**Parte1**



# Introducción

# Instalación de Node JS

Para trabajar con NodeJs debemos instalarlo. Descargamos el instalador de **Node 10 versión recomendada** <https://nodejs.org/es/download/>

# WebStorm

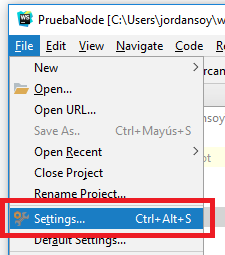
Descargamos el Webstorm <https://www.jetbrains.com/webstorm/>

Con la licencia de estudiante se puede usar de forma gratuita (de lo contrario es una versión de prueba valida durante 30 días)

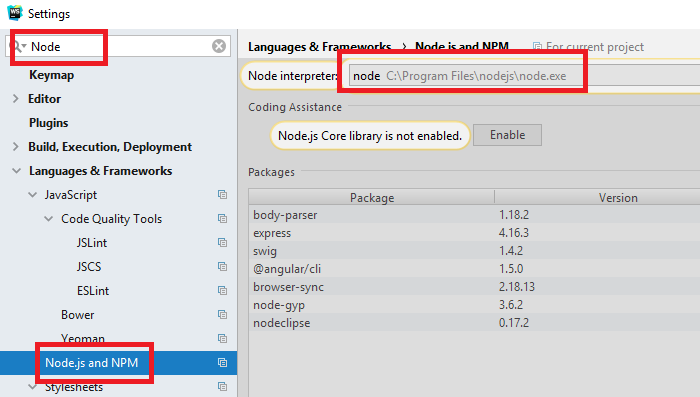
<https://www.jetbrains.com/student/>

## Validar la configuración de Node

Abrimos el Webstorm , Pulsamos en **File -> Open**



Buscamos **Node** en el cuadro de texto y validamos que la ruta es correcta



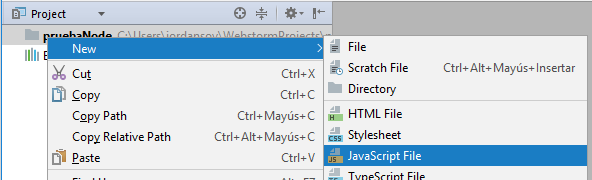
## Crear proyectos e importar Hapi

Si queremos crear un proyecto Node vacío en WebStorm pulsamos en **File -> New Project**

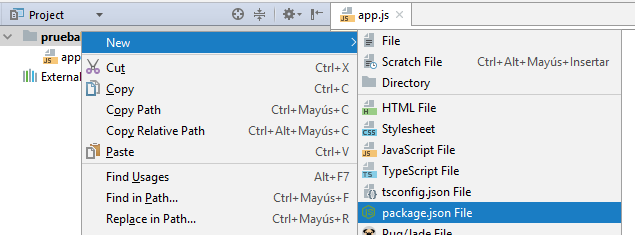
Seleccionamos el tipo **Empty Project**



Incluimos un nuevo fichero de tipo **JavaScript File** y lo llamamos **main.js**

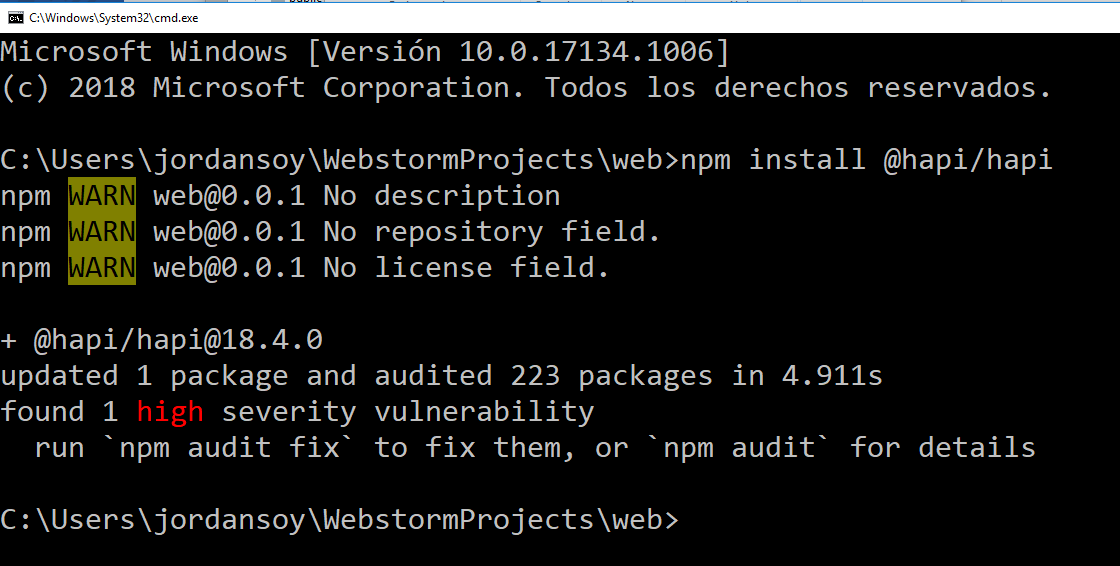


Incluimos un nuevo fichero de tipo **package.json**

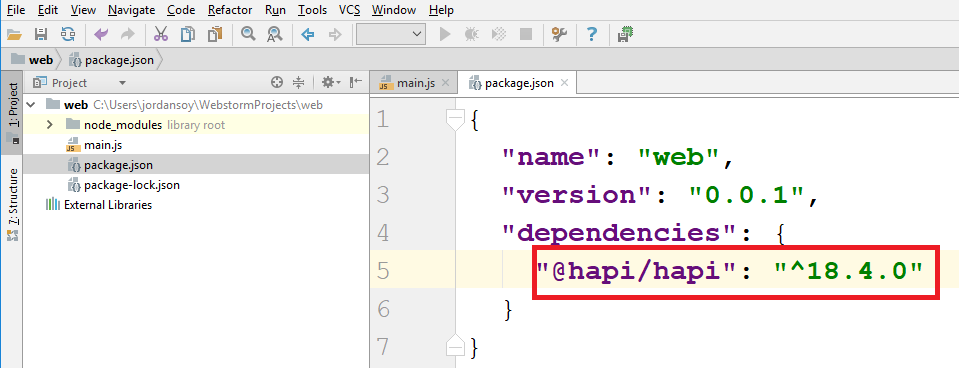


Para instalar un módulo abrimos el CMD, nos colocamos en la **carpeta raíz** del proyecto desde línea de comandos y ejecutamos el comando:

npm install @hapi/hapi@18.4.0



Los ficheros relativos al módulo aparecerán en la carpeta **node\_modules** y la dependencia se habrá agregado al **package.json**



Incluimos el siguiente código en el fichero **main.js.** La función **require** se utiliza para introducir el uso de módulos, en este caso el módulo “@hapi/hapi”, está función retornará la variable con el módulo hapi.

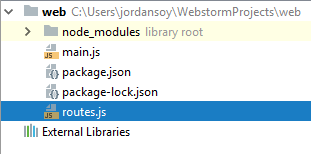
Para configurar el servidor ejecutamos la función **hapi.server(objeto de configuración).**

Para lanzar el servidor utilizamos la función **hapi.start().**

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**);  *// Servidor* **const** server = Hapi.server({  **port**: 8080,  **host**: **'localhost'**, });  **const** *iniciarServer* = **async** () => {  **try** {**await** server.start();  ***console***.log(**'Servidor localhost:8080'**);  } **catch** (error) {  ***console***.log(**'Error '**+error);  } };  *iniciarServer*(); |

# Rutas

Creamos un nuevo fichero en la carpeta raíz del proyecto, lo llamaremos **routes.js**



# El fichero va a contener un módulo. Para integrar un módulo en hapi es obligatorio colocarle un **name** , opcionalmente podemos crear una fusión **register()** que se ejecutará automáticamente al añadir el módulo a la aplicación.

En este caso la función **register()**, se va a encargar de ir añadiendo todos los controladores (rutas) al servidor, a través de la función **server.route( array de funciones rutas ).**

Cada ruta tiene un **method, path, y handler** (la función que procesa la respuesta **req** y genera una respuesta).

Las funciones **handler** están obligadas a retornar una respuesta o una promesa.

|  |
| --- |
| module.exports = {  **name**: **'MiRouter'**,  register: **async** (server, options) => {  server.**route**([  {  **method**: **'GET'**,  **path**: **'/'**,  handler: **async** (req, h) => {  **return 'Hola Mundo'**;  }  }  ])  } } |

Agregamos el nuevo módulo **routes.js** al fichero principal de la aplicación **main.js.**

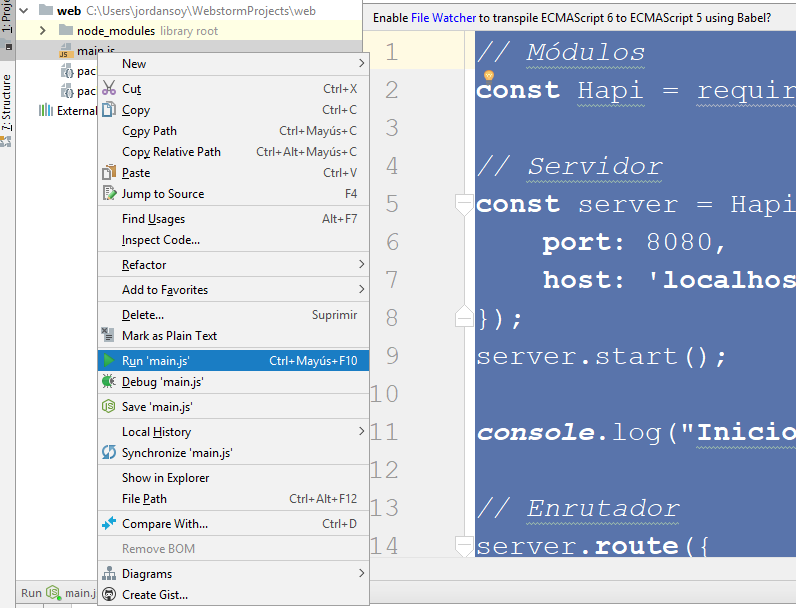
Todos los módulos se inician con **require(nombre(path)**, una vez tenemos el objeto route lo registramos en el server con **server.register(objeto).**

**await** se utiliza para ejecutar funciones asíncronas deteniendo la ejecución hasta que estas retornan su respuesta.

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**); **const routes** = require(**"./routes.js"**);  *// Servidor* **const** server = Hapi.server({  **port**: 8080,  **host**: **'localhost'**, });  **const** *iniciarServer* = **async** () => {  **try** {  **await** server.register(**routes**);   **await** server.start();  ***console***.log(**'Servidor localhost:8080'**); |

# Ejecución de la aplicación

Para ejecutar la aplicación botón derecho sobre el fichero principal **main.js** y pulsar en **Run** ‘**main.js**’



Abrimos <http://localhost:8080/>

# Peticiones GET y parámetros

Las peticiones GET pueden contener parámetros en su URL. Existen dos formas comunes de enviar parámetros (**req – request**), una de ellas es envía la clave y el valor del parámetro, como puede verse en los siguientes ejemplos:

* <http://localhost:8080/?nombre=Donald>   
  *Parámetro con clave “nombre” y valor “despacito”, se agrega con el operador* ***?***
* <http://localhost:8080/?nombre=Donald&apellido=>Trump

*Igual que el ejemplo anterior, pero con un parámetro 2 con clave “autor” y valor “Luis Fonsi”, todos los parámetros a partir del primero se concatenan con el operador &*

Para obtener los parámetros GET hace falta utilizar el objeto **req.query.<clave\_parámetro>** . Por ejemplo, para obtener el parámetro **nombre** y **apellido** de la petición **GET /** deberíamos hacer lo siguiente:

|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/'**,  handler: **async** (req, h) => {  **return 'Hola '**+req.**query**.nombre  + **" "**+req.**query**.apellido;  }  } ]) |

Probamos a ejecutar las siguientes URLs:

<http://localhost:8080/?nombre=Donald>

<http://localhost:8080/?nombre=Donald&apellido=>Trump

Los parámetros son opcionales, en el caso de no encontrar los parámetros solicitados con **req.query.<clave\_del\_parámetro>.**  en la petición nos retornará “**undefined**”. Para comprobar que la variable tiene valor podemos comprobar si el valor es distinto de null, o si el tipo **typeof()** es distinto de “undefined”.

|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/'**,  handler: **async** (req, h) => {  **respuesta** = **"Hola "  if** (req.**query**.nombre != **null**)  **respuesta**+=req.**query**.nombre  **if** (req.**query**.apellido != **null**)  **respuesta**+=**" "**+req.**query**.apellido   **return respuesta**;  }  } ]) |

Todos los valores que obtenemos a través del **req.query** son cadenas de texto, si quisiéramos tratarlas como enteros habría que convertirlas, con **parseInt(req.query.<clave>)** .

