuestro IDE favorito, en mi caso voy a usar eclipse (https://www.eclipse.org/).

Creamos la clase principal del proyecto con el método **main**, es decir el primer método que va a ejecutarse, y hacemos que esta clase principal herede de la clase Canvas y implemente la case Runnable. A continuación definimos las variables que van a contener las medidas de la ventana del juego:

```
1 private final static long serialVersionUID = 1L;
2 private final static int scale = 2;
3 private final static int WIDTH = 320;
4 private final static int HEIGHT = WIDTH / 16 * 9;
5
```

```
6  private JFrame frame;
7  private boolean running = false;
8
9  public Game() {
10     Dimension dimension = new Dimension(WIDTH*scale, HEIGHT*scale);
11     setPreferredSize(dimension);
12     frame = new JFrame();
13 }
```

Como ves, he creado un jFrame que será la ventana del juego, y una variable para saber si el juego se está ejecutando. También he creado el constructor de nuestro juego, inicializando las dimensiones que creamos anteriormente y inicializando el JFrame.

Ahora, en el método **main** configuramos el JFrame del juego:

```
1  public static void main(String[] args) {
2      Game game = new Game();
3      game.frame.setTitle("Motor gráfico 3D");
4      game.frame.add(game);
5      game.frame.pack();
6      game.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7      game.frame.setLocationRelativeTo(null);
8      game.frame.setVisible(true);
9  }
```

Si ahora ejecutamos el código veremos una ventana vacía.

Game loop

El game loop es lo que permite que un juego se actualice, es decir cada muy poco tiempo (ticks) actualiza todos los elementos del juego. También pinta todos los elementos en pantalla cada poco tiempo (fps). Es muy importante que los ticks se actualicen siempre de manera constante. Esto es por si por ejemplo, el juego se ejecuta en una máquina con menos recursos, el juego ira más lento de fps y si las animaciones dependen de este número se ejecutaran a cámara lenta.

```
1  public synchronized void start() {
2      if (running) return;
3          running = true;
4          Thread thread = new Thread(this);
5          thread.start();
6      }
7  }
```

```
8
9  public synchronized void stop() {
10     if (!running) return;
11         running = false;
12     try {
13         thread.join();
14     } catch (InterruptedException e) {
15         e.printStackTrace();
16     }
17 }
18
19 @Override
20 public void run() {
21     while(running) {
22         tick();
23         render();
24     }
25 }
26
27 private void render() {
28     // TODO Auto-generated method stub
29 }
30
31
32 private void tick() {
33     // TODO Auto-generated method stub
34 }
```

Como ves, el método start() simplemente comprueba si el juego se esta ejecutando, y si no lo está crea un thread y lo inicializa, mientras que el método stop() para su ejecución. El método run() es el que se ejecutará mientras el thread este ejecutándose. En este método simplemente llamamos a otros dos métodos que aún no hemos creado, uno para actualizar los elementos del juego y otro para pintar el juego en pantalla;

Pintar en pantalla píxel a píxel puede causar problemas en el renderizado, es por ello que tenemos que crear un frame (que no está a la vista) donde pintaremos todos estos píxeles, cuando todo el frame esta calculado, sustituimos el frame que esté en ese momento en pantalla por el nuevo frame, es decir en pantalla, se pinta frame a frame y por debajo se van pintando los píxeles.

Pero hay otro problema, tener que esperar a que el frame este pintado aprovecha poco los recursos, por eso podemos crear otro frame para ser calculado en paralelo. Es por eso que tenemos que crear 3 frames.

```
private void render() {
    BufferStrategy bs = getBufferStrategy();

    if (bs == null) {
        createBufferStrategy(3);
        return;
    }
}
```

Bitmap y Screen

Ahora vamos a crear estas dos clases, empecemos por el Bitmap. Un Bitmap es un conjunto de bits que corresponden a una image. Para nuestra clase vamos a crear tres variables, width y height para establecer sus dimensiones y un array de pixels.

```
public class Bitmap {

    public final int width;
    public final int height;
    public final int[] pixels;

    public Bitmap(int width, int height) {
        this.width = width;
        this.height = height;
        pixels = new int[width * height];

    }

    public void draw(Bitmap bitmap, int xOffs, int yOffs) {
        for (int y = 0; y < bitmap.height; y++) {
            int yPix = y + yOffs;
            if (yPix < 0 || yPix >= height)
                continue;

            for (int x = 0; x < bitmap.width; x++) {
                int xPix = x + xOffs;
                if (xPix < 0 || xPix >= width)
                    continue;

                pixels[xPix + yPix * width] = bitmap.pixels[x + y *
                    bitmap.width];
```

```
26                 }
27             }
28         }
29 }
```

La función draw se encarga de meter en el array de píxeles otro array que se le pasa, esto se usará en la clase Screen que usará este método para inicializar sus píxeles con los del bitmap. Las variables xOffs y yOffs son para desplazar los píxeles.

La clase screen:

```
1  public class Screen extends Bitmap {
2
3      private Bitmap testBitmap;
4
5      public Screen(int width, int height) {
6
7          super(width, height);
8          Random random = new Random();
9          testBitmap = new Bitmap(128, 128);
10
11         for (int i = 0; i < testBitmap.width * testBitmap.height; i++)
               {
12             testBitmap.pixels[i] = (int)System.nanoTime();
13         }
14     }
15
16     public void render() {
17
18         draw(testBitmap, 0, 0);
19     }
20
21     public void clear() {
22         for (int i = 0; i < pixels.length; i++) {
23             pixels[i] = 0;
24         }
25     }
26
27     public int[] getPixels() {
28         return pixels;
29     }
30 }
```

La clase Screen hereda de Bitmap. En el método render() simplemente llamamos al método draw() heredado con el bitmap de prueba que hemos creado. Con esto inicializaremos los píxeles con los del Bitmap de prueba. El método clear() sirve para borrar la pantalla simplemente poniendo los píxeles a cero, esto se tiene que llamar antes de volver a pintar los pixeles para que los anteriores se borren y no hayan problemas gráficos.

Para probar todo esto, en la clase Game principal, en el método render, añadimos:

```
1  private void render() {
2      BufferStrategy bs = getBufferStrategy();
3
4      if (bs == null) {
5
6          createBufferStrategy(3);
7
8          return;
9
10     }
11
12     screen.clear();
13     screen.render();
14
15     for (int i = 0; i < WIDTH * HEIGHT; i++) {
16         pixels[i] = screen.pixels[i];
17     }
18
19     Graphics g = bs.getDrawGraphics();
20     g.drawImage(image, 0, 0, WIDTH * scale, HEIGHT * scale, null);
21
22     g.dispose();
23     bs.show();
24
25 }
```

Si ejecutamos el código veremos en pantalla un sprite con colores aleatorios.

Espacio 3D

Hasta aquí hemos trabajado en 2 dimensiones. La tercera dimensión la tenemos que sacar a partir de las dos anteriores, para ello, empezaremos creando la clase Bitmap3D, que heredará de Bitmap:

```
1  public class Bitmap3D extends Bitmap {
```

```
2
3       public Bitmap3D(int width, int height) {
4           super(width, height);
5           // TODO Auto-generated constructor stub
6       }
7
8       public void render() {
9           for (int y = 0; y < height; y++) {
10              double yd = (y + 0.5 - height / 2.0) / height;
11
12              if (yd == 0)
13                  continue;
14
15              double z = 6 / yd;
16
17              for (int x = 0; x < width; x++) {
18                  double xd = (x - width / 2.0) / height;
19                  xd *= z;
20                  int xx = (int) xd & 15;
21                  int zz = (int) z & 15;
22                  pixels[x + y * width] = (xx * 16) | (zz * 16) << 8;
23              }
24          }
25      }
26
27  }
```

El método RenderFloor me sirve para hacer pruebas de renderizado de un pseudoespacio 3D. Para ello empezamos poniendo dos bucles for para recorrer las dos dimensiones. Para la tercera dimensión empiezo creado variables auxiliares.

