**Parte 2**



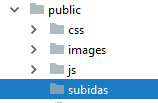
# 

Subida de ficheros

Lo primero que vamos a hacer es una pequeña modificación en la vista **publicar.html** para que solo acepte imágenes de tipo png (en lugar de cualquier tipo de imagen).

|  |
| --- |
| <**input type="file" name="foto" accept="image/png"**> |

Comenzamos creando un directorio nuevo **/subidas** dentro de la carpeta **public** del proyecto.



Accedemos a **routers.js**

Vamos a crear una función de soporte **utilSubirFichero**.

|  |
| --- |
| module.**exports** = {  **name**: **'MiRouter'**,  utilSubirFichero : **async** (binario, nombre, extension) => {  **return new** *Promise*((resolve, reject) => {  nombre = nombre + **"."** + extension;  require(**'fs'**).writeFile(**'./public/subidas/'**+nombre, binario, err => {  **if** (err) {  resolve(**false**)  }  resolve(**true**)  })  })  },  register: **async** (server, options) => { |

|  |
| --- |
| **Arquitectura – funciones de utilidad**  Está práctica no está enfocada a obtener una arquitectura optima, si las funciones de utilidad son comunes a varios routers, se podrían sacar a un módulo a parte que pueda ser obtenido desde el server.method. |

En el controlador **POST /publicar** vamos a especificar que el payload (cuerpo de la petición post) va a contener un steam de datos. Esto no es totalmente obligatorio, si no agregásemos esta configuración desde **req.paypload.foto** obtendríamos el binario de la foto, con la configuración actual vamos a poder obtener más información.

|  |
| --- |
| server.**route**([  {  **method**: **'POST'**,  **path**: **'/publicar'**,  **options** : {  **payload**: {  **output**: **'stream'** }  },  handler: **async** (req, h) => { |

Utilizamos **req.payload.foto.\_data** para obtener el binario y **req.payload.foto.hapi.filename** para obtener el nombre del fichero y la extensión. El nombre con el que se va a guardar la foto es la \_id del anuncio que acabamos que agregar (el nombre de la foto la asocia con el anuncio).

Para llamar a funciones del mismo módulo tenemos que usar **module.exports.<nombre\_función>**

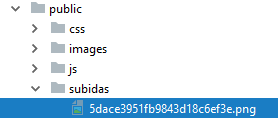
|  |
| --- |
| server.**route**([  {  **method**: **'POST'**,  **path**: **'/publicar'**,  **options** : {  **payload**: {  **output**: **'stream'** }  },  handler: **async** (req, h) => {   **anuncio** = {  **usuario**: **"sin usuario"**,  **titulo**: req.**payload**.**titulo**,  **descripcion**: req.**payload**.**descripcion**,  **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;  **idAnuncio** = id;  }  })   **binario** = req.**payload**.**foto**.**\_data**;  **extension** = req.**payload**.**foto**.**hapi**.filename.split(**'.'**)[1];    **await** module.**exports**.utilSubirFichero(  **binario**, **idAnuncio**, **extension**);   **return respuesta**;  }  }, |

|  |
| --- |
| Para llamar a funciones del propio módulo se usa **module.exports.<función>**, en lugar de **this.<función>**, otra opción muy común es guardar las funciones en variables. |

Antes de probarlo debemos asegurarnos de que el formulario contiene la propiedad **encType="multipart/form-data".** Abrimos la vista correspondiente a **/views/publicar.html** y añadimos la propiedad **encType** en el formulario.

|  |
| --- |
| <**form method="post" action="/publicar" encType="multipart/form-data"**>  <**fieldset class="uk-fieldset"**> |

Guardamos los cambios y ejecutamos la aplicación, probamos a agregar una nueva canción con una imagen como portada. Sí actualizamos el entorno de desarrollo (F5) veremos cómo aparecen los ficheros que acabamos de subir.



## Recuperar y listar los anuncios

Vamos a comenzar añadiendo en **respositorio.js** una función que obtenga todos los anuncios.

La función **obtenerAnuncios** va a retornar una promesa que se resolverá enviando los documentos almacenados en la **colección “**anuncios”. Utilizando la consulta **.find( criterio )** con un criterio vacio **= { }** retornará todos los anuncios. Los anuncios retornados deben ser transformados a un array utilizando la función **toArray**.

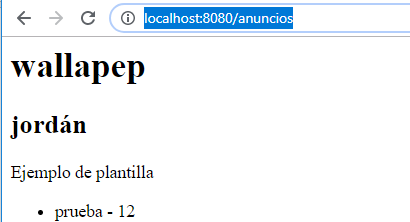
|  |
| --- |
| });  **return promise**; }, obtenerAnuncios : **async** (db, criterio) => {  **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'anuncios'**);  collection.find(criterio).toArray( (err, result) => {  **if** (err) {  resolve(**null**);  } **else** {  *// lista de anuncios* resolve(result);  }  db.close();  });  });   **return promise**; }, insertarAnuncio : **async** (db, anuncio) => { |

Accedemos a **routes.js** , buscamos el controlador que responde a **GET /anuncios.**

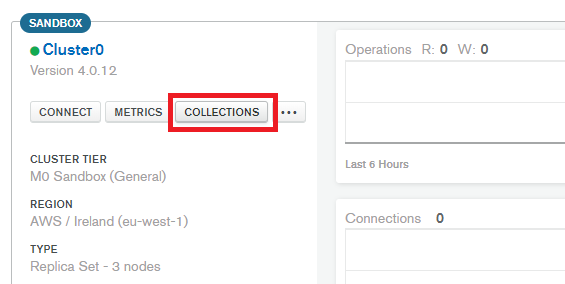
En lugar de introducir los anuncios desde código, vamos a solicitarlos a la base de datos.

|  |
| --- |
| **method**: **'GET'**, **path**: **'/anuncios'**, handler: **async** (req, h) => {   **~~anunciosEjemplo~~** ~~= [  {~~**~~titulo~~**~~:~~ **~~"iphone"~~**~~,~~ **~~precio~~**~~: 400},  {~~**~~titulo~~**~~:~~ **~~"xBox"~~**~~,~~ **~~precio~~**~~: 300},  {~~**~~titulo~~**~~:~~ **~~"teclado"~~**~~,~~ **~~precio~~**~~: 30},  ]~~   **var** criterio = {};  **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  **anunciosEjemplo** = anuncios;  })   **return** h.view(**'anuncios'**,  {  **usuario**: **'jordán'**,  **anuncios**: **anunciosEjemplo** }); } |

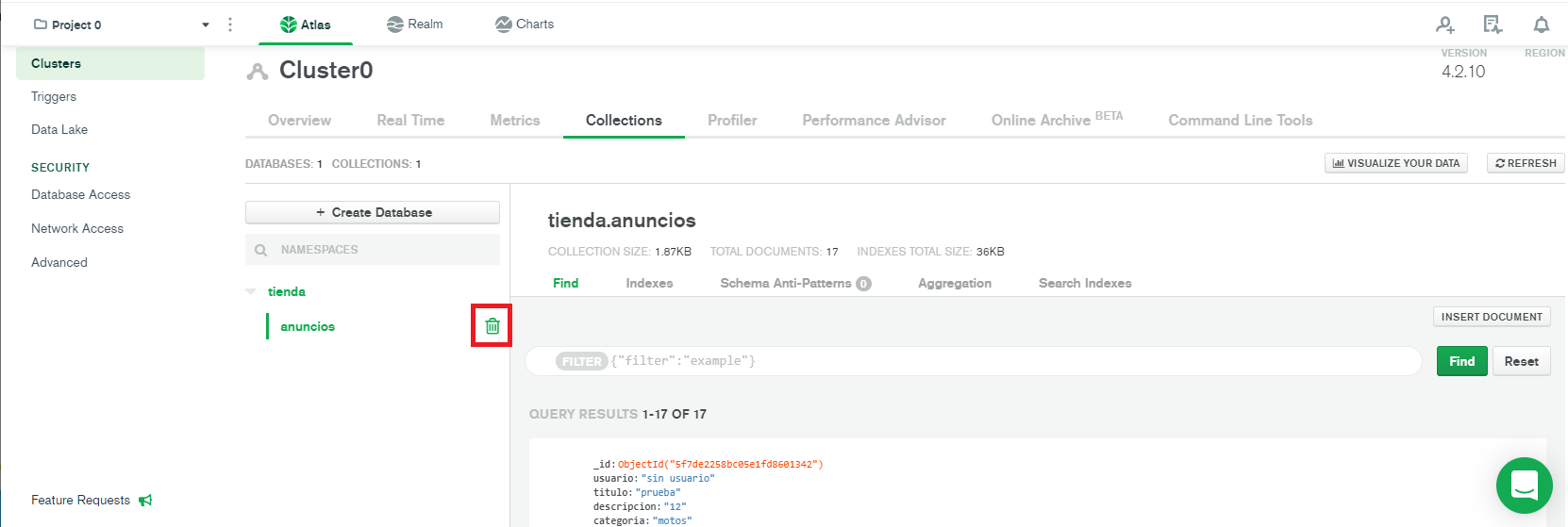
Sí guardamos los cambios y ejecutamos la aplicación podremos ver que la respuesta de <http://localhost:8080/anuncios> , debería ajustarse a la información que tenemos en la base de datos.