# Parámetros en URL

Otra forma de enviar parámetros GET es embeberlos en la URL entre /<valor\_parámetro>/ **sin especificar que Clave tienen**, se suele utilizar para ids y categorización, es la implementación del controlador la que determina la clave del parámetro en función de su posición en la URL.

* <http://localhost:8080/anuncio/1>  
  *Canción con parámetro “id” = 121*

Estos parámetros se corresponderían con estas claves sí la aplicación así lo especificase, los parámetros se deben especificar el la ruta con **:<clave\_del\_parámetro>** y se accede a ellos en el código utilizando **req.params.<clave\_del\_parámetro>.**

Agregamos un nuevo controlador para responder a la petición **GET/anuncio/{id}.**

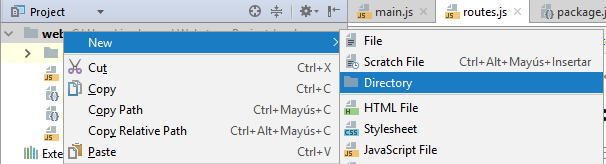
|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/anuncio/{id}'**,  handler: **async** (req, h) => {  **return 'Anuncio id: '** + req.**params**.**id**;  }  },  {  **method**: **'GET'**,  **path**: **'/'**,  handler: **async** (req, h) => {  **respuesta** = **"Hola "** |

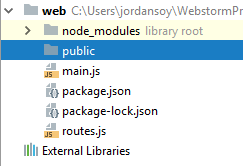
Probamos que funciona accediendo a : <http://localhost:8080/anuncio/1>

# Recursos estáticos

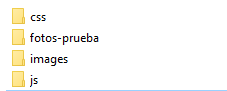
Hapi provee una función de asistencia (middleware) para facilitar el acceso a los clientes a ficheros estáticos, como pueden ser páginas HTML estáticas, imágenes, videos, css, etc.

Creamos el directorio **public** (por convenio se suele utilizar ese nombre para la carpeta publica) en la raíz del proyectoy descomprimimos dentro el contenido del fichero **recursos.zip** descargado del campus virtual.

****

****

Copiamos dentro todos los ficheros descargados del campus virtual.



En **main.js** vamos importaremos el módulo **@hapi/inert**  **, necesario para poder gestionar los ficheros.** <https://www.npmjs.com/package/@hapi/inert>

El primer paso es instalar el módulo en el proyecto.

**npm install @hapi/inert@5.2.2**

Agregamos el módulo y lo registramos en el servidor.

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**); **const** Inert = require(**'@hapi/inert'**); **const routes** = require(**"./routes.js"**);  *// Servidor* **const** server = Hapi.server({  **port**: 8080,  **host**: **'localhost'**, });  **const** *iniciarServer* = **async** () => {  **try** {  *// Registrar el Inter antes de usar directory en routes* **await** server.register(Inert);  **await** server.register(**routes**);  **await** server.start();  ***console***.log(**'Servidor localhost:8080'**);  } **catch** (error) {  ***console***.log(**'Error '**+error);  } }; |

Ahora nos dirigimos **routes.js** y creamos un nuevo controlador especial, este controlador simplemente busca los ficheros en **./public .**

|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/{param\*}'**,  **handler**: {  **directory**: {  **path**: **'./public'** }  }  },  {  **method**: **'GET'**,  **path**: **'/anuncio/{id}'**, |

Ejecutamos la aplicación y comprobamos que podemos acceder a los recursos de esta carpeta.

[http://localhost:8080/images/destacado1.png  
http://localhost:8080/images/destacado2.png](http://localhost:8080/images/destacado1.pnghttp://localhost:8080/images/destacado2.png)  
El contenido de las carpetas estáticas se puede modificar sin falta de reiniciar la aplicación.

# Vistas y Motores de plantillas

Una de las formas más comunes para intercalar datos obtenidos por medio de la ejecución de lógica de negocio en ficheros de presentación HTML son las plantillas. Las plantillas nos permiten combinar texto de salida (por ejemplo HTML) con tratamiento de datos procedentes de la lógica , aplicando difernetes etiquetas, funciones y filtros.

Podemos combinar el framework hapi con muchos motores de plantillas <https://hapi.dev/tutorials/views/?lang=en_US>

En este caso vamos a usar **handlebars** <https://handlebarsjs.com/> , se trata de un motor tremendamente simple y no demasiado potente.

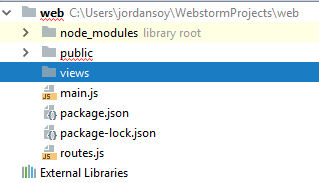
El primer paso es instalar el módulo **@hapi/vision** , nos permitirá integrar un motor de plantillas.

npm install @hapi/vision@5.5.4

Instalamos también el módulo **handlebars** del motor de plantillas (no tiene nada que ver con hapi)

npm install handlebars@4.4.3

Creamos un directorio **/views/** para guardar en él las plantillas.

  
Dentro del directorio creamos el fichero **anuncios.html**, va a ser nuestra primera plantilla.



Está plantilla contendrá HTML e instrucciones {{ }} .

|  |
| --- |
| <**html**>  <**head**>  <**title**>Wallapap</**title**>  </**head**> <**body**>  <**h1**>{{ sitioWeb }}</**h1**>  <**h2**>{{ usuario }}</**h2**>  <**p**> Ejemplo de plantilla </**p**>  <**ul**>  {{#each anuncios}}  <**li**>  {{ titulo }} - {{ precio }}  </**li**>  {{/each}}  </**ul**> </**body**> </**html**> |

Se pueden incluir muchas otras etiquetas de procesamiento.

Ahora vamos a utilizar la plantilla en la aplicación, abrimos **main.js** e introducimos el módulo **@hapi/vision** (require para obtener el módulo y server.register para registrarlo).

Después configuramos el motor de vistas desde **server.views**, básicamente nos solicita el módulo del motor de vistas a utilizar y donde se van a encontrar las plantillas.

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**); **const** Inert = require(**'@hapi/inert'**); **const** Vision = require(**'@hapi/vision'**); **const routes** = require(**"./routes.js"**);  *// Servidor* **const** server = Hapi.server({  **port**: 8080,  **host**: **'localhost'**, });  **const** *iniciarServer* = **async** () => {  **try** {  *// Registrar el Inter antes de usar directory en routes* **await** server.register(Inert);  **await** server.register(Vision);  **await** server.register(**routes**);  **await** server.views({  **engines**: {  **html**: require(**'handlebars'**)  },  **relativeTo**: \_\_dirname,  **path**: **'./views'**,  **context** : {  **sitioWeb**: **"wallapep"** }  });  **await** server.start();  ***console***.log(**'Servidor localhost:8080'**);  } **catch** (error) {  ***console***.log(**'Error '**+error);  } }; |

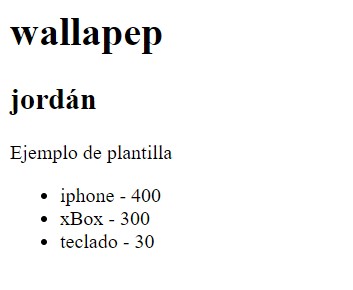
\*El objeto **context** nos permite guardar información accesible desde cualquier plantilla, en este caso el atributo **sitioWeb** se puede consultar desde todas las plantillas.

Vamos a **routes.js** e introducimos un nuevo controlador **GET /anuncios** que use la plantilla **auncios.**

Para responder una plantilla utilizamos **h.view** , le podemos enviar como parámetros los datos que van a ser usados en la plantilla, en este caso usuario y una lista de anuncios.

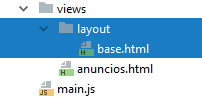
|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/anuncios'**,  handler: **async** (req, h) => {   **anunciosEjemplo** = [  {**titulo**: **"iphone"**, **precio**: 400},  {**titulo**: **"xBox"**, **precio**: 300},  {**titulo**: **"teclado"**, **precio**: 30},  ]   **return** h.view(**'anuncios'**,  {  **usuario**: **'jordán'**,  **anuncios**: **anunciosEjemplo** });  }  },  {  **method**: **'GET'**,  **path**: **'/{param\*}'**, |

Sí guardamos los cambios, ejecutamos la aplicación y accedemos a [http://localhost:8080/anuncios](http://localhost:8080/anuncios%20)  deberíamos ver la plantilla correctamente renderizada.



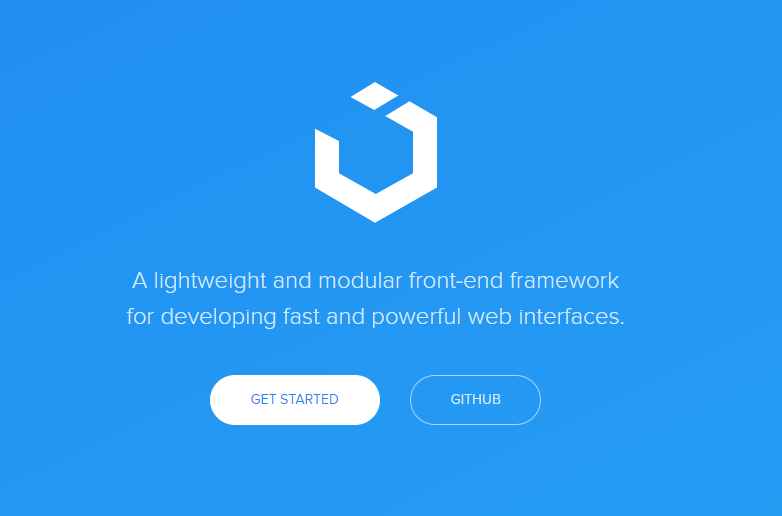
# Base de la plantilla con UiKit

Vamos a crear una nueva plantilla **/views/layout/base.html**



Está plantilla va a ser la base de todas las vistas de la aplicación.

Comenzamos incluyendo una cabecera muy básica (con todos los js de Uikit y jQuery). Para el diseño de todas las vistas utilizaremos el framework **Uikit 3** (versión 3) <https://getuikit.com/docs/introduction> , poco a poco iremos introduciendo características del framework.

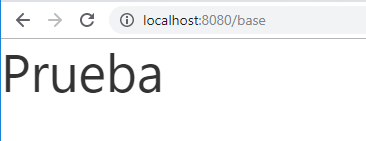


|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**title**>Wallapap</**title**>  <**meta charset="utf-8"**/>  <**meta name="viewport" content="width=device-width, initial-scale=1"**/>  <**link rel="stylesheet" href="/css/uikit.css"**/>  <**script src="/js/jquery.min.js"**></**script**>  <**script src="/js/uikit.js"**></**script**>  <**script src="/js/uikit-icons.js"**></**script**> </**head**> <**body**>  <**h1**> Prueba </**h1**> </**body**> </**html**> |

Creamos un nuevo controlador en **routes.js** , será únicamente para hacer pruebas, podemos utilizar la forma abreviada de retornar una vista (se hace con un objeto, sin implementar ninguna función).

|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/base'**,  **handler**: {  **view**: **'layout/base'** }  },  {  **method**: **'GET'**,  **path**: **'/anuncios'**, |

Probamos a acceder a <http://localhost:8080/base>



Seguimos modificando **base.html**, comenzamos incluyendo un header.