La primera variable, **yd**, se encarga de coger la variable de la altura para que conforme aumente, **yd** vaya disminuyendo, pero cuando llega al centro de la pantalla, esta variable vuelve a aumentar. Esto sirve para dibujar los pixeles objetos que se encuentren cerca más grandes y mas pequeños si estan lejos.

Creamos la variable z dividiendo 6 entre **yd** para que los valores sean más grandes.

Dentro del bucle de la coordenada x, creamos la variable **xd**, y hacemos lo mismo, ya que los objetos en pantalla se tienen que ver de distinto tamaño si están en el centro o si están en los laterales.

Por último metemos en el array de píxeles las variables anteriores multiplicando y desplazando los bits para que pinte «cuadrados».

Con la unión de estas varibles conseguimos el efecto de entorno tridimensional ya que dependiendo de estas 3 coordenadas los objetos se representaran con diferente tamaño.

Para pintar esto en pantalla, creamos un bitmap3D en la clase screen y llamamos a su método render(), luego al método draw() para que se dibuje en pantalla.

Podemos crear la clase Art, esta clase se encargará por el momento de cargar una imagen formada por sprites de 8x8, para ello:

```
1  import java.awt.image.BufferedImage;
2
3  import javax.imageio.ImageIO;
4
5  import gui.Bitmap;
6
7
8  public class Art {
9      public static Bitmap floors = loadBitmap("/assets/floors.png");
10
11     public static Bitmap loadBitmap(String filename) {
12         try {
13             BufferedImage img = ImageIO.read(Art.class.getResource(
                   filename));
14
15             int w = img.getWidth();
16             int h = img.getHeight();
17
18             Bitmap result = new Bitmap(w, h);
19             img.getRGB(0, 0, w, h, result.pixels, 0, w);
20             return result;
21         } catch (Exception e) {
22             throw new RuntimeException(e);
23         }
24     }
25 }
```

No tiene mucho misterio, simplemente carga la ruta de la imagen y coge sus píxeles. Ahora podemos cambiar los colores que salían por la textura, para ello cambiamos el método render() de la clase Bitmap3D como se muestra:

```
1  public void render() {
2
3          for (int y = 0; y < height; y++) {
4              double yd = (y + 0.5 - height / 2.0) / height;
```

```
5
6               double z = 8 / yd;
7
8               if (yd < 0)
9                   z = 4 / -yd;
10
11              for (int x = 0; x < width; x++) {
12                  double xd = (x - width / 2.0) / height;
13                  xd *= z;
14                  int xx = (int) xd & 7;
15                  int yy = (int) z & 7;
16                  pixels[x + y * width] = Art.floors.pixels[xx + yy *
                        64];
17              }
18          }
19  }
```

Recuerda que la imagen tiene que estar formada de tiles de 8x8.

```
1   public void render(Game game) {
2       int floorHeight = 6;
3       int ceilingHeight = 10;
4
5       xCam = 0;
6       yCam = 0;
7       zCam = 0;
8
9       double rot = Math.sin(0 / 40.0) * 0.5;
10
11      rCos = Math.cos(rot);
12      rSin = Math.sin(rot);
13
14      fov = height;
15
16      for (int y = 0; y < height; y++) {
17          double yd = ((y + 0.5) - height / 2.0) / fov;
18
19          double zd = (floorHeight + zCam) / yd;
20          if (yd < 0) {
21              zd = (ceilingHeight - zCam) / -yd;
22          }
23
24          for (int x = 0; x < width; x++) {
```

```
25              double xd = (x - width / 2.0) / fov;
26              xd *= zd;
27
28              double xx = xd * rCos + zd * rSin + (xCam + 0.5) * 8;
29              double yy = zd * rCos - xd * rSin + (yCam + 0.5) * 8;
30
31              int xPix = (int) (xx);
32              int yPix = (int) (yy);
33              if (xx < 0)
34                  xPix--;
35              if (yy < 0)
36                  yPix--;
37
38              pixels[x + y * width] = Art.floors.pixels[(xPix & 7) + (
                    yPix & 7) * 64];
39          }
40      }
41  }
```

Para empezar, he añadido dos nuevas variables que definen la distancia del centro del suelo y del techo. Luego he añadido otras tres variables para definir la posicion en el espacio de la cámara (xCam, yCam, zCam), y otra variable más para su rotación en el eje de las x (rot). Para calcular esta rotación hay que hacer uso de los senos y cosenos. A continuación se hace uso de estos valores para transformar el espacio en función de la posición y rotación de la cámara.

Con esto ya tenemos un «espacio» en 3D, lo siguiente que vamos a hacer es dibujar paredes.

Paredes y cubos en 3D

Un cubo esta formado por caras o «paredes» por lo tanto para pintar cubos en el espaio 3D antes tenemos que pintar las caras. Para ello vamos a crear una función en la clase Bitmap3D.

```
1  private void renderWall(double x0, double y0, double x1, double y1) {
2      double xo0 = ((x0 - 0.5) - xCam) * 2;
3      double yo0 = ((y0 - 0.5) - yCam) * 2;
4
5      double xx0 = xo0 * rCos - yo0 * rSin;
6      double u0 = ((-0.5) - zCam) * 2;
7      double l0 = ((+0.5) - zCam) * 2;
8      double zz0 = yo0 * rCos + xo0 * rSin;
9
10     double xo1 = ((x1 - 0.5) - xCam) * 2;
```

```
11      double yo1 = ((y1 - 0.5) - yCam) * 2;
12
13      double xx1 = xo1 * rCos - yo1 * rSin;
14      double u1 = ((-0.5) - zCam) * 2;
15      double l1 = ((+0.5) - zCam) * 2;
16      double zz1 = yo1 * rCos + xo1 * rSin;
17
18      double xPixel0 = (xx0 / zz0 * fov + width / 2);
19      double xPixel1 = (xx1 / zz1 * fov + width / 2);
20
21      if (xPixel0 >= xPixel1)
22          return;
23      int xp0 = (int) Math.floor(xPixel0);
24      int xp1 = (int) Math.floor(xPixel1);
25      if (xp0 < 0)
26          xp0 = 0;
27      if (xp1 > width)
28          xp1 = width;
29
30      for (int x = xp0; x < xp1; x++) {
31          double pr = (x - xPixel0) / (xPixel1 - xPixel0);
32
33          double u = (u0) + (u1 - u0) * pr;
34          double l = (l0) + (l1 - l0) * pr;
35          double zz = (zz0) + (zz1 - zz0) * pr;
36
37          double yPixel0 = (int) (u / zz * fov + height / 2);
38          double yPixel1 = (int) (l / zz * fov + height / 2);
39
40          if (yPixel0 >= yPixel1)
41              return;
42          int yp0 = (int) Math.floor(yPixel0);
43          int yp1 = (int) Math.floor(yPixel1);
44          if (yp0 < 0)
45              yp0 = 0;
46          if (yp1 > height)
47              yp1 = height;
48          for (int y = yp0; y < yp1; y++) {
49              double pry = (y - yPixel0) / (yPixel1 - yPixel0);
50              pixels[x + y * width] = 0xff00ff;
51              zBuffer[x + y * width] = 0;
52          }
53      }
```

```
54 }
```

Como puedes ver, lo primero que hacemos es crear la función pasandole 4 parámetros, los correspondientes a las 4 coordenadas de las esquinas de la pared. A continuación, creo nuevas variables auxiliares a las que voy a inicialiciar con los cálculos de las coordenadas con las coordenadas de la cámara de la escena.

```
1  public void postProcess() {
2     for (int i = 0; i < width * height; i++) {
3         int col = pixels[i];
4         int brightness = (int) (15000 / (zBuffer[i] * zBuffer[i]));
5         if (brightness < 0)
6             brightness = 0;
7         if (brightness > 255)
8             brightness = 255;
9
10        int r = (col >> 16) & 0xff;
11        int g = (col >> 8) & 0xff;
12        int b = (col) & 0xff;
13
14        r = r * brightness / 255;
15        g = g * brightness / 255;
16        b = b * brightness / 255;
17
18        pixels[i] = r << 16 | g << 8 | b;
19    }
20 }
```

Este método sirve para coger los píxeles centrales de la escena para bajarles el brillo, de esta forma conseguimos que de la apariencia de que a lo lejos se vea menos.

