

**Curso:**

# **Programador Web Inicial-Front End Developer**



Módulo 5:

# CMS y despliegue en Heroku

Unidad 3:

## CRUD (parte 3)



## Presentación

En esta unidad ampliaremos las capacidades de nuestro sitio ofreciendo al usuario la posibilidad de subir imágenes a las novedades que se cargue. Asimismo, y mediante el servicio que nos brinda Cloudinary, mostraremos a los visitantes del sitio versiones recortadas de las imágenes subidas para optimizar la descarga.



## Objetivos

### Que los participantes logren...

- Implementar las funcionalidades necesarias para realizar la subida de archivos al servidor.
- Conectar el sitio con el servicio de Cloudinary para almacenar las imágenes subidas y obtener versiones recortadas de las mismas.



## Bloques temáticos

1. Subida de archivos.
2. Manipulación de imágenes.

# 1. Subida de archivos

Nuestro sitio web permitirá al administrador subir imágenes a las novedades que ya tenemos implementadas. Para esto será necesario hacer algunas modificaciones en el código y estructura de base de datos que ya tenemos.

Necesitaremos primero instalar algunas dependencias nuevas:

## express-fileupload

La dependencia `express-fileupload` nos va a permitir, mediante un middleware, capturar y manipular archivos subidos mediante formularios a nuestro sitio web.



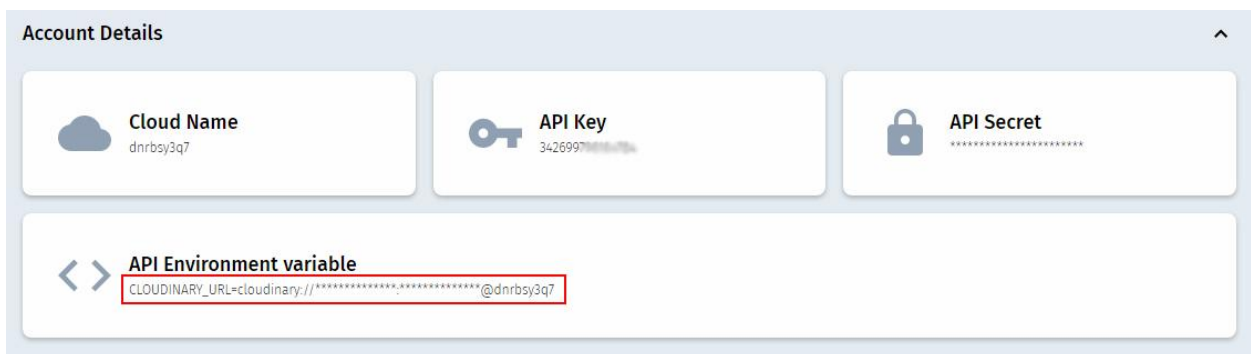
```
npm i express-fileupload
```

## cloudinary

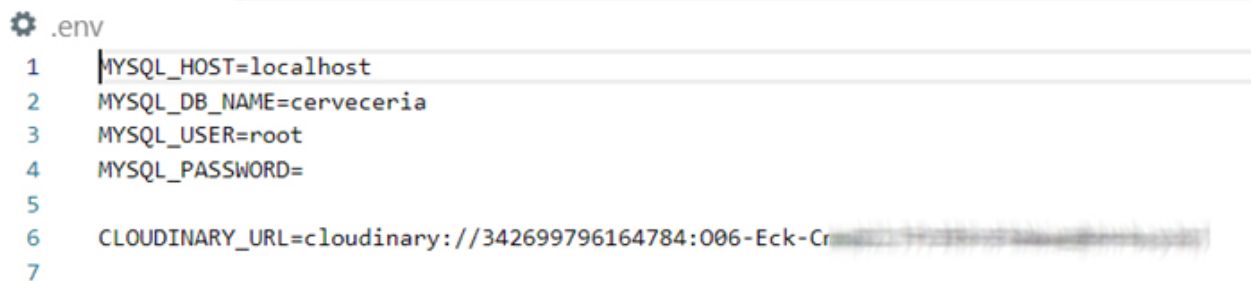
Utilizaremos el servicio gratuito de Cloudinary (<https://cloudinary.com/>) para alojar y manipular las imágenes que se vayan subiendo a las novedades. La dependencia la instalamos de la siguiente manera:

```
npm i cloudinary
```

Para poder utilizar el servicios necesitaremos registrarnos en el sitio y, una vez terminado este proceso, copiar el valor de API Environment variable a nuestro archivos .env con el nombre **CLOUDINARY\_URL**



The screenshot shows the 'Account Details' page of a Cloudinary account. It contains three main sections: 'Cloud Name' with the value 'dnrbsy3q7', 'API Key' with the value '3426997...', and 'API Secret' with a masked value. Below these is a section for 'API Environment variable' which shows a code snippet: `CLOUDINARY_URL=cloudinary://*****@dnrbsy3q7`. The variable name and the full URL are highlighted with a red box.