Podemos borrar la información de la colección anuncios fácilmente desde el sitio web <https://cloud.mongodb.com/>



Borramos todos los anuncios antiguos (borrando la colección completa), a partir de ahora solo admitimos anuncios con foto.



Sí queremos que la foto sea obligatoria podemos ponerle required en el atributo input del HTML).

Ahora vamos a modificar la vista de **GET /anuncios** para que tenga un mejor aspecto. Lo primero es indicar que utilice el layout base.

|  |
| --- |
| **return** h.view(**'anuncios'**,  {  **usuario**: **'jordán'**,  **anuncios**: **anunciosEjemplo** }, { **layout**: **'base'**} ); |

Ahora solo falta modificar la vista **anuncios.html** para que use un estilo de UiKit. Vamos a limitar el tamaño de la fotografía a 400px.

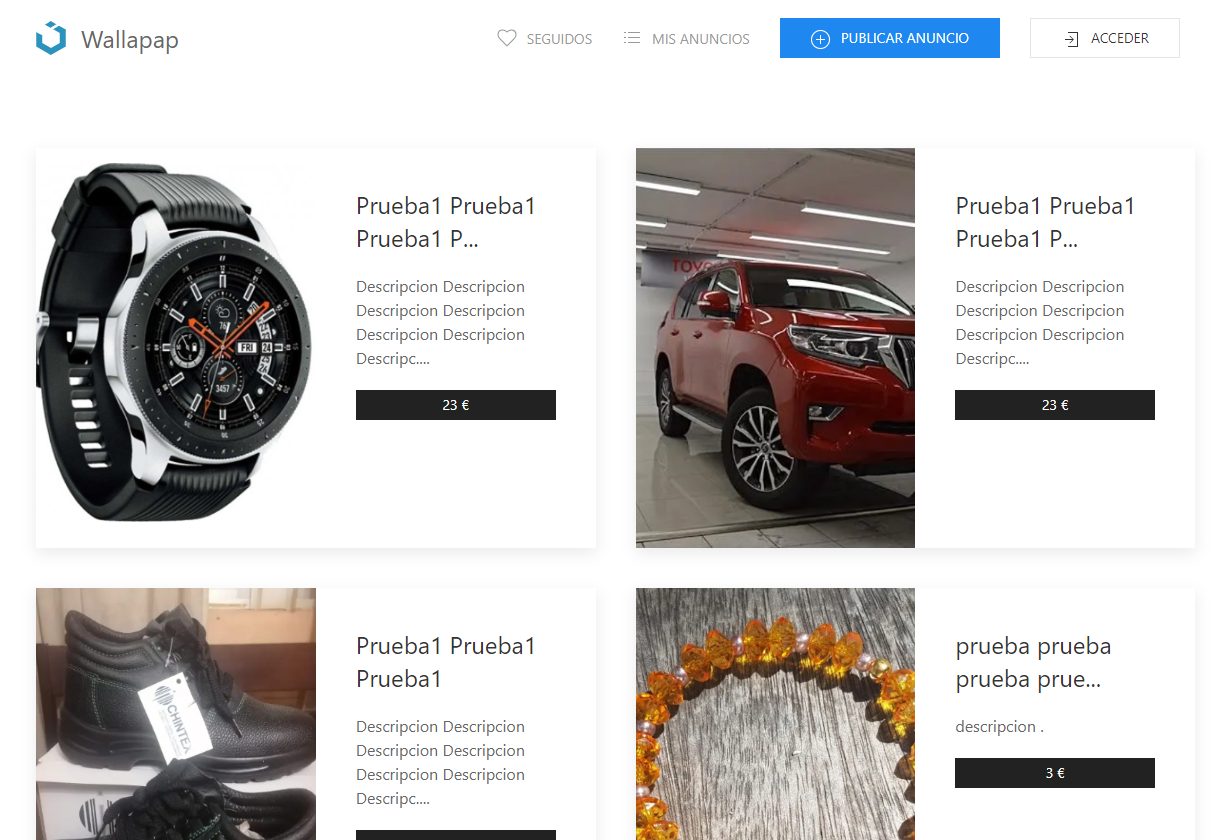
Meteremos 2 anuncios por fila en pantallas @m o superiores, y solo uno en pantallas más pequeñas (ni siquiera hace falta especificarlo, 1-1 es por defecto)

|  |
| --- |
| <**section class="uk-section uk-section-default"**>  <**div class="uk-container"**>  *<!-- sin incluir uk-grid no va el grid -->* <**div class="uk-child-width-1-2@m" uk-grid** >   {{#each anuncios}}  <**div** >  <**div style="height**:400**px**;**"  class="uk-card uk-card-default uk-grid uk-grid-collapse uk-child-width-1-2 uk-margin"**>  <**div class="uk-card-media-left uk-cover-container"**>  <**img style="height**:400**px**;**"  class="uk-cover" src="/subidas/{{ \_id }}.png"**>  </**div**>  <**div**>  <**div class="uk-card-body"**>  <**h3 class="uk-card-title"**>{{ titulo }}</**h3**>  <**p**>{{ descripcion }}.</**p**>  <**a class="uk-width-1-1 uk-button uk-button-secondary uk-button-small"  href="/comprar/{{ \_id }}"**>  {{ precio }} €  </**a**>  </**div**>  </**div**>  </**div**>  </**div**>   {{/each}}  </**div**>  </**div**> </**section**> |

Antes de devolver la lista de anuncios, vamos a recorrerla para limitar las dimensiones de los atributos **título** y **descripción** (ya que se trata de una vista en forma de lista de detalles vamos a mostrar una versión recortada del texto). En algunos casos estos recortes se aplican directamente en las vistas por medio de clases que recortan el texto.

|  |
| --- |
| **method**: **'GET'**, **path**: **'/anuncios'**, handler: **async** (req, h) => {   **var** criterio = {};  **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  **anunciosEjemplo** = anuncios;  })   *// Recorte* **anunciosEjemplo**.forEach( (e) => {  **if** (e.**titulo**.length > 25){  e.**titulo** = e.**titulo**.substring(0, 25) + **"..."**;  }  **if** (e.**descripcion**.length > 80) {  e.**descripcion** = e.**descripcion**.substring(0, 80) + **"..."**;;  }  });   **return** h.view(**'anuncios'**,  {  **usuario**: **'jordán'**,  **anuncios**: **anunciosEjemplo** }, { **layout**: **'base'**} ); } |

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



|  |
| --- |
| **Importante – Longitud de los textos y tamaño de las imágenes**  Se debe verificar que los tamaños de textos e imágenes facilitadas por el usuario no afectan negativamente al layout de la página.  Muchas aplicaciones no guardan directamente la foto que los usuarios suben, sino que las redimensionan antes de salvarlas a la resolución que realmente van a usar <https://www.npmjs.com/search?q=image-resize> (así evitan guardar fotos excesivamente grandes) |

## Listar artículos por criterio

Hasta ahora estamos utilizando un criterio vacío, **var criterio = {};** por lo que se retornan todos los anuncios. Vamos a utilizar el cuadro de búsqueda de la página principal **views/index** <http://localhost:8080>



Tal y como está implementado este formulario envía un **GET /anuncios** con el parámetro **query** con nombre **criterio**.