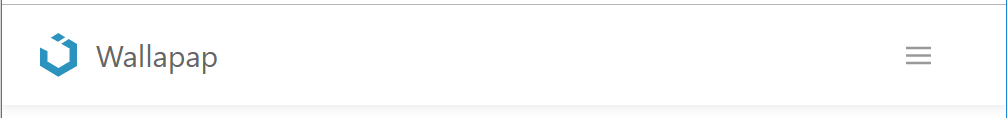
|  |
| --- |
| <**body**>  *<!--HEADER-->* <**header class="uk-box-shadow-small"**>  <**div class="uk-container uk-container-expand"**>  <**nav class="uk-navbar" data-uk-navbar**>  <**div class="uk-navbar-left"**>  <**img width=30" height="30" src="images/uikit.svg" alt=""**>  <**a class="uk-navbar-item uk-logo" href="#"**>Wallapap</**a**>  </**div**>   <**div class="uk-navbar-right"**>  *<!-- uk-visible@m visible en pantallas m o más-->* <**ul class="uk-navbar-nav uk-visible@m"**>  <**li**>  <**a href="/seguidos"**>  <**span data-uk-icon="heart"  class="uk-margin-small-right"**>  </**span**> Seguidos  </**a**>  </**li**>  <**li**>  <**a href="/misanuncios"**>  <**span data-uk-icon="list"  class="uk-margin-small-right"**>  </**span**> Mis anuncios  </**a**>  </**li**>  </**ul**>  <**div class="uk-navbar-item"**>  <**a href="/publicar"  class="uk-button uk-button-primary uk-visible@m"**>  <**span data-uk-icon="plus-circle"  class="uk-margin-small-right"**>  </**span**>Publicar anuncio  </**a**>  </**div**>  <**div class="uk-navbar-item"**>  <**a href="/login"  class="uk-button uk-button-default uk-visible@m"**>  <**span data-uk-icon="sign-in"  class="uk-margin-small-right"**>  </**span**>Acceder  </**a**>  </**div**>  *<!-- Menú de movil lo oculta en pantallas mayores de m-->* <**div class="uk-navbar-item uk-hidden@m"**>  <**a class="uk-navbar-toggle"  data-uk-toggle data-uk-navbar-toggle-icon  href="#offcanvas-nav"**>  </**a**>  </**div**>  </**div**>  </**nav**>  </**div**> </**header**>  </**body**> </**html**> |

* **uk-visible@m :** visible en pantallas de tamaño m o más.
* **uk-hidden@m :** oculto en pantallas de tamaño m o más.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | CLASS | DESCRIPTION | | --- | --- | | uk-hidden@s | Only affects device widths of *640px* and larger. | | uk-hidden@m | Only affects device widths of *960px* and larger. | | uk-hidden@l | Only affects device widths of *1200px* and larger. | | uk-hidden@xl | Only affects device widths of *1600px* and larger. | |

<https://getuikit.com/docs/visibility>

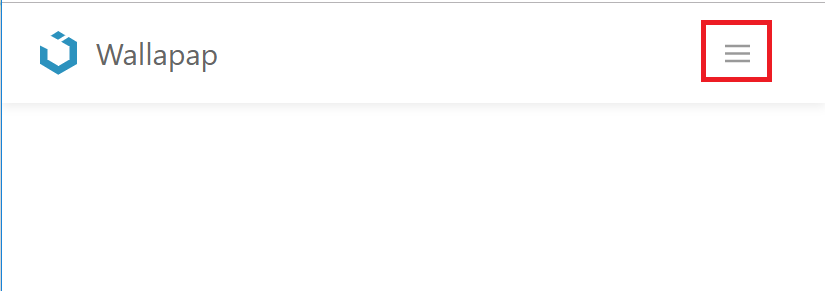




Definimos el elemento **offcanvas-nav,** es una barra de navegación que aparece fuera de la pantalla, se visualiza al pulsar el enlace (especialmente diseñada para móvil)**.**

|  |
| --- |
| </**header**>  *<!-- Menú para movil - offCanvas -->* <**div id="offcanvas-nav" data-uk-offcanvas="flip: true; overlay: true"**>  <**div class="uk-offcanvas-bar uk-offcanvas-bar-animation uk-offcanvas-slide"**>  <**button  class="uk-offcanvas-close uk-close uk-icon"  type="button"  data-uk-close**>  </**button**>  <**ul class="uk-nav uk-nav-default"**>  <**li class="uk-nav-header"**>Inicio</**li**>  <**li**>  <**a href="/seguidos"**>  <**span class="uk-margin-small-right uk-icon"  data-uk-icon="icon: heart"**>  </**span**>Seguidos  </**a**>  </**li**>  <**li**>  <**a href="/misanuncios"**>  <**span class="uk-margin-small-right uk-icon"  data-uk-icon="icon: list"**>  </**span**>Mis anuncios  </**a**>  </**li**>  <**li**>  <**a href="/publicar"**>  <**span class="uk-margin-small-right uk-icon"  data-uk-icon="icon: plus-circle"**>  </**span**>Publicar anuncio  </**a**>  </**li**>  <**li**>  <**a href="/login"**>  <**span class="uk-margin-small-right uk-icon"  data-uk-icon="icon: sign-in"**>  </**span**>Acceder  </**a**>  </**li**>  <**li class="uk-nav-header"**>Categorías</**li**>  <**li**><**a href="/anuncios"**>Coches</**a**></**li**>  <**li**><**a href="/anuncios"**>Motos</**a**></**li**>  <**li**><**a href="/anuncios"**>Bicicletas</**a**></**li**>  <**li**><**a href="/anuncios"**>Informática</**a**></**li**>  <**li**><**a href="/anuncios"**>Teléfonos</**a**></**li**>  <**li**><**a href="/anuncios"**>Tv, audio y foto</**a**></**li**>  <**li**><**a href="/anuncios"**>Electrodomésticos</**a**></**li**>  <**li**><**a href="/anuncios"**>Moda</**a**></**li**>  <**li**><**a href="/anuncios"**>Coleccionismo</**a**></**li**>  <**li**><**a href="/anuncios"**>Construcción</**a**></**li**>  <**li**><**a href="/anuncios"**>Empleo</**a**></**li**>  <**li**><**a href="/anuncios"**>Servicios</**a**></**li**>  <**li**><**a href="/anuncios"**>Otros</**a**></**li**>  </**ul**>  </**div**> </**div**>  </**body**> </**html**> |

Probamos que funciona.

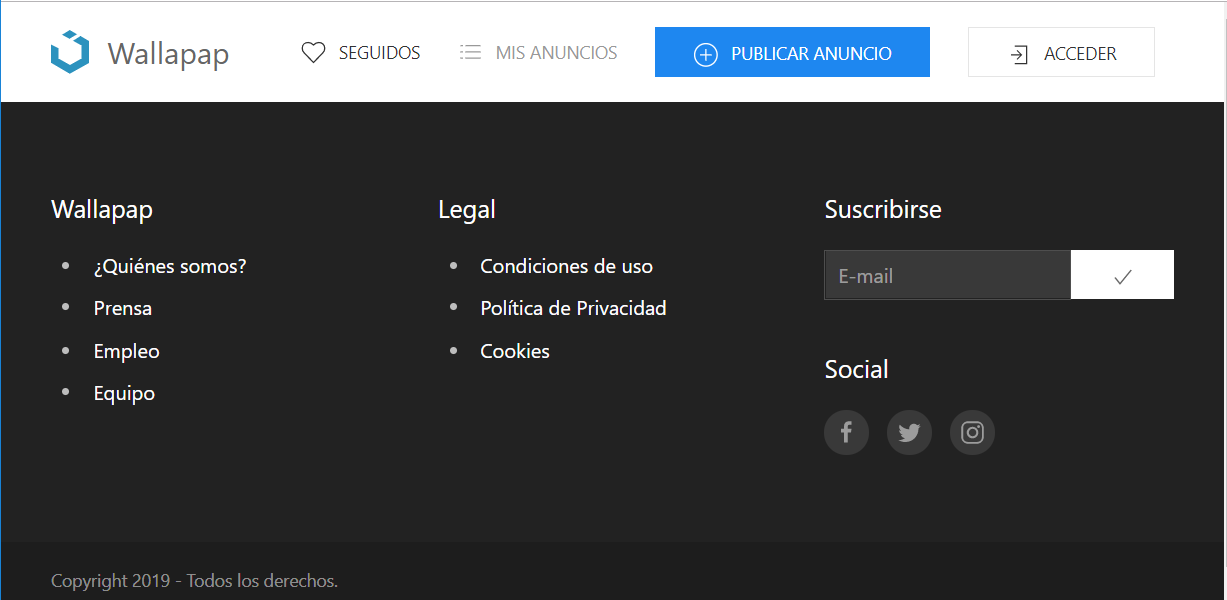




Footer (no tendrá funcionalidad real), usa el sistema de grid, lo explicaremos más adelante.

|  |
| --- |
| </**div**>  *<!--FOOTER-->* <**footer class="uk-section-secondary"**>  <**div class="uk-container uk-section"**>  <**div class="uk-grid uk-child-width-1-3@m"**>  <**div**>  <**h4 class="uk-margin-top"**>Wallapap</**h4**>  <**ul class="uk-list uk-list-bullet"**>  <**li**><**a href="#"**>¿Quiénes somos?</**a**></**li**>  <**li**><**a href="#"**>Prensa</**a**></**li**>  <**li**><**a href="#"**>Empleo</**a**></**li**>  <**li**><**a href="#"**>Equipo</**a**></**li**>  </**ul**>  </**div**>   <**div**>  <**h4 class="uk-margin-top"**>Legal</**h4**>  <**ul class="uk-list uk-list-bullet"**>  <**li**><**a href="#"**>Condiciones de uso</**a**></**li**>  <**li**><**a href="#"**>Política de Privacidad</**a**></**li**>  <**li**><**a href="#"**>Cookies</**a**></**li**>  </**ul**>  </**div**>  <**div**>  <**h4 class="uk-margin-top"**>Suscribirse</**h4**>  <**form class="uk-grid uk-grid-collapse"** >  <**div class="uk-width-expand"**>  <**input class="uk-input" type="email"  placeholder="E-mail"**>  </**div**>  <**div class="uk-width-auto"**>  <**button class="uk-button uk-button-primary"  data-uk-icon="check"**>  </**button**>  </**div**>  </**form**>  <**h4**>Social</**h4**>  <**a class="uk-icon-button uk-margin-small-right"  data-uk-icon="facebook"**></**a**>  <**a class="uk-icon-button uk-margin-small-right"  data-uk-icon="twitter"**></**a**>  <**a class="uk-icon-button"  data-uk-icon="instagram"**></**a**>  </**div**>  </**div**>  </**div**>  <**div class="uk-section uk-section-xsmall"  style="background-color**: **rgba**(0,0,0,0.15)**"**>  <**div class="uk-container"**>  <**div class="uk-grid uk-child-width-1-2@s uk-text-center uk-text-left@s"  data-uk-grid**>  <**div class="uk-text-small uk-text-muted"**>  Copyright 2019 - Todos los derechos.  </**div**>  </**div**>  </**div**>  </**div**> </**footer**>  </**body**> </**html**> |

El resultado final debería ser el siguiente.



Antes de footer introducimos la etiqueta content.

|  |
| --- |
| </**div**> </**div**>  {{{content}}}  *<!--FOOTER-->* <**footer class="uk-section-secondary"**>  <**div class="uk-container uk-section"**> |

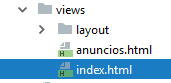
Definimos en **main.js** el uso de los ficheros que hay en la carpeta **/views/layout** como bases para crear otras vistas (definen las partes comunes)

|  |
| --- |
| **await** server.views({  **engines**: {  **html**: require(**'handlebars'**)  },  **relativeTo**: \_\_dirname,  **path**: **'./views'**,  **layoutPath**: **'./views/layout'**,  **context** : {  **sitioWeb**: **"wallapep"** } }); |

Abrimos **routes.js** y localizamos el **GET /** , retornará la vista **index** (que aún está sin crear), combinada con el layout **base**.