Para probar la creación de paredes voy a llamar a su método desde la función render de la clase Bitmap, en este caso voy a crear 5 paredes. La clase bitmap3D, por lo tanto, quedaría así:

```
1  import java.awt.MouseInfo;
2  import java.awt.Point;
3  import java.awt.PointerInfo;
4  import java.util.Random;
5
6  import Game.Art;
7  import Game.Game;
8
9  public class Bitmap3D extends Bitmap {
```

```
10      private double[] zBuffer;
11      private double xCam, yCam, zCam, rCos, rSin, fov, xCenter, yCenter,
            rot, rotY;
12      private double[] zBufferWall;
13
14      public Bitmap3D(int width, int height) {
15          super(width, height);
16          zBufferWall = new double[width];
17          zBuffer = new double[width * height];
18      }
19
20      public void render(Game game) {
21
22          for (int x = 0; x < width; x++) {
23              zBufferWall[x] = 0;
24          }
25          for (int i = 0; i < width * height; i++) {
26              zBuffer[i] = 10000;
27          }
28
29          int floorHeight = 4;
30          int ceilingHeight = 4;
31
32          xCenter = width / 2.0;
33          yCenter = height / 2.0;
34
35          xCam = 0;
36          yCam = -0.6;
37          zCam = -0;
38
39          PointerInfo a = MouseInfo.getPointerInfo();
40          Point b = a.getLocation();
41          int mouseX = (int) b.getX();
42          int mouseY = (int) b.getY();
43
44
45          double rot = Math.sin(0 / 40.0) * 0.75;
46
47          double rotY = 2.0;
48
49          rCos = Math.cos(rot);
50          rSin = Math.sin(rot);
51
```

```
52          fov = height;
53
54          for (int y = 0; y < height; y++) {
55              double yd = ((y + 0.5) - height / rotY) / fov;
56
57              double zd = (floorHeight + zCam) / yd;
58              if (yd < 0) {
59                  zd = (ceilingHeight - zCam) / -yd;
60              }
61
62              for (int x = 0; x < width; x++) {
63                  double xd = (x - xCenter) / fov;
64                  xd *= zd;
65
66                  double xx = xd * rCos + zd * rSin + (xCam + 0.5) * 8;
67                  double yy = zd * rCos - xd * rSin + (yCam + 0.5) * 8 +
                        5;
68
69                  int xPix = (int) (xx);
70                  int yPix = (int) (yy);
71                  if (xx < 0)
72                      xPix--;
73                  if (yy < 0)
74                      yPix--;
75
76                  zBuffer[x + y * width] = zd;
77                  pixels[x + y * width] = Art.floors.pixels[(xPix & 7) + (
                        yPix & 7) * 64];
78              }
79          }
80
81          renderWall(0, 1, 0, 2);
82          renderWall(0, 0, 0, 1);
83          renderWall(0, 2, 1, 2);
84          renderWall(1, 2, 1, 1);
85          renderWall(1, 1, 1, 0);
86
87      }
88
89      private void renderWall(double x0, double y0, double x1, double y1)
            {
90
91          double xc0 = ((x0 - 0.5) - xCam) * 2;
```

```
92          double yc0 = ((y0 - 0.5) - yCam) * 1.5;
93
94          double xx0 = xc0 * rCos - yc0 * rSin;
95          double u0 = ((-0.5) - zCam) * 2;
96          double l0 = ((+0.5) - zCam) * 2;
97          double zz0 = yc0 * rCos + xc0 * rSin;
98
99          double xc1 = ((x1 - 0.5) - xCam) * 2;
100         double yc1 = ((y1 - 0.5) - yCam) * 1.5;
101
102         double xx1 = xc1 * rCos - yc1 * rSin;
103         double u1 = ((-0.5) - zCam) * 2;
104         double l1 = ((+0.5) - zCam) * 2;
105         double zz1 = yc1 * rCos + xc1 * rSin;
106
107         double xt0 = 0;
108         double xt1 = 1;
109         double zClip = 0.2;
110
111         xt0 *= 8;
112         xt1 *= 8;
113
114         if (zz0 < zClip && zz1 < zClip)
115             return;
116
117         if (zz0 < zClip) {
118             double p = (zClip - zz0) / (zz1 - zz0);
119             zz0 = zz0 + (zz1 - zz0) * p;
120             xx0 = xx0 + (xx1 - xx0) * p;
121             xt0 = xt0 + (xt1 - xt0) * p;
122         }
123
124         if (zz1 < zClip) {
125             double p = (zClip - zz0) / (zz1 - zz0);
126             zz1 = zz0 + (zz1 - zz0) * p;
127             xx1 = xx0 + (xx1 - xx0) * p;
128             xt1 = xt0 + (xt1 - xt0) * p;
129         }
130         double iz0 = 1 / zz0;
131         double iz1 = 1 / zz1;
132
133         double ixt0 = xt0 * iz0;
134         double ixta = xt1 * iz1 - ixt0;
```

```
135
136         double iza = iz1 - iz0;
137
138         double xPixel0 = (xx0 / zz0 * fov + xCenter);
139         double xPixel1 = (xx1 / zz1 * fov + xCenter);
140
141         if (xPixel0 >= xPixel1)
142             return;
143         int xp0 = (int) Math.floor(xPixel0);
144         int xp1 = (int) Math.floor(xPixel1);
145         if (xp0 < 0)
146             xp0 = 0;
147         if (xp1 >= width)
148             xp1 = width - 1;
149
150         double yPixel00 = (u0 / zz0 * fov + yCenter) + 0.5;
151         double yPixel01 = (l0 / zz0 * fov + yCenter) + 0.5;
152         double yPixel10 = (u1 / zz1 * fov + yCenter) + 0.5;
153         double yPixel11 = (l1 / zz1 * fov + yCenter) + 0.5;
154
155
156         double iw = 1 / (xPixel1 - xPixel0);
157
158         for (int x = xp0; x < xp1; x++) {
159             double pr = (x - xPixel0) * iw;
160
161             double iz = iz0 + iza * pr;
162
163             if (zBufferWall[x] > iz)
164                 continue;
165             zBufferWall[x] = iz;
166
167             int xTex = (int) ((ixt0 + ixta * pr ) / iz);
168
169             double yPixel0 = yPixel00 + (yPixel10 - yPixel00) * pr;
170             double yPixel1 = yPixel01 + (yPixel11 - yPixel01) * pr;
171
172             if (yPixel0 >= yPixel1)
173                 return;
174             int yp0 = (int) Math.floor(yPixel0);
175             int yp1 = (int) Math.floor(yPixel1);
176             if (yp0 < 0)
177                 yp0 = 0;
```

```
178             if (yp1 > height)
179                 yp1 = height;
180             double ih = 1 / (yPixel1 - yPixel0);
181             for (int y = yp0; y < yp1; y++) {
182                 double pry = (y - yPixel0) * ih;
183                 int yTex = (int) (8 * pry);
184                 pixels[x + y * width] = Art.floors.pixels[(xTex & 7) + 8
                        + (yTex & 7) * 64];
185                 //pixels[x + y * width] = 0xff00ff + xTex * 100;
186                 zBuffer[x + y * width] = 1 / iz * 8;
187             }
188         }
189 
190     }
191 
192     private void renderWall(double x0, double y0, double x1, double y1,
            double xt0, double xt1) {
193         double xc0 = ((x0 - 0.5) - xCam) * 2;
194         double yc0 = ((y0 - 0.5) - yCam) * 2;
195 
196         double xx0 = xc0 * rCos - yc0 * rSin;
197         double u0 = ((-0.5) - zCam) * 2;
198         double l0 = ((+0.5) - zCam) * 2;
199         double zz0 = yc0 * rCos + xc0 * rSin;
200 
201         double xc1 = ((x1 - 0.5) - xCam) * 2;
202         double yc1 = ((y1 - 0.5) - yCam) * 2;
203 
204         double xx1 = xc1 * rCos - yc1 * rSin;
205         double u1 = ((-0.5) - zCam) * 2;
206         double l1 = ((+0.5) - zCam) * 2;
207         double zz1 = yc1 * rCos + xc1 * rSin;
208 
209         xt0 *= 16;
210         xt1 *= 16;
211 
212         double zClip = 0.2;
213 
214         if (zz0 < zClip && zz1 < zClip)
215             return;
216 
217         if (zz0 < zClip) {
218             double p = (zClip - zz0) / (zz1 - zz0);
```

```
219             zz0 = zz0 + (zz1 - zz0) * p;
220             xx0 = xx0 + (xx1 - xx0) * p;
221             xt0 = xt0 + (xt1 - xt0) * p;
222         }
223
224         if (zz1 < zClip) {
225             double p = (zClip - zz0) / (zz1 - zz0);
226             zz1 = zz0 + (zz1 - zz0) * p;
227             xx1 = xx0 + (xx1 - xx0) * p;
228             xt1 = xt0 + (xt1 - xt0) * p;
229         }
230
231         double xPixel0 = xCenter - (xx0 / zz0 * fov);
232         double xPixel1 = xCenter - (xx1 / zz1 * fov);
233
234         if (xPixel0 >= xPixel1)
235             return;
236         int xp0 = (int) Math.ceil(xPixel0);
237         int xp1 = (int) Math.ceil(xPixel1);
238         if (xp0 < 0)
239             xp0 = 0;
240         if (xp1 > width)
241             xp1 = width;
242
243         double yPixel00 = (u0 / zz0 * fov + yCenter);
244         double yPixel01 = (l0 / zz0 * fov + yCenter);
245         double yPixel10 = (u1 / zz1 * fov + yCenter);
246         double yPixel11 = (l1 / zz1 * fov + yCenter);
247
248         double iz0 = 1 / zz0;
249         double iz1 = 1 / zz1;
250
251         double iza = iz1 - iz0;
252
253         double ixt0 = xt0 * iz0;
254         double ixta = xt1 * iz1 - ixt0;
255         double iw = 1 / (xPixel1 - xPixel0);
256
257         for (int x = xp0; x < xp1; x++) {
258             double pr = (x - xPixel0) * iw;
259             double iz = iz0 + iza * pr;
260
261             if (zBufferWall[x] > iz)
```

```
262                 continue;
263             zBufferWall[x] = iz;
264             int xTex = (int) ((ixt0 + ixta * pr) / iz);
265
266             double yPixel0 = yPixel00 + (yPixel10 - yPixel00) * pr -
                    0.5;
267             double yPixel1 = yPixel01 + (yPixel11 - yPixel01) * pr;
268
269             int yp0 = (int) Math.ceil(yPixel0);
270             int yp1 = (int) Math.ceil(yPixel1);
271             if (yp0 < 0)
272                 yp0 = 0;
273             if (yp1 > height)
274                 yp1 = height;
275
276             double ih = 1 / (yPixel1 - yPixel0);
277             for (int y = yp0; y < yp1; y++) {
278                 double pry = (y - yPixel0) * ih;
279                 int yTex = (int) (16 * pry);
280                 pixels[x + y * width] = 0xff00ff;
281                 // zBuffer[x + y * width] = 1 / iz * 4;
282             }
283         }
284     }
285
286     public void postProcess() {
287         for (int i = 0; i < width * height; i++) {
288             int col = pixels[i];
289             int brightness = (int) (60000 / (zBuffer[i] * zBuffer[i]));
290             if (brightness < 0)
291                 brightness = 0;
292             if (brightness > 255)
293                 brightness = 255;
294
295             int r = (col >> 16) & 0xff;
296             int g = (col >> 8) & 0xff;
297             int b = (col) & 0xff;
298
299             r = r * brightness / 255;
300             g = g * brightness / 255;
301             b = b * brightness / 255;
302
303             pixels[i] = r << 16 | g << 8 | b;
```

```
304             }
305         }
306     }
```