Modificamos el controlador que responde a **GET /anuncios** para que consulte el criterio, lo aplicamos al campo título. En este caso la consulta retornará únicamente anuncios que tengan ese título.

|  |
| --- |
| handler: **async** (req, h) => {   **anunciosEjemplo** = [  {**titulo**: **"iphone"**, **precio**: 400},  {**titulo**: **"xBox"**, **precio**: 300},  {**titulo**: **"teclado"**, **precio**: 30},  ]   **var** criterio = {};  **if** (req.**query**.criterio != **null** ){  criterio = { **"titulo"** : req.**query**.criterio };  }  **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  **anunciosEjemplo** = anuncios;  })   *// Recorte* **anunciosEjemplo**.forEach( (e) => {  **if** (e.**titulo**.length > 25){  e.**titulo** =  e.**titulo**.substring(0, 25) + **"..."**;  }  **if** (e.**descripcion**.length > 80) {  e.**descripcion** =  e.**descripcion**.substring(0, 80) + **"..."**;;  }  }); |

En este caso y con la condición que hemos aplicado en el find el título del artículo buscado debe coincidir exactamente con el título de uno de los anuncios almacenados



Es sensible a mayúsculas y minúsculas.



No va a ser necesario para está práctica, pero podríamos aplicar comodines u otros criterios en la búsqueda:

* <https://docs.mongodb.com/manual/reference/method/db.collection.find/>
* <https://docs.mongodb.com/manual/reference/operator/query-logical/>

También expresiones regulares/patrones

* <https://docs.mongodb.com/manual/reference/operator/query/regex/>

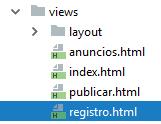
Vamos a ampliar el criterio haciendo que los resultados tengan la cadena introducida en su nombre pero que puedan tener más caracteres antes y/o después.

|  |
| --- |
| **var** criterio = {}; **if** (req.**query**.**criterio** != **null** ){  criterio = { **"titulo"** : {**$regex** : **".\*"**+req.**query**.**criterio**+**".\*"**}}; } |

Guardamos los cambios, ejecutamos la aplicación y comprobamos el nuevo funcionamiento.

## Registro de usuarios

Vamos a permitir que se puedan agregar usuarios en la aplicación, para ello comenzamos creando la vista del formulario de registro **/views/registro.html**



Incluimos el siguiente código en la plantilla, se trata de un formulario muy básico, que solicita los campos **usuario** y **password** y los envía a **POST /registro** (aun sin implementar).

|  |
| --- |
| <**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="/registro"** >  <**div class="uk-margin"**>  <**div class="uk-inline uk-width-1-1"**>  <**span class="uk-form-icon" uk-icon="icon: user"**></**span**>  <**input class="uk-input" name="usuario"  placeholder="E-mail..." type="text"**>  </**div**>  </**div**>   <**div class="uk-margin "**>  <**div class="uk-inline uk-width-1-1"**>  <**span class="uk-form-icon" uk-icon="icon: lock"**></**span**>  <**input class="uk-input" type="password" name="password"  placeholder="Password..."**>  </**div**>  </**div**>   <**input type="submit" value="Crear cuenta" class="uk-button uk-button-primary uk-width-1-1 uk-margin-small-bottom"**>   </**form**>  </**div**> </**section**> |

Accedemos a routes.js y agregamos un nuevo controlador que responderá a **GET /registro**, lo único que va a hacer este controlador es mostrar el formulario contenido en la vista **registro** (no procesar la respuesta que introduce el usuario, eso lo hará POST /registro).

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

El formulario contenido en **registro.html** envía una petición **POST /registro** con los parámetros **usuario** y **password**. Deberíamos almacenar un usuario con esos datos. Es muy recomendable no guardar passwords en texto plano dentro de una base de datos.

Antes de continuar vamos a añadir el “require” del módulo **Crypto** (se trata de un módulo incluido en el Core de node no requiere descarga desde el npm <https://nodejs.org/api/crypto.html> ), el cual vamos a utilizar pare encriptar los password en lugar de guardarlos en texto plano.

Nos situamos en **routes.js** y buscamos la implementación de la respuesta a **/POST usuario.**

1. Obtenemos el parámetro **req.body.password** y lo encriptamos con el módulo crypto
2. Creamos una variable **password** que cotnenga el password encriptado.

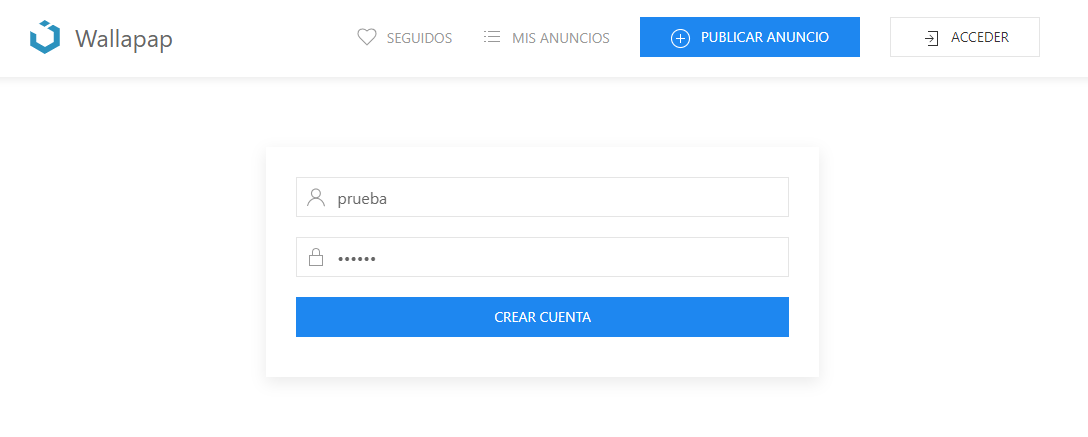
Creamos un usuario que tenga el password encriptado.

|  |
| --- |
| {  **method**: **'POST'**,  **path**: **'/registro'**,  handler: **async** (req, h) => {  **password** = require(**'crypto'**).createHmac(**'sha256'**, **'secreto'**)  .update(req.**payload**.password).digest(**'hex'**);   **usuario** = {  **usuario**: req.**payload**.**usuario**,  **password**: **password**,  }   *// await no continuar hasta acabar esto  // Da valor a respuesta* **await repositorio**.conexion()  .then((db) => **repositorio**.insertarUsuario(db, **usuario**))  .then((id) => {  **respuesta** = **""**;  **if** (id == **null**) {  **respuesta** = **"Error al insertar"** } **else** {  **respuesta** = **"Insertado id: "**+ id;   }  })   **return respuesta**;  } }, |

Agregamos la función **insertarUsuario** al **repositorio.js**.

|  |
| --- |
| **return promise**; }, insertarUsuario : **async** (db, usuario) => {   **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'usuarios'**);  collection.insert(**usuario**, (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**; }, insertarAnuncio : **async** (db, anuncio) => { |

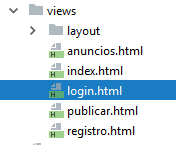
Guardamos los cambios. Accedemos a <http://localhost:8080/registro> y comprobamos que funciona correctamente. Creamos el usuario [prueba](mailto:prueba1@prueba1.com) / prueba.



|  |
| --- |
| **NOTA:** no vamos a implementarlo, pero antes de insertar el usuario habría que comprobar si la colección **usuarios** cuenta ya con un usuario con el mismo Email, en ese caso deberíamos notificarle al usuario que el email ya está en uso. |

# Identificación de usuario

Creamos una nueva vista, **login.html**



El contenido será casi igual al formulario de registro, la única diferencia es que enviará el formulario contra **POST /login**.

|  |
| --- |
| <**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="/login"** >  <**div class="uk-margin"**>  <**div class="uk-inline uk-width-1-1"**>  <**span class="uk-form-icon" uk-icon="icon: user"**></**span**>  <**input class="uk-input" name="usuario"  placeholder="E-mail..." type="text"**>  </**div**>  </**div**>   <**div class="uk-margin "**>  <**div class="uk-inline uk-width-1-1"**>  <**span class="uk-form-icon" uk-icon="icon: lock"**></**span**>  <**input class="uk-input" type="password" name="password"  placeholder="Password..."**>  </**div**>  </**div**>   <**input type="submit" value="Iniciar sesión" class="uk-button uk-button-primary uk-width-1-1 uk-margin-small-bottom"**>   <**a class="uk-button uk-width-1-1 uk-margin-small-bottom"  href="/registro"**>  Crear cuenta  </**a**>   </**form**>  </**div**> </**section**> |

Lo primero es crear la vista que muestre el formulario, entramos en routes.html e implementamos un controlador **GET /login**, únicamente mostrará la vista login para identificarse.