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/'**,  handler: **async** (req, h) => {  **return** h.view(**'index'**,  { **usuario**: **'jordán'**},  { **layout**: **'base'**});  } } |

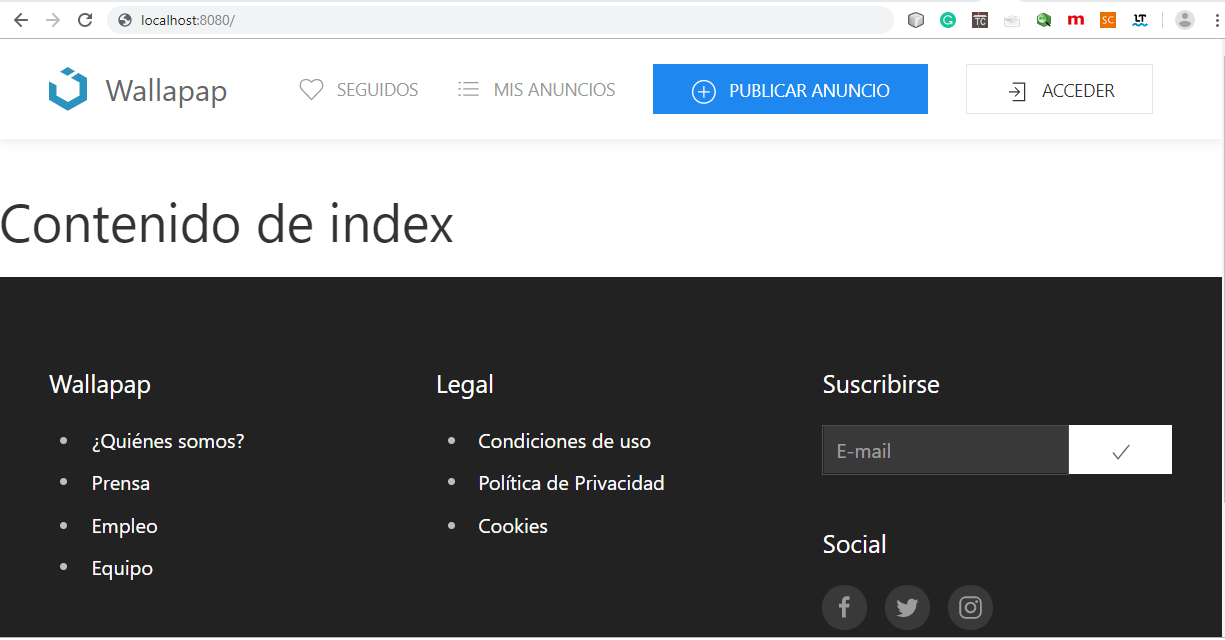
Creamos **index.html** , dentro del directorio **/views/**



Empezamos incluyendo un contenido muy simple.

|  |
| --- |
| <**h1**> Contenido de index </**h1**> |

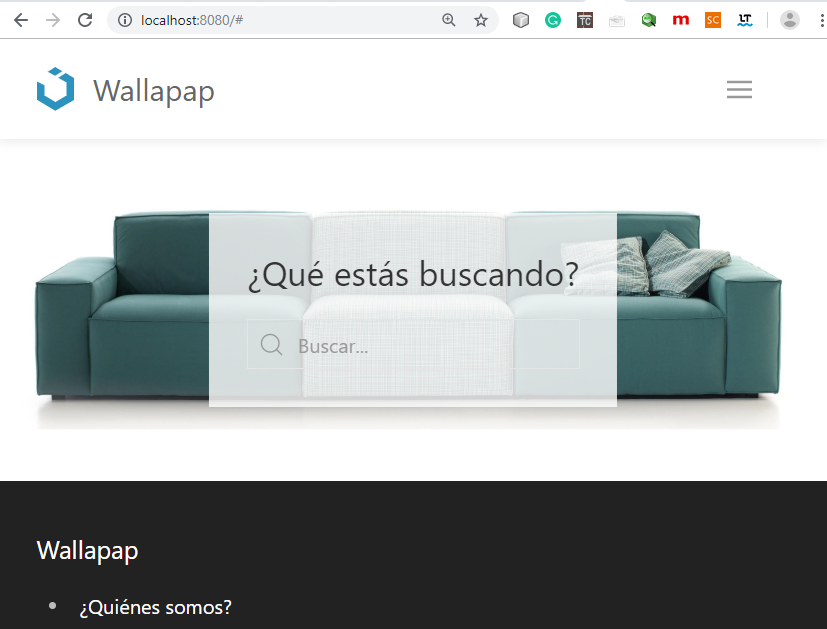
Probamos a acceder a <http://localhost:8080/>



Continuamos editando **index.html**

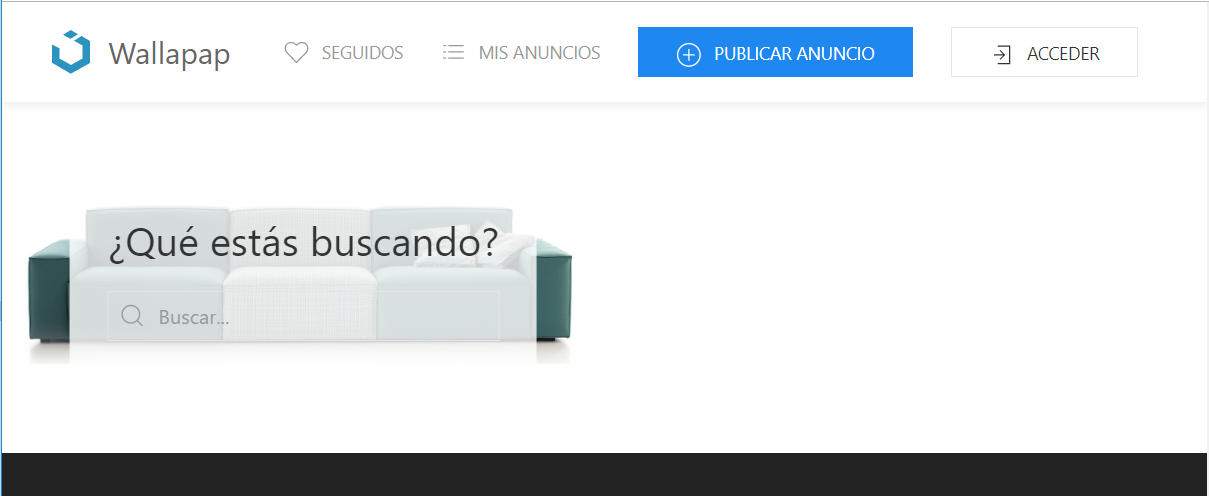
Incluimos una sección con un cuadro de busqueda sobre un fondo.

|  |
| --- |
| <**section class="uk-section"**>  *<!-- necesitamos un uk-inline para contener el div de abajo -->* <**div class="uk-inline uk-width-1-1 "**>  <**img src="images/sofa-high.jpg" alt=""**>  <**div class="uk-position-center uk-overlay uk-overlay-default"**>  <**h2 class="uk-margin"**>¿Qué estás buscando?</**h2**>  <**div class="uk-margin"**>  <**form class="uk-search uk-search-default uk-width-1-1"  action="/anuncios"**>  <**span uk-search-icon**></**span**>  <**input class="uk-search-input"  name="criterio"  type="search" placeholder="Buscar..."**>  </**form**>  </**div**>  </**div**>  </**div**> </**section**> |



\*Prueba cómo funciona el grid: Cambiar la clase **uk-width-1-1** por **uk-width-1-2** y observar el resultado.

|  |
| --- |
| <**section class="uk-section"**>  *<!-- necesitamos un uk-inline para contener el div de abajo -->* <**div class="uk-inline uk-width-1-2 "**>  <**img class="uk-width-1-1" src="images/sofa-high.jpg" alt=""**> |



Se pueden configurar diferentes tamaños para diferentes tamaños de pantalla, por ejemplo:  
 **uk-width-1-1@s , uk-width-1-2@l**

**uk-overlay** permite superponer capas, **uk-position-center** coloca la capa superpuesta en la posición central (es importante que el div padre tenga la clase **uk-inline** ) . ¿Qué sucede si quitamos las clases overlay?

|  |
| --- |
| <**section class="uk-section"**>  *<!-- necesitamos un uk-inline para contener el div de abajo -->* <**div class="uk-inline uk-width-1-1 "**>  <**img src="images/sofa-high.jpg" alt=""**>  <**div ~~class="uk-position-center uk-overlay uk-overlay-default"~~**>  <**h2 class="uk-margin"**>¿Qué estás buscando?</**h2**> |

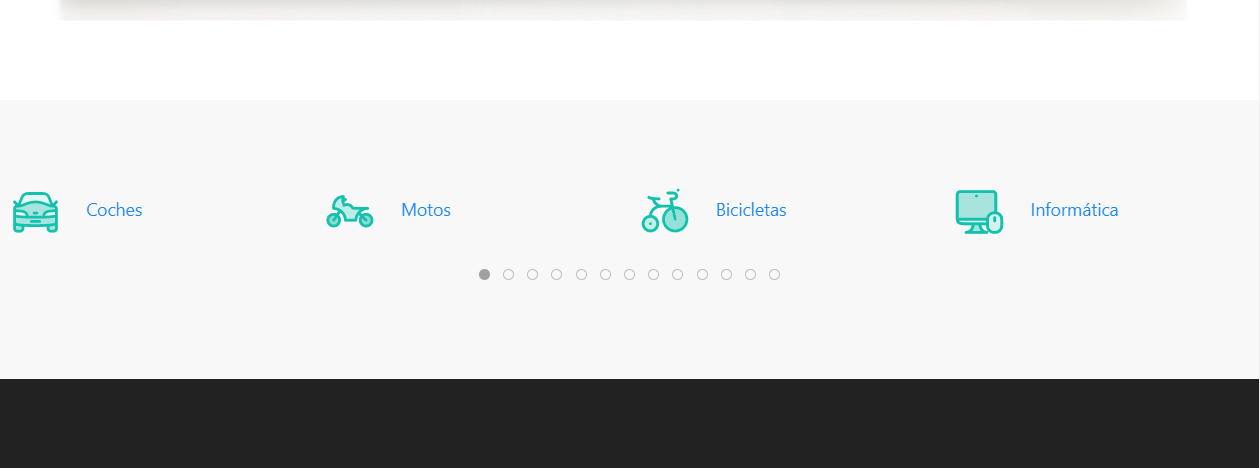
Volvemos a dejarlo todo como estaba antes de los cambios, a continuación de la barra de búsqueda **agregamos un slider** con enlaces a las diferentes categorías de anuncios. <https://getuikit.com/docs/slider>

Se definirán diferentes tamaños dependiendo de la pantalla.

En la parte final de la definición del slider se encuentran las flechas de dirección y los botones de acceso a las hojas. Copiamos la siguiente sección a continuación en el index.html

|  |
| --- |
| *<!-- Visible en m o más -->* <**section class="uk-section uk-section-muted uk-visible@m"**>  <**div uk-slider**>  <**div class="uk-position-relative uk-visible-toggle"** >   <**ul class="uk-slider-items   uk-child-width-1-4@m  uk-child-width-1-6@l"**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/coches.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Coches</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/moto.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Motos</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/bicicleta.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Bicicletas</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/informatica.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Informática</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/telefonos.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Teléfonos</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/tv-audio-foto.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Tv, audio y foto</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/electrodomesticos.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Electrodomésticos</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/moda.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Moda</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/coleccionismo.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Coleccionismo</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/construccion.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Construcción</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/empleo.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Empleo</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/servicios.svg" width="64" height="64"**>  <**span class="uk-margin-small-left"**>Servicios</**span**>  </**a**>  </**li**>  <**li**>  <**a href="/anuncios"**>  <**img src="images/otros.svg" width="64" height="64"** >  <**span class="uk-margin-small-left"**>Otros</**span**>  </**a**>  </**li**>  </**ul**>   <**a class="uk-position-center-left uk-position-small uk-hidden-hover"  href="#"  uk-slidenav-previous  uk-slider-item="previous"**>  </**a**>  <**a class="uk-position-center-right uk-position-small uk-hidden-hover"  href="#"  uk-slidenav-next  uk-slider-item="next"**>  </**a**>  </**div**>   <**ul class="uk-slider-nav uk-dotnav uk-flex-center uk-margin"**></**ul**>   </**div**> </**section**> |