Hasta aquí el tutorial **:)** El siguiente paso sería crear una clase Block con 6 paredes y una clase Level con un array de Blocks, por último pintar todos los bloques de la clase Level teniendo en cuenta las coordenadas para pintar las paredes formando bloques. Java no es una tecnología que se use mucho en el desarrollo de videojuegos porque su rendimiento no llega a ser del todo óptimo, aunque como ves se puede llegar a hacer cosas bastante interesantes.

Introducción

¿Quieres que tu web cargue rápido cuando entran los usuarios? ¿Estas usando Angular 2 y tu aplicacion carga despacio? Pues en este artículo echaremos un vistazo a las técnicas clave para aumentar el rendimiento. Angular 2 al ser un framework para crear webs SPA (Single Page Application),carga todos los recursos y scripts al iniciar la página de forma que navegar entre las «páginas» sea instantáneo.

Build en producción

Esta técnica es la más sencilla de implementar, simplemente cuando vamos hacer el build (compilar los archivos Angular) para subirlo a producción, tenemos que añadir el siguiente parámetro (Si el proyecto lo hemos creado usando Angular cli):

```
1 ng build --prod
```

Con este parámetro le estamos indicando al compilador de Angular que se optimice para que el tamaño de los archivos compilados sea menor y por tanto que carguen antes.

Si tu versión de angular cli es < 1.0.0 entonces tienes que añadir también otro parámetro:

```
1 ng build --prod --aot
```

Con el parámetro **–aot** hacemos que el compilador revise el código para optimizarlo, en las versiones de angular cli superiores a la 1.0.0 (Angular 4) este parámetro no hace falta ponerlo ya que al poner **–prod** por defecto ya añade el **aot**.

Server Side Rendering

¿Que es server side rendering? Server side rendering (SSR) significa que el los scripts pueden ser ejecutados en un servidor para que cuando un usuario abra la página estén ya cargados, aumentando

el rendimiento. La otra opción, la más habitual, es client side rendering, simplemente significa que el propio usuario carga los scripts en su ordenador. La desventaja principal de server side rendering es que se realizan más llamadas a servidor y por tanto navegar por la web es un poco más lento, pero la carga inicial es mucho más rápida que client side rendering.

Server side rendering:

Client side rendering:

Para hacer que una aplicación web hecha con Angular tenemos que hacer uso de una librería llamada Angular Universal:

```
npm install --save @angular/platform-server @nguniversal/module-map-
    ngfactory-loader ts-loader
```

Lo primero que teenemos que hacer es añadir Angular Universal a AppModule:

```
@NgModule({
  bootstrap: [AppComponent],
  imports: [
    // El appId es un identificador único en la página
    BrowserModule.withServerTransition({appId: 'my-app'}),
    ...
  ],

})
export class AppModule {}
```

El siguiente paso es crear un archivo en el mismo nivel que AppModule llamado app.server.module.ts

```
import {NgModule} from '@angular/core';
import {ServerModule} from '@angular/platform-server';
import {ModuleMapLoaderModule} from '@nguniversal/module-map-ngfactory-
    loader';

import {AppModule} from './app.module';
import {AppComponent} from './app.component';

import { REQUEST } from './request';


export function getRequest() {
  return {cookie: document.cookie};
}
```

```
14
15 @NgModule({
16   imports: [
17
18     AppModule,
19     ServerModule,
20     ModuleMapLoaderModule // <-- *Importante
21   ],
22
23   bootstrap: [AppComponent],
24
25   providers: [
26     {
27       provide: REQUEST,
28       useFactory: (getRequest)
29     }
30 ]
31
32 })
33 export class AppServerModule {}
```

Tambíen creamos un archivo llamado request.ts:

```
1 import { InjectionToken } from '@angular/core';
2
3 export const REQUEST = new InjectionToken<string>('REQUEST');
```

Para configurar el servidor que vamos a crear para cargar los scripts, tenemos que crear un arhivo llamado main.server.ts en la raiz de **src**:

```
1 import { InjectionToken } from '@angular/core';
2
3 export const REQUEST = new InjectionToken<string>('REQUEST');
```

También creamos su archivo de configuracion json, llamado **tsconfig.server.json**:

```
1 {
2     "extends": "../tsconfig.json",
3     "compilerOptions": {
4
5       "outDir": "../out-tsc/app",
6       "baseUrl": "./",
7
```

```
      "module": "commonjs",
      "types": []
    },
    "exclude": [
      "test.ts",
      "**/*.spec.ts"
    ],

    "angularCompilerOptions": {
      "entryModule": "app/app.server.module#AppServerModule"
    }
  }
```