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

Vamos a **repositorio.js** e implementamos una nueva función **obtenerUsuarios()** en el **repositorio.js**, el proceso va a ser el mismo casi el mismo que el utilizado previamente en **obtenerAnuncios ()** incluiremos un parámetro **criterio** configurable que nos servirá para hacer cualquier tipo de selección.

|  |
| --- |
| obtenerUsuarios : **async** (db, criterio) => {  **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'usuarios'**);  collection.find(criterio).toArray( (err, result) => {  **if** (err) {  resolve(**null**);  } **else** {  *// lista de anuncios* resolve(result);  }  db.close();  });  });   **return promise**; }, |

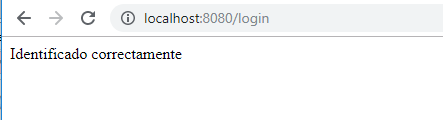
Por el momento únicamente respondemos si hemos conseguido identificar al usuario, si retorna 0 coincidencias no hemos conseguido identificarlo.

Para finalizar, agregamos en **routes.js** un controlador que responda a **POST /login** obtendrá el usuario que se corresponda con los criterios recibidos utilizando **gestorBD.obtenerUsuarios(criterio)**

En caso de que la función retorne coincidencias se trata de un usuario identificado. Debemos recordar que para que la búsqueda sea correcta el password del usuario debe estar encriptado siguiendo el mismo sistema que utilizamos cuando se salvaron los datos del usuario.

|  |
| --- |
| {  **method**: **'POST'**,  **path**: **'/login'**,  handler: **async** (req, h) => {  **password** = require(**'crypto'**).createHmac(**'sha256'**, **'secreto'**)  .update(req.**payload**.password).digest(**'hex'**);   **usuarioBuscar** = {  **usuario**: req.**payload**.**usuario**,  **password**: **password**,  }   *// await no continuar hasta acabar esto  // Da valor a respuesta* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerUsuarios(db, **usuarioBuscar**))  .then((usuarios) => {  **respuesta** = **""**;  **if** (usuarios == **null** || usuarios.**length** == 0 ) {  **respuesta** = **"No identificado"** } **else** {  **respuesta** = **"Identificado correctamente"**;  }  })   **return respuesta**;  } }, |

Guardamos los cambios, ejecutamos la aplicación y accedemos a <http://localhost:8080/login> para comprobar que el funcionamiento es correcto debería permitir identificarnos si introducimos los datos de un usuario ya registrado.



# Autenticación

Existen varias formas de integrar un sistema de autenticación en hapi. Vamos a utilizar un sistema basado en cookies configurables, para ello debemos instalar el módulo externo **hapi/cookie**.

npm install @hapi/cookie@10.1.2

Habilitar la configuración de **@hapi/Cookie**, se configura de la misma forma que el resto de módulos, obtenemos el objeto con el **require(@hapi/Cookie)** y lo registramos en el servidor con **server.register** .

|  |
| --- |
| *// Módulos* **const** Hapi = require(**'@hapi/hapi'**); **const** Inert = require(**'@hapi/inert'**); **const** Vision = require(**'@hapi/vision'**); **const** Cookie = require(**'@hapi/cookie'**); **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** () => {  **try** {  *// Registrar el Inter antes de usar directory en routes* **await** server.register(Inert);  **await** server.register(Vision);  **await** server.register(Cookie);  **await** server.register(**routes**);  **await** server.views({ |

El objeto **server.auth.strategy** nos permite configurar diferentes estrategias de autenticación, lo primero es darles un nombre, nosotros usaremos auth-registrado (se puede dar el que queramos), lo siguiente será configurar en un objeto el funcionamiento de la autenticación.

* **redirectTo**: página a la que se redirige en caso de intentar acceder a una dirección sin permisos.
* **validateFunc** : función asíncrona que comprueba si hay un usuario identificado. Vamos a comprobar si hay un usuario, cokie.usuario y también que la cookie tiene una palabra secreta, sí No existen retornamos { valid : false }, si existe retornamos { valid : true , credentials : <identificador del usuario> } es importante porque podremos acceder al objeto desde cualquier controlador a través de **req.state["session-id"].usuario** (ahí tendremos el identificador del usuario).

|  |
| --- |
| **const** *iniciarServer* = **async** () => {  **try** {  *// Registrar el Inter antes de usar directory en routes* **await** server.register(Inert);  **await** server.register(Vision);  **await** server.register(**Cookie**);  *//Configurar seguridad* **await** server.**auth**.**strategy**(**'auth-registrado'**, **'cookie'**, {  **cookie**: {  **name**: **'session-id'**,  **password**: **'secretosecretosecretosecretosecretosecretosecreto'**,  **isSecure**: **false** },  **redirectTo**: **'/login'**,  validateFunc: **function** (request, cookie){  **promise** = **new** *Promise*((resolve, reject) => {   **usuarioCriterio** = {**"usuario"**: cookie.**usuario**};  **if** ( cookie.**usuario** != **null** && cookie.**usuario** != **""** &&  cookie.**secreto** == **"secreto"**){   resolve({**valid**: **true**,  **credentials**: cookie.**usuario**});   } **else** {  resolve({**valid**: **false**});  }  });   **return promise**;  }  });  **await** server.register(**routes**);  **await** server.views({ |

|  |
| --- |
| **Comprobación del usuario**  La cookie está cifrada con un password, dentro contiene el **usuario** y un **secreto**. |

A partir de este momento ya podemos utilizar la política de seguridad **auth-registrado.**

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

Solo nos falta hacer que el **POST /login** incluya el usuario y el secreto la cookie, para que cuando la función **validateFunc** solicite el cookie.usuario y cookie.secreto lo encuentre.

|  |
| --- |
| {  **method**: **'POST'**,  **path**: **'/login'**,  handler: **async** (req, h) => {  **password** = require(**'crypto'**).createHmac(**'sha256'**, **'secreto'**)  .update(req.**payload**.**password**).digest(**'hex'**);   **usuarioBuscar** = {  **usuario**: req.**payload**.**usuario**,  **password**: **password**,  }   *// await no continuar hasta acabar esto  // Da valor a respuesta* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerUsuarios(db, **usuarioBuscar**))  .then((usuarios) => {  **respuesta** = **""**;  **if** (usuarios == **null** || usuarios.**length** == 0 ) {  **respuesta** = **"No identificado"** } **else** {  req.cookieAuth.set({  **usuario**: usuarios[0].**usuario**,  **secreto** : **"secreto"** });  **respuesta** = **"Identificado correctamente"**;   }  })  **return respuesta**;  } }, |

También podemos crear un nuevo controlador **GET /desconectarse** para eliminar la cookie.

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/desconectarse'**,  handler: **async** (req, h) => {  req.cookieAuth.set({ **usuario**: **""**, **secreto**: **""** });  **return** h.view(**'login'**,  { },  { **layout**: **'base'**});  } }, |

Seguimos el procedimiento:

1. Intentamos acceder a <http://localhost:8080/publicar>
2. Sí vemos que no nos permite hacerlo, tratamos de identificarnos antes <http://localhost:8080/login>
3. Sí ahora nos permite entrar en <http://localhost:8080/publicar> es que la autenticación está funcionando correctamente.
4. Probamos el <http://localhost:8080/desconectar>se
5. Intentamos acceder a <http://localhost:8080/publicar> , no debería dejarnos.

# Colecciones relacionadas – Usuario y anuncios

Vamos a implementar que **GET /misanuncios** retorne únicamente los anuncios publicados por el usuario identificado en la aplicación, para ello tendremos que introducir el usuario que hay actualmente registrado en el req.state["session-id"].usuario (este nombre lo guardamos previamente en el auth-registrado)

Está URL será accesible dentro de la política de seguridad **'auth-registrado'**

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/misanuncios'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** criterio = { **"usuario"** : req.**state**[**"session-id"**].**usuario**};  *// cookieAuth* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  **anunciosEjemplo** = anuncios;  })   **return** h.view(**'anuncios'**,  { **anuncios**: **anunciosEjemplo** },  { **layout**: **'base'**} );  } }, |

Pero realmente ninguno de los anuncios, agregados hasta el momento tenía un usuario asociado (ya que no sabíamos que usuario estaba identificado en cada momento).

Vamos a hacer que se guarde el usuario identificado en el momento de la creación del anuncio, nos dirigimos al controlador de **POST /publicar**.