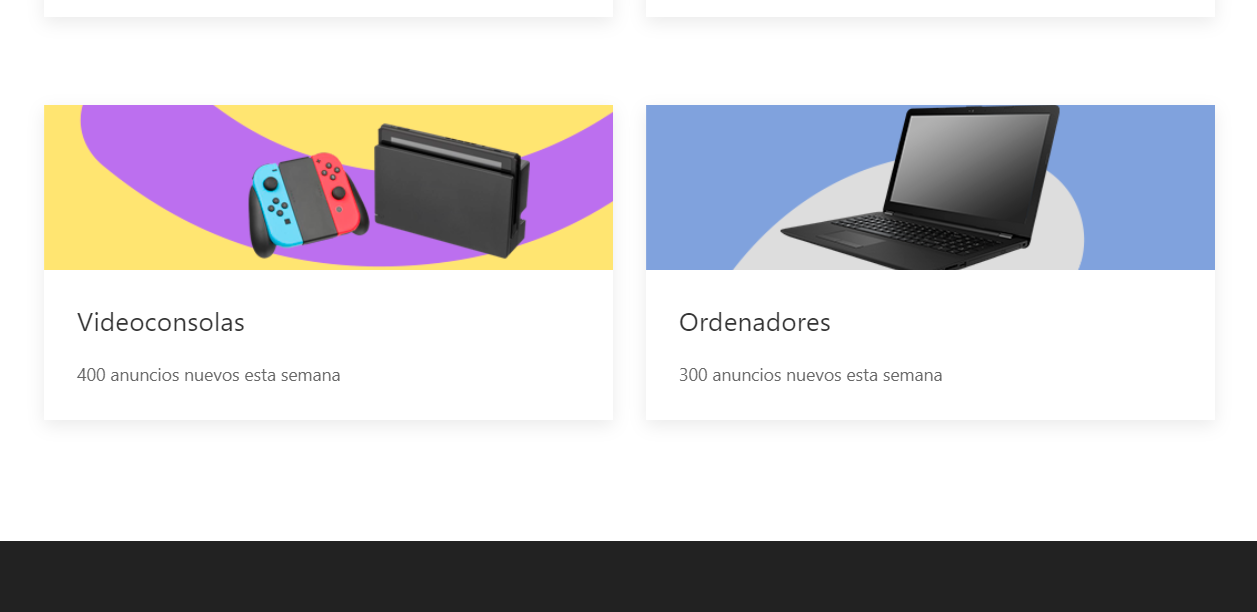
Comprobamos como muestra diferente número de elementos dependiendo de la resolución (también vemos que se oculta si la resolución es menor de m).



Incluimos una nueva sección, donde habrá unas cards **uk-card uk-card-default** las cuales van a incluir unas animaciones, para que aparezcan por la izquierda o la derecha **uk-scrollspy="cls: uk-animation-slide-left; repeat: true"** <https://getuikit.com/docs/card>

|  |
| --- |
| <**section class="uk-section uk-section-default"**>  <**div class="uk-container"**>  *<!-- Todos los hijos ocupan la mitad 1-2 cuando es de m o más -->* <**div class="uk-grid uk-child-width-1-2@m"**>  <**div class="uk-margin-medium"**>  <**div class="uk-card uk-card-default"  uk-scrollspy="cls: uk-animation-slide-left; repeat: true"**>  <**div class="uk-card-media-left uk-cover-container"**>  <**img src="images/destacado1.png" uk-cover**>  <**canvas**></**canvas**>  </**div**>  <**div**>  <**div class="uk-card-body"**>  <**h3 class="uk-card-title"**>Bicicletas</**h3**>  <**p**>400 anuncios nuevos esta semana</**p**>  </**div**>  </**div**>  </**div**>  </**div**>  <**div**>  <**div class="uk-card uk-card-default"  uk-scrollspy="cls: uk-animation-slide-right; repeat: true"**>  <**div class="uk-card-media-left uk-cover-container"**>  <**img src="images/destacado2.png" alt="" uk-cover**>  <**canvas**></**canvas**>  </**div**>  <**div**>  <**div class="uk-card-body"**>  <**h3 class="uk-card-title"**>Muebles</**h3**>  <**p**>300 anuncios nuevos esta semana </**p**>  </**div**>  </**div**>  </**div**>  </**div**>  <**div class="uk-margin-medium"**>  <**div class="uk-card uk-card-default"  uk-scrollspy="cls: uk-animation-slide-left; repeat: true"**>  <**div class="uk-card-media-left uk-cover-container"**>  <**img src="images/destacado12.png" alt="" uk-cover**>  <**canvas**></**canvas**>  </**div**>  <**div**>  <**div class="uk-card-body"**>  <**h3 class="uk-card-title"**>Videoconsolas</**h3**>  <**p**>400 anuncios nuevos esta semana</**p**>  </**div**>  </**div**>  </**div**>  </**div**>  <**div class="uk-margin-medium"**>  <**div class="uk-card uk-card-default"  uk-scrollspy="cls: uk-animation-slide-right; repeat: true"**>  <**div class="uk-card-media-left uk-cover-container"**>  <**img src="images/destacado7.png" alt="" uk-cover**>  <**canvas**></**canvas**>  </**div**>  <**div**>  <**div class="uk-card-body"**>  <**h3 class="uk-card-title"**>Ordenadores</**h3**>  <**p**>300 anuncios nuevos esta semana </**p**>  </**div**>  </**div**>  </**div**>  </**div**>  </**div**> </**section**> |

La sección se verá del siguiente modo:



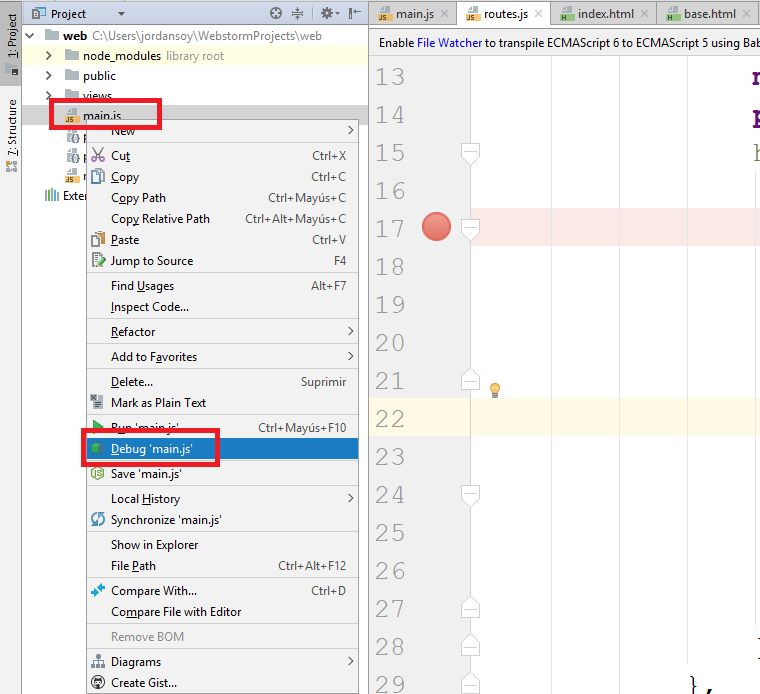
|  |
| --- |
| **Plantillas y URLs absolutas.**  Es muy recomendable utilizar URLs absolutas en las plantillas **/ejemplo/recurso.png** en lugar de **ejemplo/recurso.png** |

Depurar

Seleccionamos cualquier fichero .js y añadimos los puntos de ruptura (en una parte donde realmente se pueda detener la ejecución, no colocar por ejemplo en la cabecera de una función)



Para ejecutar en modo depuración pulsamos botón derecho sobre el fichero principal, **app.js** y la acción **Debug ‘app.js’**

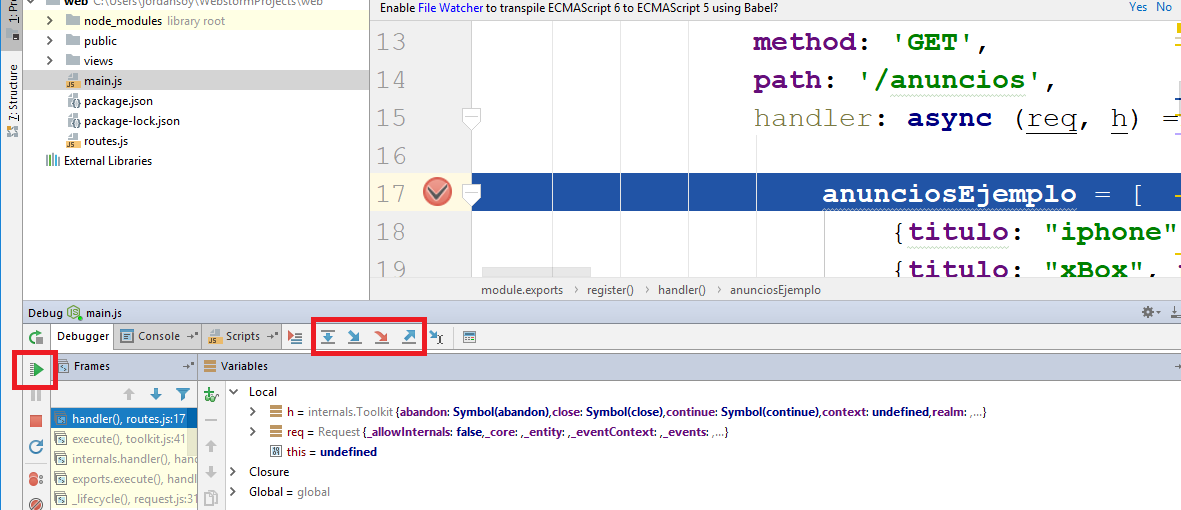


Si tenemos el puerto libre la aplicación se ejecutará el modo de depuración

Al hacer la petición que haga entrar a la petición en un punto de ruptura, se detendrá la ejecución, por ejemplo <http://localhost:8080/anuncios>

Desde la pestaña **Debugger** podemos ver toda la información de depuración.

* Las flechas de la parte central derecha nos permiten hacer avanzar la ejecución.
* La flecha verde de la parte izquierda reanuda la ejecución hasta el siguiente punto de ruptura

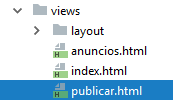


# Controlador y vista para agregar anuncio

La aplicación responderá a la petición **GET /publicar** enviando una vista **publicar** donde se mostrará un formulario que solicitará los datos del anuncio.

|  |
| --- |
| server.**route**([  {  **method**: **'GET'**,  **path**: **'/publicar'**,  handler: **async** (req, h) => {  **return** h.view(**'publicar'**,  { **usuario**: **'jordán'**},  { **layout**: **'base'**});  }  },  { |

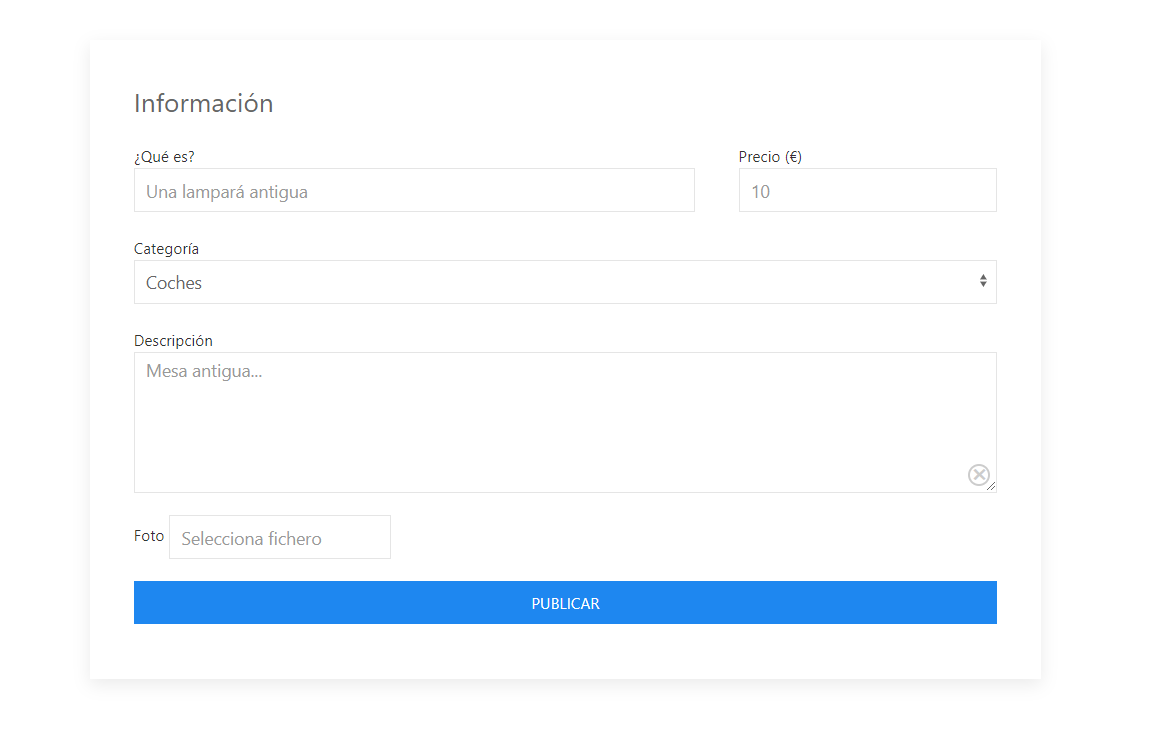
Creamos la vista **publicar.html** dentro de **/views/**