Creamos en la raíz de nuestro proyecto el archivo **server.js**, el servidor express:

```
require('zone.js/dist/zone-node');
require('reflect-metadata');

const express = require('express');
const fs = require('fs');

const { platformServer, renderModuleFactory } = require('@angular/
    platform-server');
const { ngExpressEngine } = require('@nguniversal/express-engine');

const {ModuleMapLoaderModule} = require('@nguniversal/module-map-
    ngfactory-loader');
// Import the AOT compiled factory for your AppServerModule.
// This import will change with the hash of your built server bundle.
const { AppServerModuleNgFactory, LAZY_MODULE_MAP } = require(`./dist-
    server/main.bundle`);

const { provideModuleMap } = require('@nguniversal/module-map-ngfactory
    -loader');

const app = express();
const port = 8000;
const baseUrl = `http://localhost:${port}`;

// Set the engine
app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
```

```
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');

app.set('views', './');
app.use('/', express.static('./', {index: false}));

app.get('*', (req, res) => {
  res.render('index', {
    req,
    res
  });
});

app.listen(port, () => {
  console.log(`Listening at ${baseUrl}`);
});
```

Por último queda modificar los scripts del archivo **packcage.json**, para que ejecuten y compilen adecuadamente el nuevo servidor express que hemos creado

```
"scripts": {
    ...
    "build:universal": "ng build --prod && ng build --prod --app 1 --
       output-hashing=false && cpy ./server.js ./dist",
    "serve:universal": "npm run build:universal && cd dist && node
       server"
    ...
},
```

Si ejecutamos **npm run serve:universal** y abrimos en el navegador **http://localhost:8000/** se abrirá la aplicación angular pero habiendo cargado con anterioridad en el servidor express.

Service workers

Imagina que vas conduciendo un coche con la radio puesta, las canciones van sonando sin que tengas que interaccionar con la radio pero si quieres cambiar de canción cambias la emisora. En este símil, la radio son los service workers, código en segundo plano que se ejecuta sin que el usuario tenga abierta la página. Esto sirve para cargar la web más rápido y para que la web funciona offline. Para añadir los

web workers a la app Angular, primero instala los web workers:

```
npm install @angular/service-worker --save
```

Para habilitarlos, ejecuta (versiones de angular cli superiores a la 1.6):

```
ng set apps.0.serviceWorker=true
```

o añade manualmente en el archivo **angular.cli.josn**:

```
{

  "apps": [

    { "serviceWorker": true }
  ]

}
```

Ahora al hacer build, angular genera nuevos archivos js:

- **ngsw-manifest.json**: Manifest con todos los recursos de nuestra web
- **sw_register**: Archivo que registra el service worker
- **worker-basic**: Archivo de apoyo para el funcionamiento de los service workers.

Si queremos que nuestra app pueda ser instalable en dispositivos móviles, añade un archivo llamado **manifest.json** dentro de la carpeta **src**:

```
{
  "name": "Nombre de la app",
  "short_name": "Nombre corto",
  "theme_color": "#FFFFFF",
  "background_color": "#3F51B5",
  "start_url": "/",
  "display": "standalone",
  "orientation": "portrait",
  "icons": [
    {
      "src": "\/android-chrome-36x36.png",
      "sizes": "36x36",
      "type": "image\/png",
      "density": "0.75"
    },
    {
```

```
17        "src": "\/android-chrome-48x48.png",
18        "sizes": "48x48",
19        "type": "image\/png",
20        "density": "1.0"
21      },
22      {
23        "src": "\/android-chrome-72x72.png",
24        "sizes": "72x72",
25        "type": "image\/png",
26        "density": "1.5"
27      },
28      {
29        "src": "\/android-chrome-96x96.png",
30        "sizes": "96x96",
31        "type": "image\/png",
32        "density": "2.0"
33      },
34      {
35        "src": "\/android-chrome-144x144.png",
36        "sizes": "144x144",
37        "type": "image\/png",
38        "density": "3.0"
39      },
40      {
41        "src": "\/android-chrome-192x192.png",
42        "sizes": "192x192",
43        "type": "image\/png",
44        "density": "4.0"
45      }
46    ]
47  }
```

- **name**: El nombre completo de la app
- **short_name**: Nombre corto que se muestra en el cajón de aplicaciones del movil, junto al icono
- **theme_color**: Color junto al icono de la app
- **background_color**: Color de fondo
- **start_url**: Punto de entrada a nuestra app
- **display**: Standalone para apps móviles
- **orientation**: Orientación de la app
- **icons**: Iconos de la app con sus respectivos tamaños para diferentes aplicaciones

Ahora, al abrir la página web, en chrome, cuando abrimos las opciones de desarrollador, en la pestaña

Application, observaremos que el service worker se ha registrado correctamente:

Si abrimos dos veces la web en el móvil usando Chrome, dejando pasar 5 minutos, aparecerá un cartel preguntando si instalar la app, al aceptar, automáticamente se añadirá la web junto a las aplicaciones del móvil.

Conclusión

Aunque se puede optimizar aún más las aplicaciones Anagular mediante web workers (ejecutar la lógica de Angular en un thread aparte), es más complicado de implementar que las técnicas que he enseñado anteriormente. En mi caso con estas optimizaciones he logrado hacer que la web de Angular cargue en menos de 3 segundos la primera vez que se abre (antes tardaba más de 10 segundos), y que cargue casi instantáneamente cuando la tenemos cargada en la caché (antes tardaba unos segundos).

Introducción

La realidad es que exixten cientos o incluso miles de tecnologías y frameworks distintos para el desarrollo web, contarlas todas sería imposible ya que cada poco tiempo aparecen nuevas tecnologías. En este post vamos a ver las tecnologías más famosas y importantes para que sepas qué opciones tienes a la hora de desarrollar una web o aplicación web. Antes de empezar es importante diferenciar dos conceptos: frontend y backend

El frontend es la parte del lado del cliente en la web, es decir, la apariencia y lo que el usuario ve en la página. El backend en cambio es la parte del lado del servidor, es decir, bases de datos, y toda la lógica de la web no visible para el usuario.

En este artículo voy a mostrar las tecnologías web más populares (a fecha 2018) de cada tipo.

Tecnologías para el frontend

Los navegadores web solo entienden código escrito en HTML, CSS o Javascript, es decir, los frameworks y herramientas expuestos a continuación pueden ser sustituidos por código en Javascript puro.

Frameworks CSS

Archivos CSS ya creados con muchos estilos predefinidos que ayudan al programador. Es decir, contienen estilos y códigos JS listos para ser usados como clases en etiquetas HTML para crear una web de una manera más sencilla.

- **Bootstrap**: Una de las más utilizadas en la actualidad. Buena compatibilidad con el diseño responsive y buena documentación. Recientemente han sacado su versión 4.
- **Foundation**: Alternativa muy buena a Boostrap. Tiene mucha flexibilidad, aunque puede resultar difícil de aprender.
- **Semantic UI**: El mas fácil de aprender, el nombre de sus clases está diseñado para que resulten fáciles de memorizar. Es el que más pesa de los 3.

Frameworks JS/TS

El más conocido y que lleva usándose muchos años es Jquery. Jquery sirve para tener que escribir menos código JS, que para proyectos sencillos está bien. Para proyectos de dimensiones más grandes recomiendo los siguientes frameworks:

- **Angular**: En su primera versión, AngularJS, su tenías que utilizar JS para usarlo, pero en las versiones actuales (Angular 4) se utiliza Typescript con lo cual si Javascript no es tu fuerte, ésta es tu mejor opción ya que la sintaxis de Typescript es muy parecida a Java. Su rendimiento es menor a los frameworks de la competencia.
- **React**: Promovido por Faceebook. A diferencia de Angular, React no es un framework completo. Tiene una comunidad muy completa con un montón de librerías para usar con este framework. Tiene una curva de aprendizaje alta para aplicaciones muy grandes.
- **Vue**: Uno de los frameworks más fáciles de aprender. Se puede usar como sustituto de Jquery o incluso para aplicaciones más grandes. Tiene muchas similitudes con React y con AngularJS. La comunidad en torno a este framework es menor aunque con el tiempo se esta popularizando.
- **Ember**: Sigue las buenas prácticas. Muy estable, aunque difícil de aprender. Antiguamente era muy popular pero con el paso del tiempo su comunidad ha ido disminuyendo.