* Incluimos la “option” auth : ‘auth-registrado’ , para forzar a que el usuario este validado para acceder a la URL
* En el momento de crear el anuncio recuperamos el usuario identificado con **req.state["session-id"].usuario**

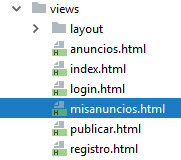
|  |
| --- |
| {  **method**: **'POST'**,  **path**: **'/publicar'**,  **options** : {  **auth**: **'auth-registrado',  payload**: {  **output**: **'stream'** }  },  handler: **async** (req, h) => {   **anuncio** = {  **usuario**: req.**state**[**"session-id"**].**usuario**,  **titulo**: req.**payload**.**titulo**,  **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** = **""**; |

Probamos a agregar un anuncio (este se guardará con el identificador del usuario) , después accedemos a mis anuncios.



Vamos a modificar la vista que responde el controlador, para mostrar una versión en forma de tabla más simplificada. Tablas en UiKit <https://getuikit.com/docs/table> .

Creamos el fichero **misanuncios.html** dentro de la carpeta **/views/**



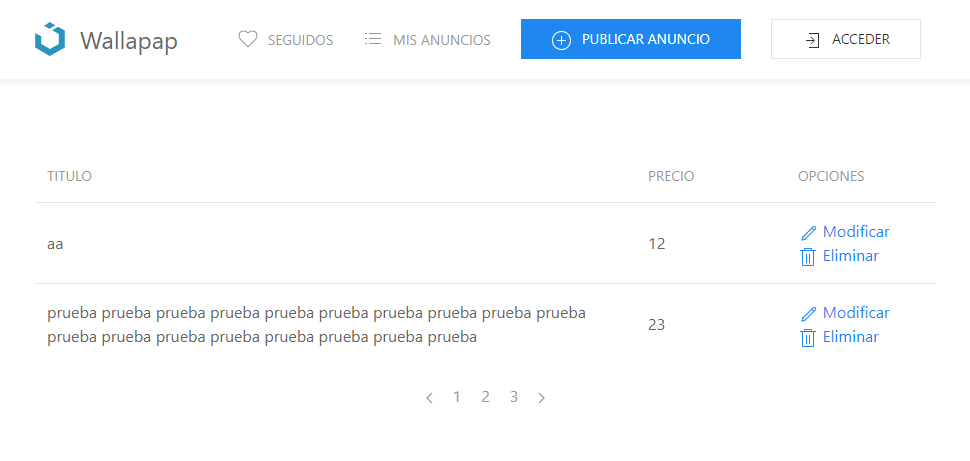
Agregamos el siguiente HTML, será muy parecido al de la vista **anuncios.html** pero en formato tabla, además incluiremos enlaces para modificar y eliminar cada anuncio.

|  |
| --- |
| <**section class="uk-section"**>  <**div class="uk-container"**>  <**div class="uk-overflow-auto"**>  <**table class="uk-table uk-table-hover uk-table-middle uk-table-divider"**>  <**thead**>  <**tr**>  <**th class="uk-table-expand"**>Titulo</**th**>  <**th class="uk-width-small"**>Precio</**th**>  <**th class="uk-width-small"**>Opciones</**th**>  </**tr**>  </**thead**>  <**tbody**>  {{#each anuncios}}  <**tr**>  <**td** >{{ titulo }}</**td**>  <**td** >{{ precio }}</**td**>  <**td**>  <**a class="uk-link" href="/anuncio/{{\_id}}/modificar"**>  <**span uk-icon="pencil"**></**span**>  Modificar  </**a**> <**br**>  <**a class="uk-link" href="/anuncio/{{\_id}}/eliminar"**>  <**span uk-icon="trash"**></**span**>  Eliminar  </**a**>  </**td**>  </**tr**>  {{/each}}  </**tbody**>  </**table**>  </**div**>   <**ul class="uk-pagination uk-flex-center" uk-margin**>  <**li**><**a href="#"**><**span uk-pagination-previous**></**span**></**a**></**li**>  <**li**><**a class="uk-active" href="#"**>1</**a**></**li**>  <**li**><**a href="#"**>2</**a**></**li**>  <**li**><**a href="#"**>3</**a**></**li**>  <**li**><**a href="#"**><**span uk-pagination-next**></**span**></**a**></**li**>  </**ul**>  </**div**> </**section**> |

Cambiamos la vista que responde el controlador, ahora responde **misanuncios**.

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/misanuncios'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** criterio = { **"usuario"** : req.state["session-id"].usuario };  *// cookieAuth* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  **anunciosEjemplo** = anuncios;  })   **return** h.view(**'misanuncios'**,  { **anuncios**: **anunciosEjemplo** },  { **layout**: **'base'**} );  } }, |

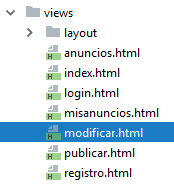
# Probamos a acceder a mis anuncios, para comprobar que la nueva vista funciona.



# Modificar anuncios

La vista **misanuncios**, contiene un botón modificar con un enlace a /anuncio/{id}/modificar

Para poder modificar los anuncios vamos a construir una vista (formulario) para editar el contenido actual del anuncio, la vista se va a parecer mucho a publicar, pero espera recibir del controlador un anuncio, con los datos actuales.



Incluimos el siguiente contenido.

* La petición se envía a POST /anuncio/{{ id }}/modificar
* El formulario debe aparecer ya cubierto con los datos actuales del anuncio. En la mayor parte d ellos elementos del formulario resulta muy sencillo colocar un valor.

El más tedioso es el <select> que permite seleccionar la categoría, debemos añadirle un id a cada campo <option> y utilizar jQuery para marcar como seleccionado el <option> que se corresponda con la categoría.

|  |
| --- |
| <**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="/anuncio/{{ anuncio.\_id}}/modificar"  encType="multipart/form-data"**>   <**fieldset class="uk-fieldset"**>   <**legend class="uk-legend"**>Modificar </**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"  value="{{ anuncio.titulo }}"  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"  value="{{ anuncio.precio }}"  required="true"**>  </**div**>  </**div**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Categoría</**label**>  <**select class="uk-select" name="categoria"  required="true"**>  <**option id="coches"   value="coches"**>Coches</**option**>  <**option id="motos"   value="motos"**>Motos</**option**>  <**option id="bicicletas"   value="bicicletas"**>Bicicletas</**option**>  <**option id="informatica"   value="informatica"**>Informática</**option**>  <**option id="telefonos"   value="telefonos"**>Teléfonos</**option**>  <**option id="moda"   value="moda"**>Moda</**option**>  <**option id="otros"   value="otros"**>Otros</**option**>  </**select**>  </**div**>   <**script**>  $(**"#{{ anuncio.categoria }}"**).attr(**"selected"**,**"selected"**);  </**script**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Descripción</**label**>  <**textarea class="uk-textarea" rows="5"  name="descripcion"  placeholder="perfecto 180 de alto..."  required="true"**>{{ anuncio.descripcion }}  </**textarea**>  </**div**>   <**div class="uk-margin"**>  <**img width="400" src="/subidas/{{ anuncio.\_id}}.png"**>  </**div**>   <**div class="uk-margin"**>  <**label class="uk-form-label"**>Cambiar 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"**>  Modificar  </**button**>  </**fieldset**>  </**form**>  </**div**> </**section**> |

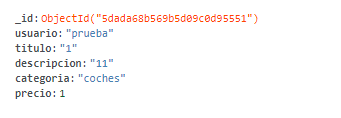
\*Nota: El input de la **foto** no va a tener la propiedad **required**, puesto que solo la van a añadir si quieren “sustituir” la actual.

Vamos a agregar un controlador que responda a la petición **GET /anuncio/{id}/publicar**, debe obtener el anuncio del {id} de la base de datos y adjuntarlo a la plantilla e **modificar.html**

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/anuncio/{id}/modificar'**,  handler: **async** (req, h) => {   **var** criterio = { **"\_id"** : req.**params**.**id** };   **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  *// ¿Solo una coincidencia por \_id?* **anuncio** = anuncios[0];  })   **return** h.view(**'modificar'**,  { **anuncio**: **anuncio**},  { **layout**: **'base'**} );  } }, |

Probamos el código, **algo falla!!,** la búsqueda por nuestro **criterio** no está dando resultados. Eso es porque las \_id de mongo no son cadenas de texto.