En **publicar.html** definimos un formulario HTML estándar.

|  |
| --- |
| <**section class="uk-section"**>  <**div class="uk-card uk-card-default uk-card-body  uk-width-1-2@m uk-align-center"**>  <**form method="post" action="/publicar"**>  <**fieldset class="uk-fieldset"**>   <**legend class="uk-legend"**>Información </**legend**>   <**div class="uk-grid uk-margin"**>  <**div class="uk-width-2-3"**>  <**label class="uk-form-label"**>  ¿Qué es?  </**label**>  <**input class="uk-input" type="text"  name="titulo"  placeholder="Una lámpara antigua"  required="true"**>  </**div**>  <**div class="uk-width-1-3"**>  <**label class="uk-form-label"**>  Precio (€)  </**label**>  <**input class="uk-input" type="number"  name="precio" placeholder="10"  required="true"**>  </**div**>  </**div**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Categoría</**label**>  <**select class="uk-select" name="categoria"   required="true"**>  <**option value="coches"**>Coches</**option**>  <**option value="motos"**>Motos</**option**>  <**option value="bicicletas"**>Bicicletas</**option**>  <**option value="informatica"**>Informática</**option**>  <**option value="telefonos"**>Teléfonos</**option**>  <**option value="moda"**>Moda</**option**>  <**option value="otros"**>Otros</**option**>  </**select**>  </**div**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Descripción</**label**>  <**textarea class="uk-textarea" rows="5"  name="descripcion"  placeholder="perfecto estado,180 de alto..."   required="true"**></**textarea**>  </**div**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Foto</**label**>  <**div uk-form-custom="target: true"**>  <**input type="file" name="foto" accept="image/\*"**>  <**input class="uk-input" type="text"  placeholder="Selecciona fichero"**>  </**div**>  </**div**>   <**button class="uk-button uk-button-primary uk-width-1-1  uk-margin-small-bottom"  type="submit"**>  Publicar  </**button**>  </**fieldset**>  </**form**>  </**div**> </**section**> |

Sí ejecutamos la aplicación y entramos en <http://localhost:8080/publicar> podremos ver el formulario.



# Peticiones POST y parámetros

A diferencia de las peticiones GET, las POST tienen un cuerpo (body) que se puede contener datos (pares de clave-valor, texto plano, json, binario, o cualquier otro tipo de datos). Este tipo de peticiones se utilizan comúnmente en formularios.

El fichero localizado en la ruta GET <http://localhost:8080/publicar> define un formulario que envía una petición **POST /publicar** .

El formulario contiene varios inputs con atributo **name** (toma los valores: nombre, categoría, descripción, foto).  
  
Podemos acceder a estos parámetros con **req.payload.<nombre\_del\_parámetro>.**

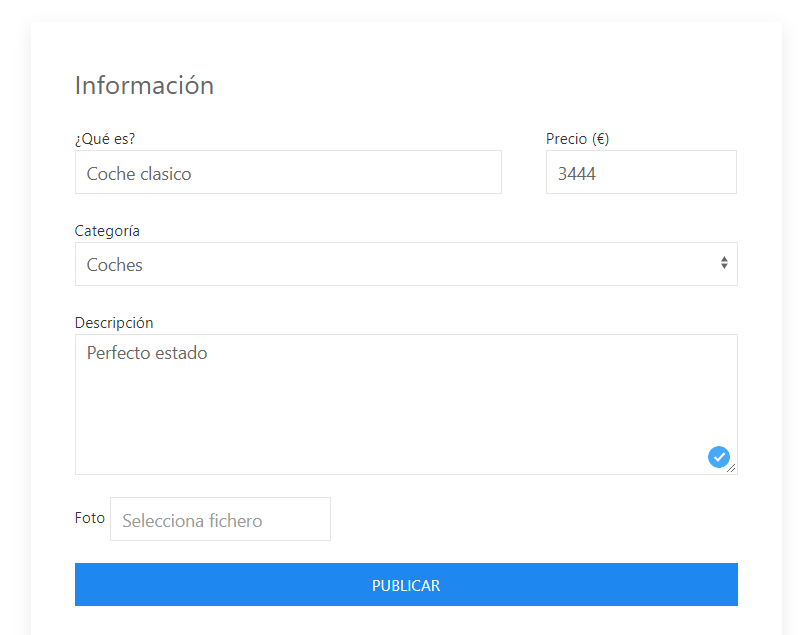
|  |
| --- |
| server.**route**([  {  **method**: **'POST'**,  **path**: **'/publicar'**,  handler: **async** (req, h) => {   **anuncio** = {  **usuario** : **"sin usuario"**,  **titulo**: req.**payload**.**titulo**,  **descripcion**: req.**payload**.**descripcion**,  **categoria**: req.**payload**.**categoria**,  **precio**: ***Number***.parseFloat(req.**payload**.**precio**),   }  **return** h.view(**'publicar'**,  { **mensaje**: **'Articulo agregado'**},  { **layout**: **'base'**});  }  },  { |

Modificamos la **base.html** para que cualquier vista la variable **mensaje**, si es que es distinto de null.

|  |
| --- |
| *<!-- NOTIFICACION -->* {{#if mensaje}} <**script**>  UIkit.notification({**message**: **'{{ mensaje }}'**, **status**: **'success'**}) </**script**> {{/if}}  *<!--HEADER-->* <**header class="uk-box-shadow-small"**>  <**div class="uk-container uk-container-expand"**>  <**nav class="uk-navbar" data-uk-navbar**>  <**div class="uk-navbar-left"**> |

La notificación la podemos colocar en cualquier parte, la función **UIKit.notification**, agrega la notificación a través del DOM, no hay que colocarla realmente en el HTML. Podemos utilizar diferentes tipos de notificaciones. <https://getuikit.com/docs/notification> <https://getuikit.com/v2/docs/notify.html> (es de la versión anterior, pero usa la misma API).

Guardamos los cambios, ejecutamos la aplicación, utilizamos el formulario <http://localhost:8081/agregar.html> para comprobar que la petición POST está funcionando correctamente **no agregar los ficheros**.





# Mongo DB en la Nube (**Obligatorio despliegue en la nube**)

MongoDB es una base de datos no relacional, orientada a **documentos**, la información en los documentos almacena siguiendo una estructura JSON, internamente MongoDB maneja BSON, se trata de una versión más ligera en formato binario creada a partir del formato JSON [https://www.mongodb.com/json-and-bson](https://www.mongodb.com/json-and-bson%20%20).

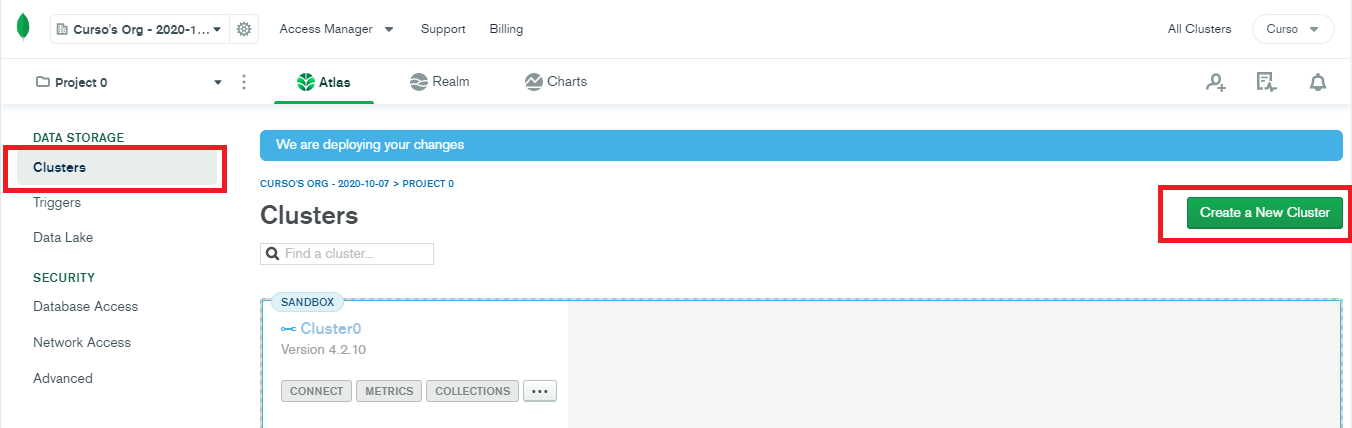
Ejemplo de documento proyecto en MongoDB. Ejemplo de documento en MongoDB:

|  |
| --- |
| **{**  nombre**:** "Cambiar ordenadores"**,**  descripcion**:** "Cambiar todos los ordenadores del piso 1"  **}** |

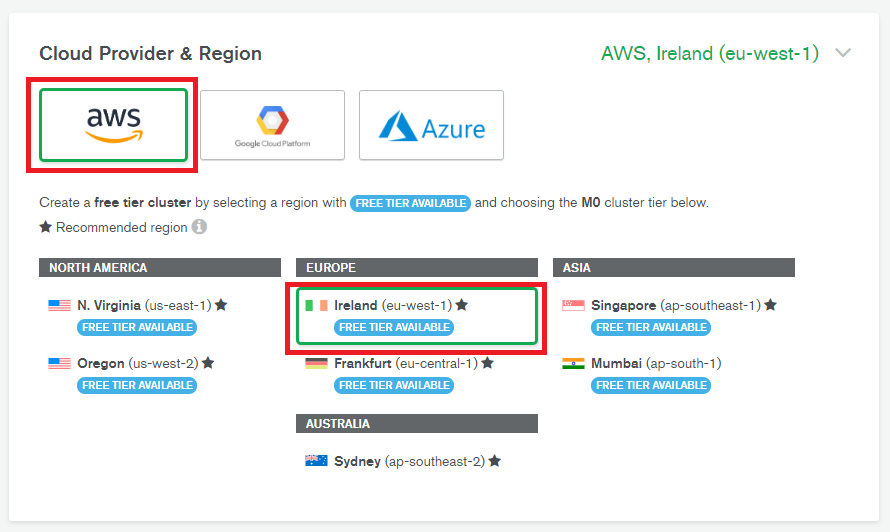
El proveedor https://cloud.mongodb.com/ nos permite crear una instancia de mongoDB en la nube utilizando una cuenta gratuita, esta versión de la cuenta está limitada a 500mb.

Completamos el proceso de registro y nos identificamos en la plataforma.

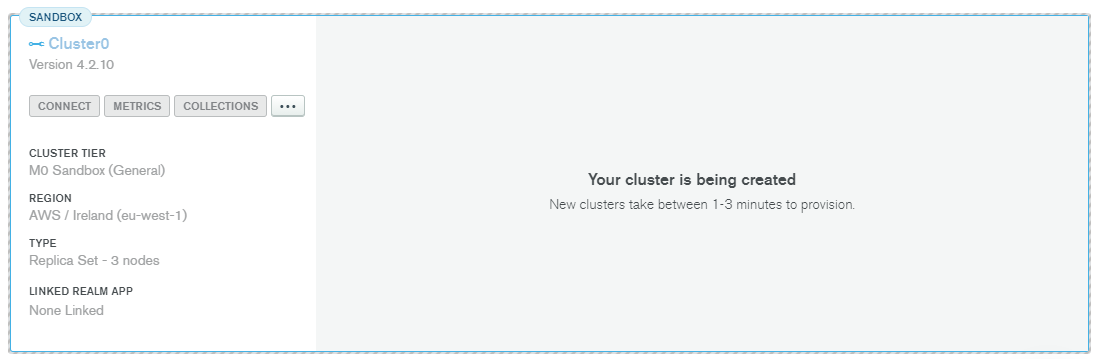
Creamos un nuevo **Cluster (Sí justo acabos de crear la cuenta es posible que nos cree un nuevo cluster automáticamente).**



Seleccionamos **AWS** como nuestro proveedor, en la región de Irlanda .