Task runners

Sirven de gran ayuda para programar ciertas tareas, como por ejemplo, ejecutar analizadores de código cada vez que se guarda un archivo, comprimirlos automáticamente, etc

- **Gulp**: Uno de los más populares actualmente. Usa sintaxis javascript. Más rápido que Grunt.
- **Grunt**: Antiguamente era muy popular pero su comunidad ha ido disminuyendo. Usa una sintaxis JSON. Para proyectos muy grandes puede ser problemático por la longitud de su código.
- **Webpack**: El más reciente. Aunque no es técnicamente un task runner, webpack permite compilar y comprimir muchos archivos con dependencias a archivos js estáticos.

Transpiladores Javascript

Podemos escribir código con otra sintaxis, y gracias a los transpiladores, será transformado a código Javascript

- **Typescript**: Una sintaxis muy parecida a Java. Tiene tipado estático y azúcar sintáctico. No tiene gestor de paquetes.
- **CoffeScript**: Cada vez se usa menos. Añade abstracciones sintácticas y rendimiento a Javascript aunque no añade tipado fuerte estático.
- **Babel**: Babel transpila el código ES6+ a código ES5. El compilador es bastante lento y el código compilado suele tener menos rendimiento.

Transpiladores CSS

De igual manera, también exixten transpiladores para transformar código en CSS, en este caso se usan para aumentar las características de CSS:

- **Less**: Fácil de aprender. Sintaxis parecida a CSS. Escrito en Javascript y muy popular con buena comunidad. No soporta uso de más condicionales, por ejemplo, los ternarios.
- **Sass**: Muy potente. Muy recomendado para aplicaciones ruby. Simplifica el código CSS existente. Para compilar código SCSS es necesario tener instalado ruby.

Tecnologías para el backend

A diferencia de en el frontend, en el backend, existe más variedad de lenguajes con los que podemos crear un servidor conectado a una base de datos. Muchas de estas tecnologías se pueden usar junto con los frameworks para el frontend.

Java

- **Spring**: El más utilizado de esta categoría. Puedes hacer desde un backend con API REST a uno con SOAP. También ofrece inyección de dependencias.

- **Struts**: Se usa mucho en ámbitos empresariales. No es una libería muy ágil pero ahora bastante código a los programadores.
- **Vaadin**: El menos utilizado de estos tres. Ofrece la posibilidad de seguir un patrón MVC o MVP. También ofrece librerías UI y de componentes para facilitar la vida a los programadores.

Python

- **Django**: El más popular dentro de python. Puedes crear aplicaciones web una forma muy sencilla y limpia
- **TurboGears**: Combina muchas librerías de python en una. Muy extensible y configurable.

NodeJS

- **ExpressJS**: El más popular con diferencia. Muchas otras librerías de nodejs parten de ésta. No ofrece integraciones con bases de datos por lo que tendremos que instalar más librerías junto a Express.
- **Koa**: Basado en ExpressJS pero con la finalidad de ofrecer una experiencia minimalista para mejorar el rendimiento. Ofrece soluciones muy eficaces para controlar la ejecución de javascript de forma asíncrona.
- **Sails**: Uno de los más sencillos de utilizar. Con este framework tendremos que seguir el patrón MVC. Por defecto ofrece integraciones con multitud de bases de datos de una forma muy sencilla.

Golang

- **Revel**: El más popular. Esta librería, a diferencia de las demás, viene con un montón de características para que no tengas que instalar librerías adicionales. No ofrece por defecto integración con MongoDB.
- **Gin**: Muy minimalista esto hace que muchas de las cosas las tengas que hacer o buscar por tu cuenta.
- **Martini**: Termino medio entre un ecosistema completo y simplicidad. Es más lento que Gin. No está tan actualizado respecto a los demas frameworks.
- **Gorilla**: Modularidad. Es muy fácil de escalar ya que todo se basa en módulos separados. Ofrece por defecto websockets.

Ruby

- **Rails**: Muy sencillo de utilizar, con ORM para bases de datos. No escala muy bien para aplicaciones muy grandes.

Conclusiones

Como ves existen un montón de frameworks y librerías y todavía exieten muchos más que no he puesto. Escoger uno o otro dependerá de los gustos de cada uno y de la magnitud del proyecto que se vaya a realizar. El mundo web es un mundo en constante cambio, esto quiere decir que las tecnologías que están de moda hoy, mañana pueden no estarlo, por eso es importante tener una mentalidad abierta para adaptarte a los posibles cambios que puedan surgir.

Introducción

Angular 2 (en adelante) soporta la creación de formularios de dos tipos diferentes, mediante templates (como lo hacíamos con AngularJS) o de forma reactiva. Cuando hablamos de formularios reactivos no usamos la directiva ngModel, sino que creamos modelos con los que Angular creará los formularios, manteniendo la lógica de nuestra aplicación web en una sola parte, haciendo que el código sea más fácil de testear y más fácil de mantener.

Formularios reactivos

Para usarlos tenemos que declarlos en el @NgModule:

```
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  imports: [
    ...,
    ReactiveFormsModule
  ],
  declarations: [...],
  bootstrap: [...]
})
export class AppModule {}
```

Antes de continuar es importante definir qué es un FormControl y un FormGroup.

FormControl es un objeto qué se usa en los formularios para tener un control sobre su valor y su estado en el formulario. Para usarlo:

```
const ctrl = new FormControl('some value');
console.log(ctrl.value);     // 'some value'
```

FormGroup es un cojunto de FormControls, el estado de este objeto depende del estado de todos sus objetos, es decir, si uno de los FormControl es inválido, el grupo entero es inválido. Para usarlo:

```
const form = new FormGroup({
  first: new FormControl('Nancy', Validators.minLength(2)),
  last: new FormControl('Drew'),
});
```

¿Y cómo podemos conectar estos oobjetos con los formularios del HTML? Es bastante sencillo, por ejemplo:

```
<form novalidate [formGroup]=”group”>
  Name: <input type=”text” formControlName=”name”>
  Location: <input type=”text” formControlName=”location”>
</form>
```

En el componente creamos un formGroup con el nombre group con dos FormControl, name y location. Como ves, ya no utilizamos ngModel para conectar los inputs con sus valores del componente.

Para entender todo esto mejor vamos a usar un ejemplo real.

Ejemplo

Imaginemos que queremos crear un formulario de registro, para ello creamos la siguiente interfaz:

```
export interface User {
  name: string;
  password: string;
  passwordRepeat: string,
  }
}
```

Ahora en nuestro componente creamos un FormGroup con FormControl para cada uno de los campos:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';

@Component({...})
export class SignupFormComponent implements OnInit {
  user: FormGroup;
  ngOnInit() {
    this.user = new FormGroup({
      name: new FormControl(''),
      password: new FormControl(''),
      passwordRepeat: new FormControl('')
    });
  }
}
```

Ahora creamos el formulario en el HTML del template y conectamos los campos como hemos hecho antes:

```
<form novalidate [formGroup]=”user”>
```

```
  <label>
    <span>Full name</span>
    <input
      type="text"
      placeholder="Your full name"
      formControlName="name">
  </label>
  <label>
      <span>Email address</span>
      <input
        type="password"
        placeholder="Your passwornd"
        formControlName="password">
    </label>
    <label>
      <span>Confirm address</span>
      <input
        type="password"
        placeholder="Confirm your password"
        formControlName="passwordRepeat">
    </label>
  <button type="submit">Sign up</button>
</form>
```

¿Y qué pasa con el botón de submit? Pues se hace exactamente igual que al hacerlo con templates:

```
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="user">
...
</form>
```

Validadores

Lo primero para validar es importar en el componente los validadores:

```
import { FormControl, FormGroup, Validators } from '@angular/forms';
```

Para validar la información de los campos del formulario, podemos usar las funciones que nos ofrece Angular o implementar las nuestras propias, para ello cambiamos los objetos:

```
ngOnInit() {
  this.user = new FormGroup({
```

```
    name: new FormControl('', [Validators.required, Validators.
        minLength(2)]),
      password: new FormControl('', Validators.required),
      passwordRepeat: new FormControl('', Validators.required)
  });
}
```

Con Validators.requiered aseguramos que el dato exista y con minLength aseguramos que al menos el usuario introduzca dos caracteres. Como puedes ver, si queremos añadir más de un validador, lo podemos hacer mediante un array de validadores.

Si ahora queremos añadir una contraseña para comprobar que las contraseñas coinciden es tan fácil como hacer:

```
function passwordMatchValidator(g: FormGroup) {
   return g.get('password').value === g.get('passwordRepeat').value
      ? null : {'mismatch': true};
}
```

Para usarlo lo metemos con los otros validadores y listo.

También podemos hacer que el botón de submit esté deshabilitado mientras que no esté todo validado:

```
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="user">
  ...
  <button type="submit" [disabled]="user.invalid">Sign up</button>
</form>
```

Para mostrar, por ejemplo, mensajes de error personalizados:

```
<div
  class="error"
  *ngIf="user.get('name').hasError('required') && user.get('name').
     touched">
  Nombre requerido
</div>
```

Añadiendo estilos css a gusto de cada uno, el aspecto que tiene el formulario es este:

FormBuilder

Ahora que entendemos bien estos conceptos, podemos que dejar que Angular añada azúcar sintáctico para que cree el FormGroup y el FormControl.

Para ello vamos a hacer uso del FormBuilder, primero lo importamos junto con lo que vamos a necesitar:

```
1 import { FormBuilder, FormGroup, Validators } from '@angular/forms';
```

A continuación lo inyectamos en el constructor:

```
1  constructor(private fb: FormBuilder) {}
```

Ahora refactorizamos el código para usar el FormBuilder:

```
1 ngOnInit() {
2  this.user = this.fb.group({
3    name: ['', [Validators.required, Validators.minLength(2)]],
4      password: ['', Validators.required],
5      passwordRepeat: ['', Validators.required]
6  });
7 }
```

Y ya estaría, como ves no añade funcionalidad nueva, simplemente es una manera de dejar el código más bonito y mantenible.

Conclusiones

Personalmente, recomiendo el uso de FormBuilder (o en su defecto FormGroup con FormControl) sobre NgModel, sobre todo para formularios, ya que Angular proporciona mecanismos que facilitan la validación de los parámetos y el estado de los mismos, haciendo que el código sea mucho más visual y limpio.

Qúe es un componente

Como vimos anteriormente, Angular se basa en componentes. Un componente on un conjunto de características que actualmente están siendo añadidas por el W3C, se basa en la creación de fragmentos con vista, estilos y controladores de forma que puedan ser reutilizadas en distintas partes de la web. Por ejemplo, pongamos que creamos un componente que reciba una lista de elementos y los pinte en

el html. Una vez creado podemos añadir este componente de lista que hemos creado en varios sitios de tal forma que le podemos pasar los elementos que queremos pintar para que los pinte. Esto ofrece la ventaja de no tener el código modular, es decir, si tenemos que efectuar un cambio en la manera en la que visualizamos las listas, por ejemplo, solo lo tenemos que realizar una vez para todas las listas.

Un componente se puede componer de varios archivos: vista, estilos, controladores, servicios, etc.

La vista (html) y los estilos (css), definen qué y cómo queremos representar la web. En los controladores se encuenta la lógica de los componentes. Desde este archivo podemos inicializar las variables para la vista, actualizarlos, llamar a otros archivos, crear funciones, etc. Desde los servicios es donde se hacen las llamadas para gestionar los datos, por ejemplo guardar datos, es decir, desde los controladores es mejor no gestionar datos directamente, sino que lo mejor es delegar estas funciones a los servicios.

Creando componentees en Angular

Para crear los componentes en Angular existen dos maneras de hacerlo: la método sencillo y el manual.

Método sencillo

Si estamos usando Angular cli, en nuestro proyecto, existe un comando para generar componentes:

Por ejemplo, imaginemos que queremos crear un componente para mostrar el navbar en todas las páginas:

```
1 ng generate component navbar
```

Si navegamos ahora a la carpeta **app** del proyecto veremos que Angular cli ha creado por nosotros una carpeta llamada **navbar**. Dentro de la carpeta navbar se ha creado un archivo css, un archivo html y un archivo .ts (controlador) junto con un .spec.ts (tests). Otro detalle a tener en cuenta es que Angular ha importado por nosotros el nuevo componente en el archivo **app.module.ts**.

Método manual

Como podrás imaginar, este método consiste en crear los archivos que te hagan falta manualmente para el componente. También tendrás que importarlo manualmente en el archivo **app.module.ts**.

Angular recomienda separar los componentes en carpetas según su funcionalidad, de esta forma será más fácil localizarlos y el código será más fácil de mantener.

Para crear un controlador vacío (archivo .ts) la estructura es la siguiente (por ejemplo para el componente de navbar):

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})

export class NavbarComponent  {

  constructor() { }

}
```

Importamos «Component» de @angular/core, llamamos a @Component y le pasamos tres cosas (de momento):

- **selector**: El selector es el nombre que va a tener la etiqueta html que se crea con el componente, es decir, para el navbar será `<app-navbar></app-navbar>`, es decir desde el html de cualquier otro componente, poniendo esa etiqueta se pintará el navbar. Angular tiene una convención de nombre para el selector, kebab-case, es decir el nombre de los selectores tiene que ser una palabra guión otra palabra: algo-ejemplo.
- **templateUrl**: La url de la vista html asociada al componente.
- **styleUrl**: La url del estilo CSS asociado al componente.

Por último hacemos export class y el nombre de la clase.

Routing en Angular

Con esto sabemos crear componentes sueltos (todavía sin funcionalidad) pero qué pasa si hemos creado un componente para una página entera, es decir, ¿cómo se crean páginas en Angular, y cómo asignarles una ruta en la página?

Para ello necesitamos hacer uso del routing de Angular, necesitamos el archivo con las rutas.

Creamos un archivo al mismo nivel que el **app.module.ts** y lo llamamos **app.routing.ts**. Una vez creado importamos las rutas de Angular:

```
import { RouterModule, Routes } from '@angular/router';
```

Ahora, creamos una variable con las rutas:

```
const appRoutes = [
    { path: '', component: ItemListComponent,  pathMatch: 'full'}
];
```

- **path**: La ruta a que queremos configurar
- **component**: Componente asociado a esa ruta. Para que funcione tienes que importar el componente en la parte de arriba, por ejemplo:

  ```
  import { ItemListComponent } from './item-list/item-list.component
      ';
  ```

- **pathMatch**: Esto es opcional, significa que toda la ruta URL tiene que coincidir.

Ahora, imaginemos que queremos crear una página para mostrar en detalle los items de la lista, entonces necesitamos que Angular genere una ruta para cada item, eso se puede hacer de la siguiente manera:

```
  { path: 'hero/:id', component: ItemDetailComponent }
```

:id indica que se generarán rutas con distintos id, luego dentro del controlador del detalle de item, recogeremos este valor y mostraremos el item correspondiente. Pero esto lo veremos en detalle en artículos posteriores.

Y si queremos iuna página 404 que aparezca cuando una ruta no coincide con alguna de las anteriores:

```
  { path: '**', component: PageNotFoundComponent }
```

Al final del fichero introducimos:

```
export const routing = RouterModule.forRoot(appRoutes);
```

El fichero completo, sin la ruta a la página 404, quedaría de la siguiente manera:

```
import { RouterModule } from '@angular/router';
import { AppComponent } from './app.component';
import { ItemListComponent } from './item-list/item-list.component';
import { ItemDetailComponent } from './item-detail/item-detail.
    component';

const appRoutes = [
    { path: '', component: ItemListComponent,  pathMatch: 'full'},
```

```
    { path: '', component: ItemListComponent, },
];
export const routing = RouterModule.forRoot(appRoutes);
```

Ahora, tenemos que importar las rutas en el archivo **app.module.ts**, para ello importamos la ruta y lo añadimos, esta vez en la parte de **imports**:

```
import { routing } from './app.routing';

...

  imports: [
    BrowserModule,
    routing
  ],
```

Si pruebas las páginas con estos cambios te darás cuenta de que todavía no se muestran las nuevas rutas, esto pasa porque en el archivo **app.component.html** que es el primer componente que se carga, tenemos que quitar el html que viene por defecto para poner una etiqueta especial:

```
<router-outlet></router-outlet>
```

Ahora sí que se tendrían que cargar todas las rutas.