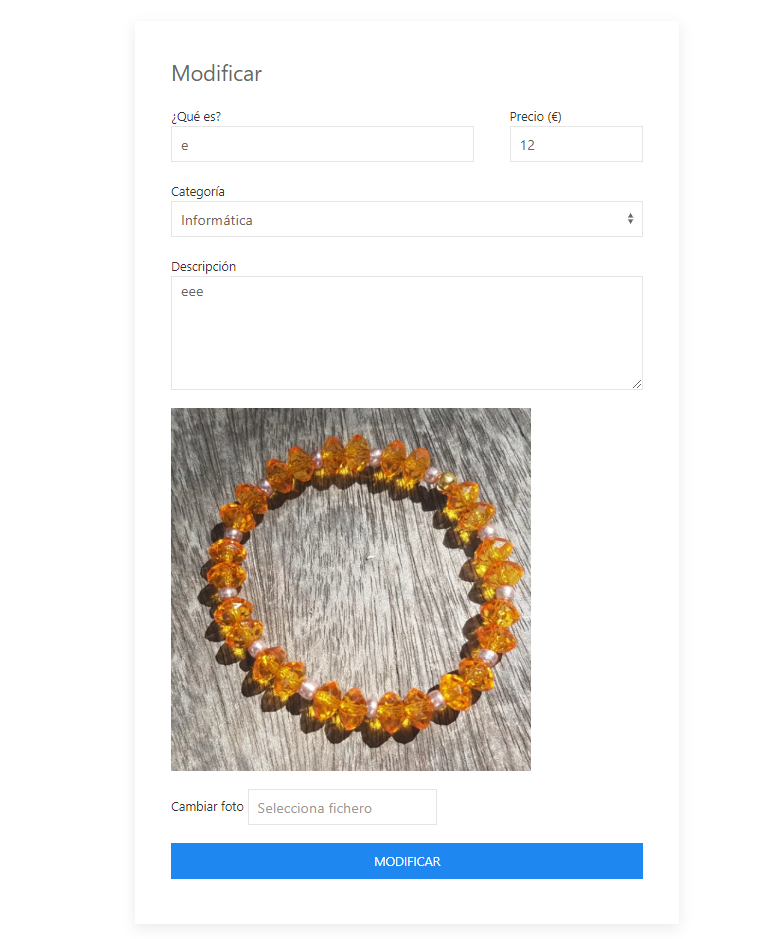
Sí entramos en la colección correspondiente en <https://cloud.mongodb.com/> podemos ver que las \_id son realmente ObjectsIDs, objetos de un tipo específico.



Transformamos nuestra **id** string a aun ObjectID usando el módulo de mondogb.

|  |
| --- |
| **method**: **'GET'**, **path**: **'/anuncio/{id}/modificar'**, handler: **async** (req, h) => {   **var** criterio = { **"\_id"** : require(**"mongodb"**).**ObjectID**(req.**params**.**id**) };   **await repositorio**.conexion() |

Probamos a pulsar sobre el botón modificar de un anuncio para ver que nos retorna la información del anuncio.



Ahora falta implementar el controlador que procesa la respuesta del formulario **POST /anuncio/{id}/modificar**, se va a parecer mucho a lo que hacíamos para publicar un nuevo anuncio.

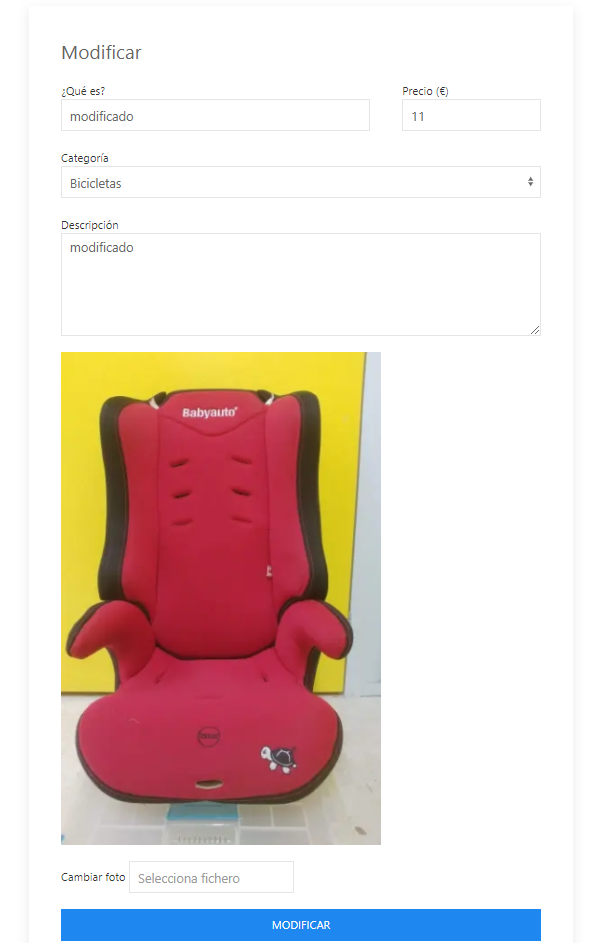
Agregamos a **routes.js** el controlador que responde a **POST /anuncio/{id}/modificar**

|  |
| --- |
| {  **method**: **'POST'**,  **path**: **'/anuncio/{id}/modificar'**,  **options** : {  **auth**: **'auth-registrado'**,  **payload**: {  **output**: **'stream'** }  },  handler: **async** (req, h) => {   *// criterio de anucio a modificar* **var** criterio = { **"\_id"** : require(**"mongodb"**).**ObjectID**(req.**params**.**id**) };   *// nuevos valores para los atributos* **anuncio** = {  **usuario**: req.state["session-id"].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**.modificarAnuncio(db,criterio,**anuncio**))  .then((id) => {  **respuesta** = **""**;  **if** (id == **null**) {  **respuesta** = **"Error al modificar"** } **else** {  **respuesta** = **"Modificado "**;  }  })   *// ¿nos han enviado foto nueva?* **if** ( req.**payload**.**foto**.filename != **""**) {  **binario** = req.**payload**.**foto**.**\_data**;  **extension** = req.**payload**.**foto**.**hapi**.filename.split(**'.'**)[1];   **await** module.**exports**.utilSubirFichero(  **binario**, req.**params**.**id**, **extension**);  }   **return respuesta**;  } }, |

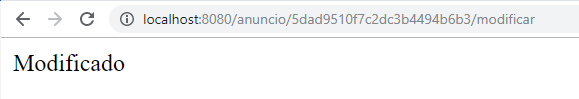
Por último incluimos la parte del **repositorio.js**, implementamos la función **modificarAnuncio(db, criterio selección, nuevos datos)**.

|  |
| --- |
| modificarAnuncio : **async** (db, criterio, anuncio) => {   **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'anuncios'**);  collection.update(criterio, {**$set**: anuncio}, (err, result) => {  **if** (err) {  resolve(**null**);  } **else** {  *// modificado* resolve(result);  }  db.close();  });  });   **return promise**; }, |

Probamos a ejecutar la aplicación y modificamos uno de nuestros anuncios (primero solo los datos, luego implementaremos el sistema para modificar la foto).



Retorna la respuesta de modificado.



# Eliminar anuncio

Abrimos **routes.js** y añadimos el controlador **GET /anuncio/{id}/eliminar.** Una vez eliminado volvemos a la vista **/misanuncios**, enviando por GET un parámetro mensaje (luego veremos cómo procesar este parámetro).

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/anuncio/{id}/eliminar'**,  handler: **async** (req, h) => {   **var** criterio = { **"\_id"** :  require(**"mongodb"**).**ObjectID**(req.**params**.**id**) };   **await repositorio**.conexion()  .then((db) => **repositorio**.eliminarAnuncios(db, criterio))  .then((resultado) => {  ***console***.log(**"Eliminado"**)  })   **return** h.*redirect*(**'/misanuncios?mensaje="Anuncio Eliminado"'**)  } }, |

Implementamos la del **repositorio.js** encargada de eliminar los anuncios**,**  **eliminarAnuncios**.

|  |
| --- |
| eliminarAnuncios : **async** (db, criterio) => {  **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'anuncios'**);  collection.remove(criterio, (err, result) => {  **if** (err) {  resolve(**null**);  } **else** {  resolve(result);  }  db.close();  });  });   **return promise**; }, |

Sí ejecutamos la aplicación podemos observar que al borrar un anuncio se elimina correctamente y nos redirecciona a **/misanuncios** .



¿Se muestra el mensaje de “Anuncio Eliminado”?, parece que el mensaje **NO** se está mostrando en la notificación, tal y como tenemos implementado el sitio el mensaje solo se está procesando como parámetro de la vista (motor de plantillas).