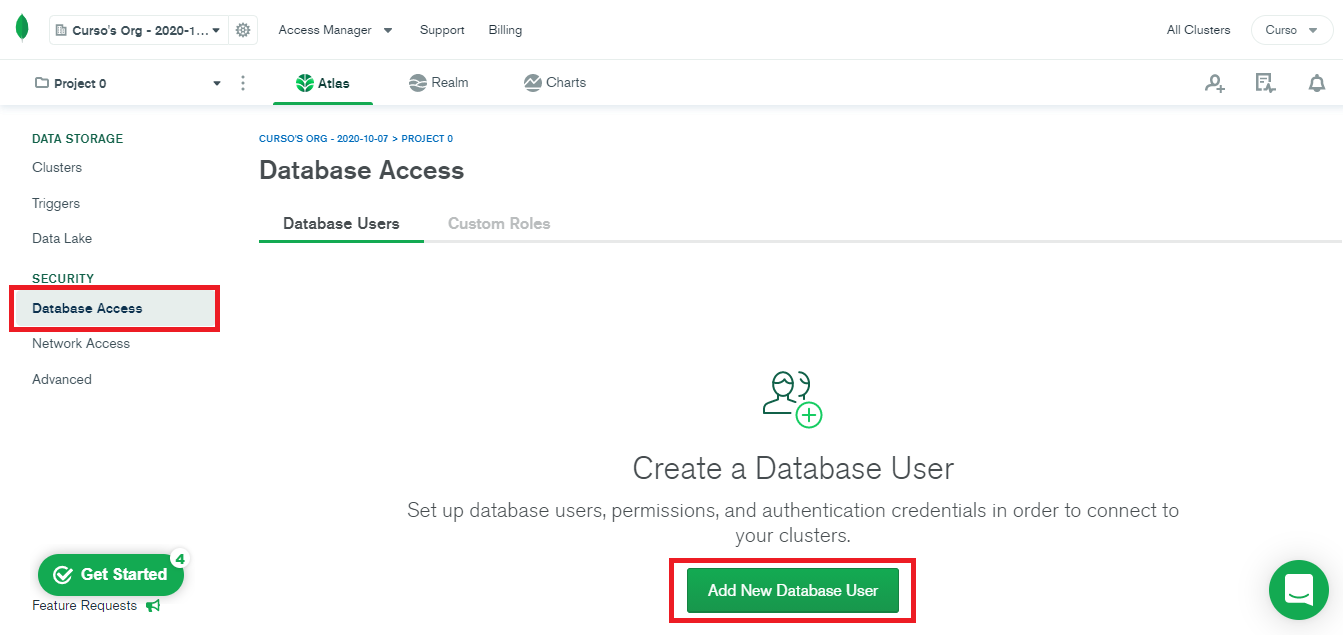


Pulsamos en el botón de la parte inferior **Create a Cluster**, tardará unos 5 minutos en crearlo.

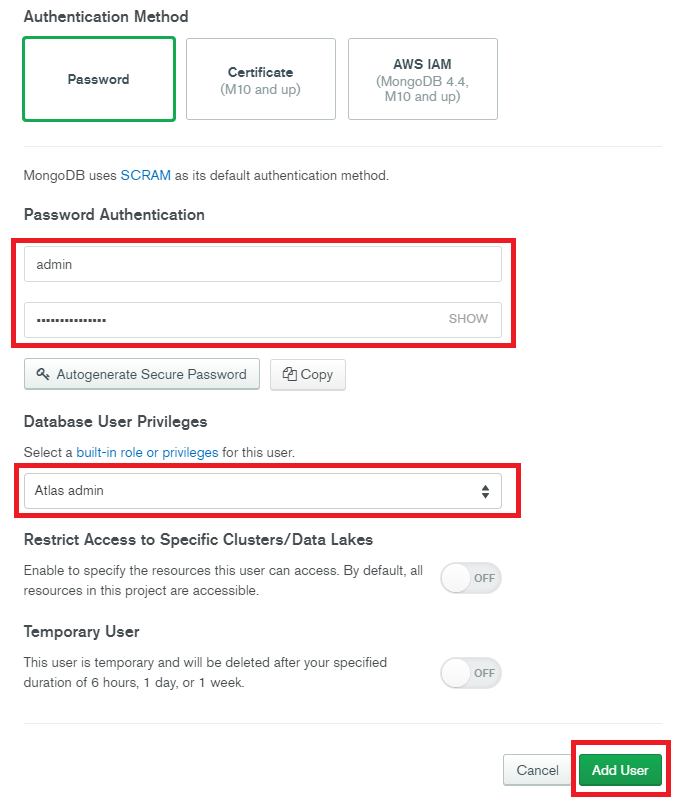


Accedemos a la opción **Database Access**

Dentro de seguridad pulsamos en **Add New User**



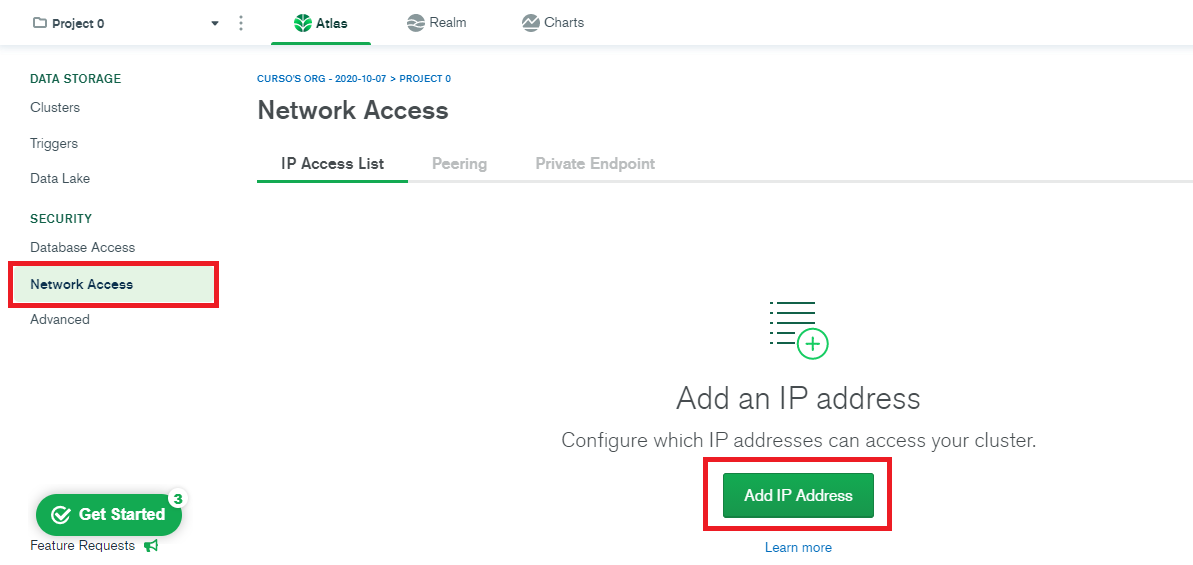
Creamos un nuevo usuario con privilegios **Atlas admin**, nombre **admin** y password **informatica1234** (recomendado otro password más complejo).



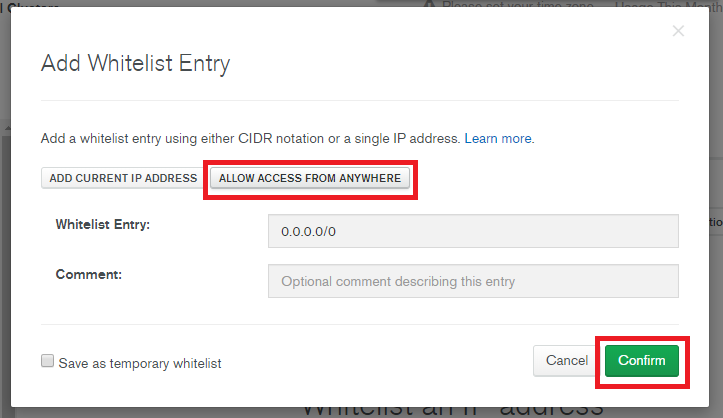
Normalmente deberíamos configurar que IPs pueden conectarse a la base de datos (por seguridad solo debería tener acceso la IP del ordenador que ejecutara la aplicación).

Pulsamos en la opción de menú **Network Access**

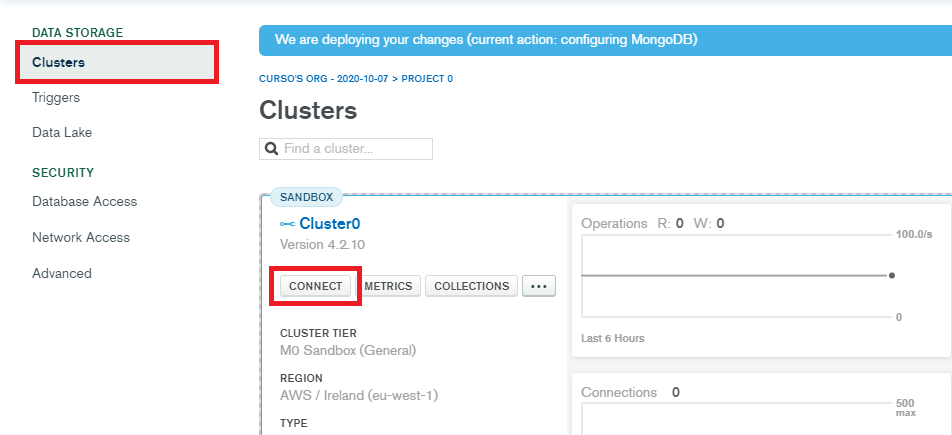
Después pulsamos en **Add Ip Address**



Como en nuestro caso vamos a acceder desde diferentes sitios (casa, universidad) y no queremos perder tiempo configurando la IP, vamos a permitir conexiones desde todas, esto es altamente inseguro para un entorno en producción.



Ya estamos listos para probar la conexión, accedemos a la sección **Clusters**, y después pulsamos el botón **Connect.**

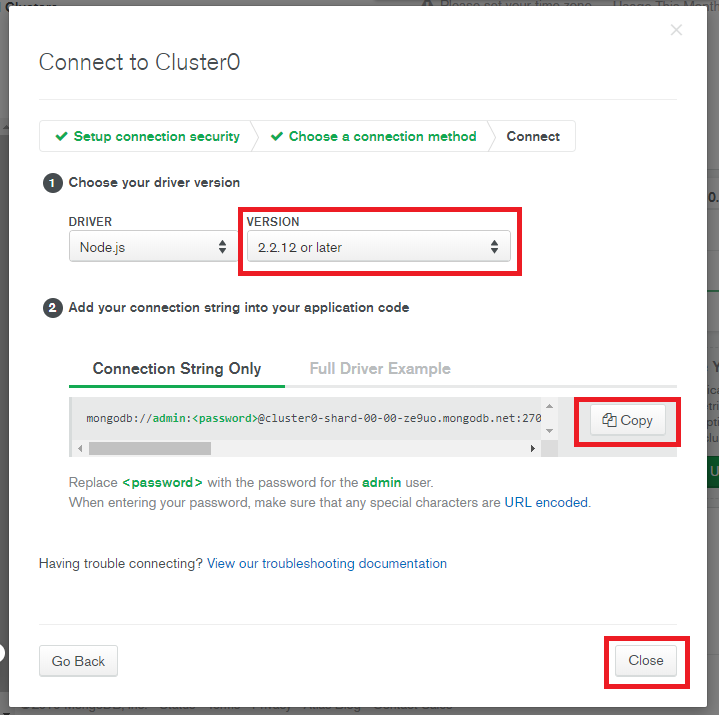


Nos va a generar una cadena de conexión segura dependiendo del tipo de sistema que vayamos a utilizar para acceder a la base de datos.