Dentro de esta etiqueta especial es donde se cargarán las vistas de las rutas. Por ejemplo antes de esta etiqueta podemos meter el navbar poniendo una etiqueta con el nombre del selector que tenga el navbar, por ejemplo:

```
<app-navbar></app-navbar>
```

¿Y en el navbar cómo podemos poner links a páginas de nuestra web?

Ya no podemos usar el atributo href, a no ser que queramos navegar a una página fuera de la web.

Para poner un link ahora tenemos que usar:

```
  <a routerLink="/list" routerLinkActive="active">List</a>
```

De esta manera, por ejemplo navegaremos a la url list, y si la tenemos configurada en el **app.routing.ts** se cargará su vista dentro del router-outlet visto anteriormente.

Conclusiones

Con esto ya sabemos cómo crear componentes y como asignarlos a rutas, aunque ésto es solo una parte de todo lo que se puede hacer con rutas y componentes. Si quieres obtener más información sobre estos temas visita la documentación oficial de Angular (en inglés):

https://angular.io/guide/router

Qué es Angular

Angular es un framework para la creación de páginas web SPA mantenido por Google. SPA es el acrónimo de «Single Page Application» o lo que es lo mismo, cuando un usuario entra en una web SPA, se carga todo a la vez en una misma página y Angular lo que hace pòr debajo es cambiar la vista al navegar por la página para que de la apariencia de una web normal. ¿Qué ventajas tiene esto?

- Velocidad de carga lenta la primera vez que se abre la web, pero luego navegar por la web es instantáneo debido a que se carga toda la web de golpe.
- Cómo SPA es una página solo hay una ruta que tiene que enviar el servidor.
- Aplicaciones modulares y escalares.

Entre las características de Angular destacan:

- Lenguaje Typescript, tiene una sintaxis muy parecida a Java, con tipado estático.
- Sigue el patrón MVC, con la vista separada de los controladores.
- Basado en componentes, es decir, podemos escribir componentes web con vista y lógica para después reutilizarlos en otras páginas.
- Comunidad muy grande con multitud de tutoriales y librerías.
- Inyección de dependencias, un patrón de diseño que se basa en pasar las dependencias directamente a los objetos en lugar de crearlas localmente.
- Programación reactiva, la vista de actualiza automáticamente a los cambios.
- Disponde de asistente por línea de comandos para crear proyectos base.
- Se integra bien con herramientas de testing.
- Se integra bien con Ionic, para adaptar aplicaciones web a dispositivos móviles.

Cómo se instala

Para instalarlo tenemos que disponer de Node y NPM instalados en el equipo. Si no lo tienes instalado, puedes decargar las dos herramientas desde aquí: https://www.npmjs.com/get-npm.

Una vez instalado NPM ejecuta la terminal y escribe:

```
npm install -g @angular/cli
```

Este comando instalará Angular cli de forma global en nuestro equipo. Angular cli es la herramienta de consola de Angular que nos ayudará en la programación con Angular.

Para que Angular cli cree un proyecto vacío de base para crear una aplicación con Angular, escribe:

```
ng new NOMBRE_APP
```

Cambia NOMBRE_APP por el nombre que le quieras dar a tu aplicación.

Ahora muévete en la terminal con el comando cd a la carpeta que se acaba de crear y ejecuta:

```
npm install
```

Este comando servirá para que se instalen en el proyecto las dependencias que hagan falta.

Por último para ejecutar la aplicación web que acabamos de crear simplemente:

```
ng serve --open
```

El flag –open sirve para que se abra automáticamente el navegador web con la web. Por defecto Angular se ejecuta en el puerto 4200.

Si todo ha ido bien, veremos una web como esta (si no se te ha abierto el nevegador, abre http://localhost:4200/):

Hello world

Cuando generamos un proyecto con Angular cli nos genera la siguiente estructura (en mi caso he llamado a la aplicación tutoApp:

Voy a pasar a explicar por encima para que sirve cada archivo y carpeta:

- **e2e**: Esta carpeta por el momento no nos es útil, aqui se encuentra el código para escribir tests end to end que prueben la aplicación
- **node_modules**: En esta carpeta se encuentran las librerías de angular y sus dependencias, cuando instalemos librerías se añadiran aquí. Generalmente no hay que tocar nada de esta carpeta.
- **src**: Aquí se encuentran los archivos que componen nuestra aplicación
  - **app**: Aquí se donde se van a encontrar los componentes, vistas, y servicios de la app. Por el momento hay un componente llamado app con sus respectivos archivos (css, html controlador, tests, etc)

        * **app.module.ts**: En este archivo se especifica los componentes que vamos a usar en la app web. Cuando creemos un componente tenemos que importarlo en este archivo.
    - **favicon**: El favicon de la web
    - **index.html**: Punto de entrada a nuestra web, este archivo se carga en todas las webs, por lo que puedes poner código para que se incluya en todas las vistas.
    - **main.ts**: Algunas configuraciones de Angular, de momento no nos hace falta tocarlo.
    - **polyfills.ts**: Configuraciones y código que se ejecutará antes de que se inicie la app. De momento tampoco nos hace falta tocarlo.
    - **styles.css**: Estilos css globales que se aplicarán en toda las vistas de la página.
    - **test.ts**: Configuración para los tests. No es útil de momento.
    - **tsconfig.app.json, tsconfig.spec.json y typings.d.ts**: Lo mismo que el anterior.
- **.angular-cli.json**: Archivo de configuración de la app.
- **.editorconfig**: Configuraciones a la hora de desarrollar, por ejemplo, como van a ser las identaciones.
- **.gitignore**: Archivo para que git ignore ciertas carpetas que no hace falta subir, como node_modules (cuando te bajas el proyecto ejecutas npm install para que descargue las dependencias en node_modules).
- **karma.conf.js**: Más configuraciones para los tests, esta vez los de Karma.
- **package-lock.json**: Árbol de dependencias que se crea automáticamente
- **package.json**: Archivo con las dependecias instaladas y los comandos que se pueden ejecutar con npm
- **protractor.conf.js**: Configuración para protractor, una herramienta para realizar tests en el navegador.
- **README.md** Archivo readme con información de la aplicación.
- **tsconfig.json**: Configuración para Typescript, el lenguaje de Angular.
- **tslint.json**: Configración del linter de TypeScript (un linter sirve para hacer comprobaciones del estilo del código que escribimos).

Ahora si abres el archivo **app.component.ts** situado en la carpeta src/app y cambias el string de:

```
1   title = 'app';
```

por:

```
1   title = 'my wonderful app';
```

por poner un ejemplo, si ahora abres la página (si no tienes funcionando el comando **ng serve**, ejecútalo), verás que ahora en la página pone **Welcome to my wonderful app!**

Como ves, existe una variable llamada **title** (Typescript tiene inferencia de tipos y no hace falta que

especifiques de que tipo es la variable) que automáticamente se pinta en el html, para ello si abres el archivo **app.component.html** verás que en la 4ª linea hay:

```
1 Welcome to {{ title }}!
```

Con {{ nombre_variable }} puedes pintar variables creadas en el controlador (archivos ts) de su correspondiente componente (en este caso el componente es app).

Conclusiones

Recapitulando, hemos visto qué es Angular, como instalarlo, cómo crear el esqueleto de una app, y una idea aproximada de para qué sirven los archivos y carpetas que crea por defecto. Te animo a que pruebes y cambies cosas del código para que vayas viendo como funciona Angular y Typescript. Si quieres encontrar más info de Angular lo puedes hacer en su página oficial: https://angular.io/