Procesar un parámetro GET desde el cliente: Con una modificación sencilla de **layout/base.html** podemos hacer que el mensaje obtenido como parámetro GET tenga una funcionalidad “equivalente” al parámetro mensaje enviado desde los controladores al motor de plantillas.

|  |
| --- |
| *<!-- NOTIFICACION -->* {{#if mensaje}} <**script**>  UIkit.notification({**message**: **'{{ mensaje }}'**, **status**: **'success'**}) </**script**> {{/if}} <**script**>  **var *mensaje*** = *getUrlParameter*(**'mensaje'**);  **var *tipoMensaje*** = *getUrlParameter*(**'tipoMensaje'**);   **if** ( ***mensaje*** != **""**){  **if** (***tipoMensaje*** == **""** ){  ***tipoMensaje*** = **'primary'**;  }  UIkit.notification({**message**: ***mensaje***, **status**: ***tipoMensaje***})  }   **function** *getUrlParameter*(name) {  name = name.replace(/[\[]/, **'\\['**).replace(/[\]]/, **'\\]'**);  **var** regex = **new *RegExp***(**'[\\?&]'** + name + **'=([^&#]\*)'**);  **var** results = regex.exec(***location***.search);  **return** results === **null** ? **''** :  *decodeURIComponent*(results[1].replace(/\+/g, **' '**));  }; </**script**> |

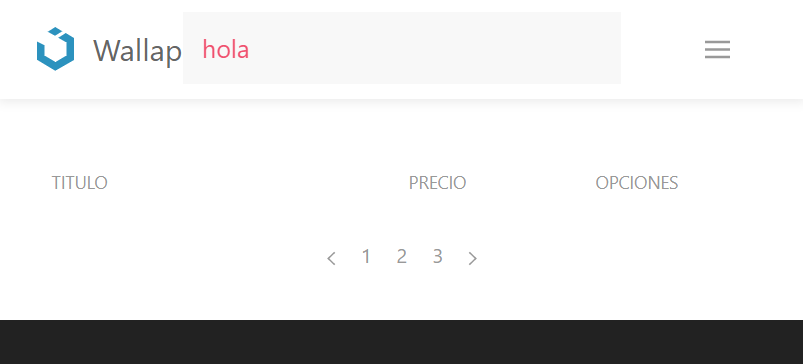
Vamos a admitir los tipos de notificación incluidos en el framework (solo cambia el color), en caso de no recibir tipo de notificación ponemos el por defecto: primary

* primary
* success
* warning
* danger

Como el sistema está activo en todas las páginas podemos hacer pruebas escribiendo URLS de forma manual.

<http://localhost:8080/misanuncios?mensaje=hola>

<http://localhost:8080/misanuncios?mensaje=hola&tipoMensaje=danger>



|  |  |  |
| --- | --- | --- |
| **Importante – Completar redirecciones y mensajes.** Habría que intentar arreglar todas las redirecciones del sitio web, para que ninguna acción mostrase simplemente texto, todas las acciones tienen que retornar a una página HTML.  Por ejemplo:   * **POST /login** -> debería redirigir a /misanuncios anuncios si nos identificamos correctamente   y a /login si nos identificamos incorrectamente, debería incluir un mensaje indicando que el login no es correcto.   * **POST /publicar** -> debería redirigir a /misanuncios , incluyendo un mensaje indicando que el anuncio se agregó correctamente. * **POST /modificar** -> debería redirigir a /misanuncios , incluyendo un mensaje indicando que el anuncio se modificó correctamente. * **POST /registro** -> debería redirigir a /login , incluyendo un mensaje indicando que el usuario se creó correctamente.   **NO** colocar los **return** dentro de código de las promesas   |  | | --- | | return h.redirect('/mi…') |   No puede haber respuestas de este tipo, **todas tienen que ser una vista o una redirección a una página.**   |  | | --- | | **if** (id == **null**) {  **respuesta** = **"Error al insertar" NOOOOOO** } **else** {  **respuesta** = **"Insertado id: "**+ id; **NOOOOOO**  **idAnuncio** = id; |   **Importante.** Hay que tratar de maximizar el feedback, ofreciendo mensajes de todo lo que está haciendo el usuario, borrar, modificar, etc. |

# Revisión de la autenticación

Debemos estar seguros de que el sistema de autenticación se está utilizando en todos los controladores que lo requieren, por ejemplo, tal y como está la aplicación cualquier usuario puede acceder a la vista de modificar de un anuncio, podemos requerir el sistema **auth-registrado** para poder acceder al formulario de modificación.

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/anuncio/{id}/modificar'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** criterio = { **"\_id"** : require(**"mongodb"**).**ObjectID**(req.**params**.**id**) };   **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  *// ¿Solo una coincidencia por \_id?* **anuncio** = anuncios[0];  })   **return** h.view(**'modificar'**,  { **anuncio**: **anuncio**},  { **layout**: **'base'**} );  } }, |

¿Pero es suficiente?, seguramente NO, el usuario que acceder a **GET /anuncio/{id}/modificar**, **POST /anuncio/{id}/modificar**, **GET /anuncio/{id}/eliminar**, debería ser el dueño del anuncio, no basta con que este identificado en la aplicación.

Esto lo podríamos solucionar con una simple comprobación al hacer la consulta que obtiene los anuncios, añadiendo el usuario como criterio.

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/anuncio/{id}/modificar'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** criterio = {  **"\_id"** : require(**"mongodb"**).**ObjectID**(req.**params**.**id**),  **"usuario"**: req.**state**[**"session-id"**].**usuario**  }  **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnuncios(db, criterio))  .then((anuncios) => {  *// ¿Solo una coincidencia por \_id?* **anuncio** = anuncios[0];  })   **return** h.view(**'modificar'**,  { **anuncio**: **anuncio**},  { **layout**: **'base'**} );  } }, |

|  |
| --- |
| **Importante para la seguridad**  Habría que revisar en toda la aplicación:   * Las URLS privadas comprueban que el usuario está autentificado, utilizando la autenticación implementada en **'auth-registrado'** * Modificar y eliminar, comprueban que el usuario es el dueño del recurso.   \*Otra opción que no haremos es crear otro **server.auth.strategy** , más restrictivo que además de comprobar que estás identificado eres el dueño del recurso cuya id figura en la petición. |

# Sistema de paginación

En muchos casos no suele ser adecuado utilizar listados con un gran número de elementos en una sola página, sería recomendable utilizar paginación.

* Impacto negativo en la experiencia de usuario
* Impacto negativo en el rendimiento

Existen varios módulos específicos para crea un sistema de paginación, como **hapi-pagination**  (se integra con hapi, pero no es oficial <https://www.npmjs.com/package/hapi-pagination>). No obstante implementarla desde cero no es complejo y ayuda a comprender su funcionamiento.

En **routes.js** implementamos la función **obtenerAnunciosPg(pag),** no devolverá todos los anuncios sino solo los

correspondientes a una página concreta. Hemos decidido que va a haber por 2 anuncios por página. La página 1 tendrá los anuncios 1 – 2, la página 2 los anuncios 3 – 4, etc.

Primero contamos cuantos anuncios hay en la colección con TODOS los anuncios (**.count**), después solicitamos los anuncios situados en ciertas posiciones utilizando la función **skipt()** , pondremos siempre un límite siempre de 2 resultados utilizando la función **limit()** .

En **repositorio.js** implementamos la función **obtenerAnunciosPg**.

1. Obtiene toda la colección y la cuenta
2. Escapa los primeros registros - **skip**
3. Limita el número total de registros - **limit**

|  |
| --- |
| obtenerAnunciosPg : **async** (db, pg, criterio) => {  **promise** = **new** *Promise*((resolve, reject) => {  **var** collection = db.**collection**(**'anuncios'**);  collection.count( criterio, (err, count) => {  collection.find(criterio).skip( (pg-1)\*2 ).limit( 2 )  .toArray( (err, result) => {   **if** (err) {  resolve(**null**);  } **else** {  *// Guardar el total de anuncios* result.**total** = count;  resolve(result);  }  db.close();  });  })  });   **return promise**; }, |

En el controlador modificamos el contenido de la función **GET /misanuncios**, vamos a admitir que pueda recibir un parámetro **GET** con nombre **pg** que indicara que anuncios hay que cargar (los correspondientes a una página concreta), por ejemplo, **misanuncios?pg=2**.