Indicamos que nos vamos a conectar desde una Aplicación, **Connect Your Application**.



El driver que estamos usando es Node.js , **la versión 2.**



Copiamos la URL de conexión y la completamos con los datos de nuestro usuario.

mongodb://admin:**<password>**@cluster0-shard-00-00.n0jhi.mongodb.net:27017,cluster0-shard-00-01.n0jhi.mongodb.net:27017,cluster0-shard-00-02.n0jhi.mongodb.net:27017/**<dbname>**?ssl=true&replicaSet=atlas-j04uo3-shard-0&authSource=admin&retryWrites=true&w=majority

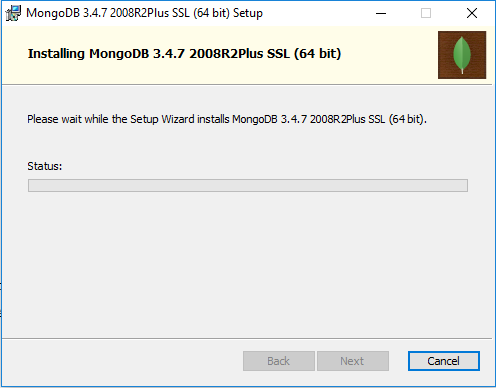
No uséis mi cadena de conexión, cada uno la suya, acordándose de incluir el password.

El dbname puede ser el que queramos.

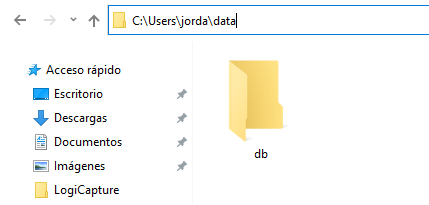
# Instalación y verificación de mongo DB local **(alternativa no hacer)**

En este apartado se instalará un servidor de base de datos MongoDB en local, que luego podrá ser utilizada para persistir datos en aplicaciones.

Para instalar Mongo descargamos el **Community Server** para Windows de la web oficial <https://www.mongodb.com/download-center> y seguimos los pasos para la instalación completa.



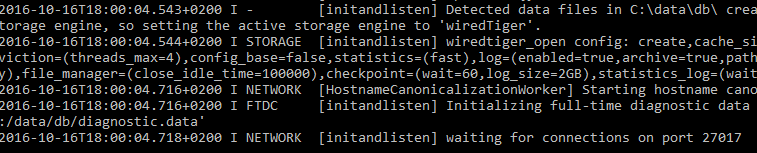
Creamos una estructura de carpetas **/data/db** en un directorio en el que tengamos permisos.



En este caso: **C:\Users\jorda\data\db** pero podría ser cualquier otro.

Desde la línea de comandos CMD accedemos a la ruta donde hayamos instalado MongoDB (por defecto: " C:\Program Files\MongoDB\Server\3.4\bin " ). Para arrancar el servidor ejecutamos el comando mongod --dbpath C:\Users\jordansoy\data\db importante el directorio debe coincidir con el de nuestra carpeta DB

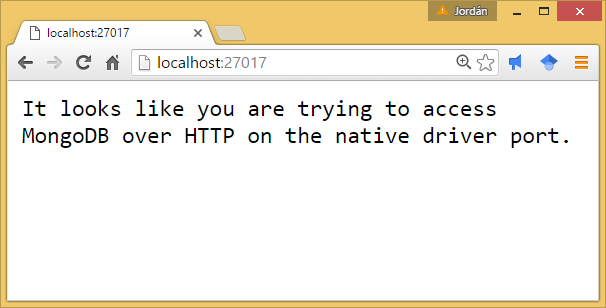
Cuando el servidor arranca de forma correcta debería mostrar el siguiente mensaje: **waiting connections on port 27017**.



|  |  |
| --- | --- |
| Sí aparece algún problema al iniciar la base de datos debemos probar a utilizar el siguiente comando:  mongod --storageEngine=mmapv1 --dbpath C:\Users\jordansoy\data\db  Puede ser buena idea crear un script **iniciarBD.bat** para iniciar la base de datos, para las rutas anteriormente citadas el contenido sería el siguiente:   |  | | --- | | *cd C:\Program Files\MongoDB\Server\3.4\bin mongod --dbpath C:\Users\jorda\data\db* | |

Para detener el servidor de bases de datos mongo debemos pulsar **Control + C.**

Para comprobar que el servidor listo y esperando por conexiones accedemos a [http://localhost:27017](http://localhost:27017%20) desde el propio navegador.



# Acceso a datos de Mongo desde Node.js

Para conectarnos a la base de datos desde la aplicación utilizaremos el módulo **mongodb** <https://docs.mongodb.com/ecosystem/drivers/node-js/>. Se trata de un módulo externo que nos permite conectarnos fácilmente a mongo, pero no es la única alternativa disponible en Node.

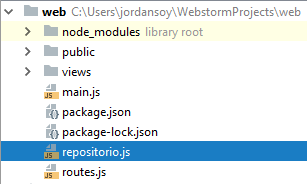
Abrimos la consola de comandos y nos situamos en el directorio raíz del proyecto.

Instalamos el módulo.

**npm install mongodb@2.2.33**

Es importante instalar la versión 2.\* ya que las versiones 2.\* y 3.\* se mantienen en paralelo y utilizan un conjunto de funciones totalmente diferentes.

Vamos a crear un nuevo fichero en el directorio raíz **repositorio.js** .



En este fichero vamos a crear un módulo.

¿Cómo funciona la conexión a mongo?

La función **connect()** del objeto **MongoClient** nos sirve para establecerconexiones, esta función recibe como parámetros:

1. La cadena de conexión, la cual tenemos almacenada en la variable de aplicación ‘db’.
2. La función que se ejecutará al completar la conexión. Esta función tiene dos parámetros:
   * **err** : es un objeto donde se almacenan los errores, es nulo/vacío si no ha habido ningún problema.
   * **db** : es una referencia a la base de datos, sobre este objeto se pueden ejecutar las acciones (insertar, consultar, etc.).

En la función que enviamos como parámetro utilizaremos la conexión a la base de datos para insertar el objeto **cancion**. Recuperamos la colección canciones con **db.collection(canciones)**, sobre esta colección ejecutamos una inserción, con la función **insert**.

La función **insert** recibe como parámetros:

1. El propio objeto a insertar, **anuncio**
2. Una función que se ejecuta al finalizar la operación insert, desde esta función debemos ejecutar la función **db.close()** para cerrar la conexión a la base de datos. Está función recibe otros dos parámetros
   * **err** : es un objeto donde se almacenan los errores, es nulo/vacío si no ha habido ningún problema.
   * **result** : es un array con los objetos que ha sido insertado en la base de datos. Insert agrega a los documentos una propiedad automática “\_id” . Sí insertamos un solo documento y queremos consultar su \_id debemos acceder a **result.ops[0].\_id** .

Vamos a crear dos funciones

* **Conexión** -> retorna la conexión a la base de datos, importante introducir la cadena de conexión correctamente.
* **insertarAnuncio** -> recibe la conexión y el anuncio e inserta el anuncio en la colección anuncios.

|  |
| --- |
| module.**exports** = {  conexion : **async** () => {  **var** mongo = require(**"mongodb"**);  **var** db = **"mongodb://admin:informati…"**;  **promise** = **new** *Promise*((resolve, reject) => {  mongo.**MongoClient**.connect(db, (err, db) => {  **if** (err) {  resolve(**null**)  } **else** {  resolve(db);  }  });  });  **return promise**;  },  insertarAnuncio : **async** (db, anuncio) => {  **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'anuncios'**);  collection.insert(anuncio, (err, result) => {  **if** (err) {  resolve(**null**);  } **else** {  *// \_id no es un string es un ObjectID* resolve(result.**ops**[0].**\_id**.toString());  }  db.close();  });  });   **return promise**;  } } |

Vamos a **main.js** y declaramos el repositorio, lo registramos el **server.method** para que pueda ser accedido desde cualquier parte de la aplicación a través del método **getRepositorio**.

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**); **const** Inert = require(**'@hapi/inert'**); **const** Vision = require(**'@hapi/vision'**); **const routes** = require(**"./routes.js"**); **const repositorio** = require(**"./repositorio.js"**);  *// Servidor* **const** server = Hapi.server({  **port**: 8080,  **host**: **'localhost'**, });  *// declarar metodos comunes* server.method({  **name**: **'getRepositorio'**,  method: () => {  **return repositorio**;  },  **options**: {} });  **const** *iniciarServer* = **async** () => { |

Ahora en el **routes.js** accedemos a la función **register** (se ejecuta automáticamente al registrar el módulo) , creamos una variable repositorio que será utilizada desde todos los controladores.

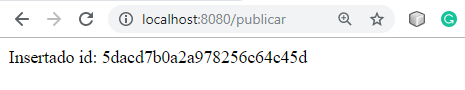
|  |
| --- |
| module.**exports** = {  **name**: **'MiRouter'**,  register: **async** (server, options) => {  **repositorio** = server.**methods**.getRepositorio();   server.**route**([  { |

Modificamos la lógica del **POST /publicar**,

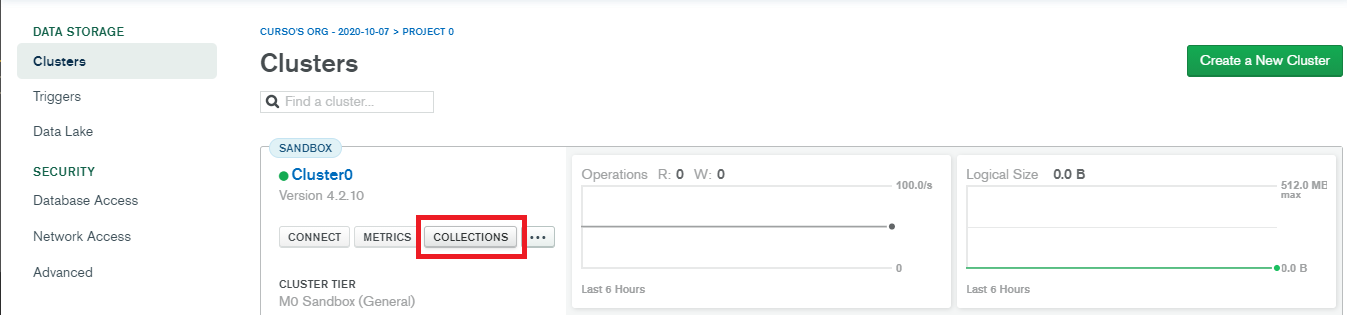
|  |
| --- |
| server.**route**([  {  **method**: **'POST'**,  **path**: **'/publicar'**,  handler: **async** (req, h) => {   **anuncio** = {  **usuario**: **"sin usuario"**,  **titulo**: req.**payload**.**titulo**,  **descripcion**: req.**payload**.**descripcion**,  **categoria**: req.**payload**.**categoria**,  **precio**: ***Number***.parseFloat(req.**payload**.**precio**),   }   *// await no continuar hasta acabar esto  // Da valor a respuesta* **await repositorio**.conexion()  .then((db) => **repositorio**.insertarAnuncio(db, **anuncio**))  .then((id) => {  **respuesta** = **""**;  **if** (id == **null**) {  **respuesta** = **"Error al insertar"** } **else** {  **respuesta** = **"Insertado id: "**+ id;  }  })   **return respuesta**;  }  }, |

|  |  |
| --- | --- |
| La primera promesa llama a conexión() para obtener la variable db, referencia a la conexión de la base de datos  A continuación la segunda promesa **insertarAnuncio** envía como parámetro la referencia a la conexión y el anuncio a insertar, obtiene la referencia id del anuncio insertado   |  | | --- | | **await repositorio**.conexion()  .then((db) => **repositorio**.insertarAnuncio(db, **anuncio**))  .then((id) => {  **respuesta** = **""**;  **if** (id == **null**) {  **respuesta** = **"Error al insertar"** } **else** {  **respuesta** = **"Insertado id: "**+ id;  }  }) | |

Una vez agregada la respuesta será del tipo:

****

Desde la propia web podemos ver los elementos insertados en la colección.

****