Debemos ser cuidadosos ya que el valor de los parámetros es siempre un string, hay que transformarlo a entero **parseInt**(), también hay que tener en cuenta que se trata de un parámetro opcional , si no se recibe el parámetro le asignamos valor 1 (ir por defecto a la página 1)

¿División exacta? Debemos tener en cuenta que si el resultado es que la última página es 5,2 debería de ser la 6, es decir quitar la parte decimal y sumar uno.

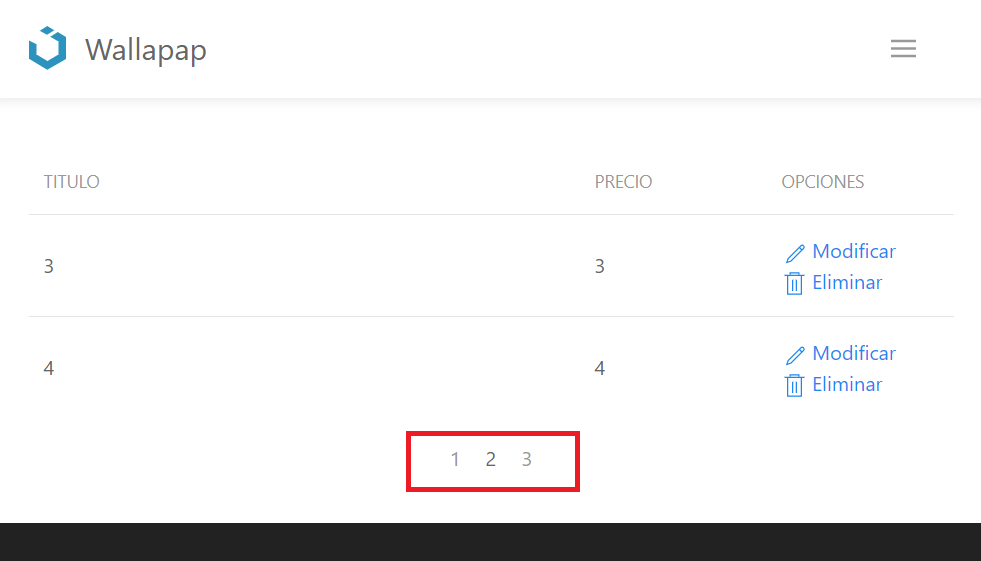
Vamos a crear una lista paginas donde se envíen a la vista todas las páginas que hay que mostrar, la página actual va a ir resaltada con el atributo **clase : "uk-active",** que coincide con la clase de css que el framework utiliza para resaltar la página actual

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/misanuncios'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** pg = *parseInt*(req.**query**.pg); *// Es String !!!* **if** ( req.**query**.pg == **null**){ *// Puede no venir el param* pg = 1;  }   **var** criterio = { **"usuario"** : req.state["session-id"].usuario };  *// cookieAuth* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnunciosPg(db, pg, criterio))  .then((anuncios, total) => {  **anunciosEjemplo** = anuncios;  **pgUltima** = **anunciosEjemplo**.total/2;  *// La págian 2.5 no existe  // Si excede sumar 1 y quitar los decimales* **if** (**pgUltima** % 2 > 0 ){  **pgUltima** = ***Math***.trunc(**pgUltima**);  **pgUltima** = **pgUltima**+1;  }   })   **var** paginas = [];  **for**( **i**=1; **i** <= **pgUltima**; **i**++){  **if** ( **i** == pg ){  paginas.push({**valor**: **i** , **clase** : **"uk-active"** });  } **else** {  paginas.push({**valor**: **i**});  }  }  **return** h.view(**'misanuncios'**,  {  **anuncios**: **anunciosEjemplo**,  **paginas** : paginas  },  { **layout**: **'base'**} );  } }, |

Modificaremos la parte final de la vista **misanuncios.html,** para que recorra la lista de páginas.

|  |
| --- |
| </table>  </div>  <ul class=**"uk-pagination uk-flex-center"** uk-margin>  <li><a href=**"#"**><span uk-pagination-previous></span></a></li>  <li><a class=**"uk-active"** href=**"#"**>**1**</a></li>  <li><a href=**"#"**>**2**</a></li>  <li><a href=**"#"**>**3**</a></li>  <li><a href=**"#"**><span uk-pagination-next></span></a></li>  </ul>  <ul class=**"uk-pagination uk-flex-center"** uk-margin>  **{{#each paginas}}**  <li class=**"{{ clase }}"**>  <a href=**"/misanuncios?pg={{ valor }}"**>**{{ valor }}**</a>  </li>  **{{/each}}**  </ul>  </div>  </section> |

Guardamos los cambios, ejecutamos la aplicación y comprobamos que la paginación funciona en todos los casos.



|  |  |  |  |
| --- | --- | --- | --- |
| **Potencia del motor de plantillas**  A diferencia de otros motores de vistas **handlebars** No permite prácticamente operaciones lógicas ni aritméticas “de forma directa”, es decir algo como lo siguiente NO funcionaria:   |  | | --- | | <**a href="/misanuncios?pg={{ valor + 1}}"**>{{ valor + 1 }}</**a**> |   La alternativa que ofrece es implementar funciones JS y asociarlas al motor, por ejemplo, en **main.js** podríamos introducir una función sumar que sumase a + b.   |  | | --- | | });  **var** handlebars = require(**'handlebars'**); handlebars.*registerHelper*(**"sumar"**, (a, b) => {  **return** a + b; })  **await** server.register(**routes**); **await** server.views({  **engines**: {  **html**: require(**'handlebars'**)  },  **relativeTo**: \_\_dirname,  **path**: **'./views'**,  **layoutPath**: **'./views/layout'**,  **context** : {  **sitioWeb**: **"wallapep"** } }); **await** server.start(); |   Después estaría permitido usar la función sumar en las plantillas de la siguiente forma:   |  | | --- | | <**a href="/misanuncios?pg={{sumar valor 1}}"**>{{sumar valor 1 }}</**a**> | |

# Usuario autenticado desde las vistas

Los controladores van a enviar a las plantillas el identificador del usuario registrado (cuando tengamos uno.

De esta forma el usuario puede utilizarse en la plantilla, por ejemplo para mostrar u ocultar el botón de Login.

Modificamos **base.html,** esperamos que la vista pueda recibir una variable de la plantilla llamada **{{ usuarioAutenticado }}.**

|  |
| --- |
| <**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**>  {{#if usuarioAutenticado}} <**div class="uk-navbar-item"**>  <**div class="uk-button uk-button-default uk-visible@m"**>  {{usuarioAutenticado}}  </**div**> </**div**> {{/if}} {{^usuarioAutenticado}} <**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**> {{/usuarioAutenticado}}  *<!-- 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**> |

A modo de ejemplo lo implementamos en uno de los controladores, por ejemplo, **GET /misanuncios.**

Basta con que enviar la variable **usuarioAutenticado** a la vista, con el usuario que hay autenticado **req.state["session-id"].usuario**

|  |
| --- |
| {  **method**: **'GET'**,  **path**: **'/misanuncios'**,  **options**: {  **auth**: **'auth-registrado'** },  handler: **async** (req, h) => {   **var** pg = *parseInt*(req.**query**.pg); *// Es String !!!* **if** ( req.**query**.pg == **null**){ *// Puede no venir el param* pg = 1;  }   **var** criterio = { **"usuario"** : req.state["session-id"].usuario };  *// cookieAuth* **await repositorio**.conexion()  .then((db) => **repositorio**.obtenerAnunciosPg(db, pg, criterio))  .then((anuncios, total) => {  **anunciosEjemplo** = anuncios;  **pgUltima** = **anunciosEjemplo**.total/2;   *// La págian 2.5 no existe  // Si excede sumar 1 y quitar los decimales* **if** (**pgUltima** % 2 > 0 ){  **pgUltima** = ***Math***.trunc(**pgUltima**);  **pgUltima** = **pgUltima**+1;  }   })   **var** paginas = [];  **for**( **i**=1; **i** <= **pgUltima**; **i**++){  **if** ( **i** == pg ){  paginas.push({**valor**: **i** , **clase** : **"uk-active"** });  } **else** {  paginas.push({**valor**: **i**});  }  }  **return** h.view(**'misanuncios'**,  {  **anuncios**: **anunciosEjemplo**,  **usuarioAutenticado**: req.**state**[**"session-id"**].**usuario**,  **paginas** : paginas  },  { **layout**: **'base'**} );  } |

Probamos a identificarnos y acceder a <http://localhost:8080/misanuncios>