The screenshot shows a text editor with a file named '.env'. The content of the file is as follows:

```
1 MYSQL_HOST=localhost
2 MYSQL_DB_NAME=cerveceria
3 MYSQL_USER=root
4 MYSQL_PASSWORD=
5
6 CLOUDINARY_URL=cloudinary://342699796164784:006-Eck-Cr*****
7
```

## Agregar imágenes a la novedades

### Paso 1

Incluimos la dependencia de **express-fileupload** y los configuramos como middleware de nuestra aplicación de express. Es importante configurar el uso de

archivos temporales a fin de que podamos, una vez subidos, enviarlos al servidor de Cloudinary donde quedarán almacenados.

```
var fileUpload = require('express-fileupload');

app.use(fileUpload({
  useTempFiles: true,
  tempFileDir: '/tmp/'
}));
```

## Paso 2

Agregamos en la tabla de novedades, la columna **img\_id**, con el tipo de datos varchar, cantidad de caracteres 250 y tildamos la opción NULL (para que permitir la carga de novedades sin imágenes)

## Paso 3

En el archivo **view/admin/agregar.hbs**, agregamos en la etiqueta **<form>**: el atributo **enctype** con el valor **multipart/form-data**. Esto hará que el formulario envíe no solo los campos que veníamos usando hasta ahora, sino que también enviará los archivos que incluyamos.

```
<form action="/admin/novedades/agregar" method="post"
  enctype="multipart/form-data" />
```



Debajo de la etiqueta `input` cuerpo agregamos el **input** del tipo **file** para poder adjuntar la imagen y le ponemos de nombre imagen.

```
<div class="form-group">
  <label>Imagen: <input type="file" name="imagen"
                      id="imagen" /></label>
</div>
```

## Paso 4

Modificamos nuestro archivo **routes/admin/novedades.js** e incluimos tanto la dependencia de **cloudinary** como la de **util** que usaremos para convertir el método **upload** en una función asíncronica.

```
var util = require('util');
var cloudinary = require('cloudinary').v2;

const uploader = util.promisify(cloudinary.uploader.upload);
```

Dentro del controlador `router.post (/agregar)` vamos a verificar, antes de enviar la imagen a nuestro modelo, inicializamos una variable `img_id` como vacía además chequeamos si existe el objeto `files` dentro de la petición y si incluye uno o más objetos dentro suyo.

Esta es la función que realiza express-fileupload; agregar en la petición el objeto files, y dentro de él facilitarnos todos los inputs del tipo file que se hayan enviado con su nombre.

```
router.post('/agregar', async (req, res, next) => {
  try {
    var img_id = '';
    if (req.files && Object.keys(req.files).length > 0) {
      imagen = req.files.imagen;
      img_id = (await uploader(imagen.tempFilePath)).public_id;
    }

    if (req.body.titulo !== "" && req.body.subtitulo !== ""
      && req.body.cuerpo !== "") {

      await novedadesModel.insertNovedad({
        ...req.body,
        img_id
      });

      res.redirect('/admin/novedades')
    }
  }
});
```

En caso de que se hayan enviado archivos, buscaremos uno llamado imagen (recordar el atributo name del input que agregamos) y utilizaremos la ruta temporal donde fue almacenado en nuestra computadora como archivo de origen para enviar al uploader de Cloudinary. Por último, al completarse la subida a cloudinary guardamos el valor de la propiedad public\_id en nuestra variable img\_id.

Este proceso hace que la variable img\_id, que enviamos junto con el resto de los campos a la base de datos, solo contenga un valor (el id publico que cloudinary asigno a nuestra imagen y que necesitaremos mas adelante para obtenerla y manipularla) en caso de que se haya subido una imagen. En caso contrario, enviaremos un valor vacío y la novedad quedará almacenada sin imagen.

## Mostrar imágenes en el listado del administrador

### Paso 1

En el archivo **routes/admin/novedades.js** agregamos la lógica necesaria para enviar junto con cada novedad la imagen que se haya subido en caso de haber una o nada en caso de no haberse subido una imagen a esa novedad.

Para esto utilizaremos el método `image` de `cloudinary` que nos devuelve una etiqueta `img` html completa.



```
router.get('/', async function (req, res, next) {
  var novedades = await novedadesModel.getNovedades();

  novedades = novedades.map(novedad => {
    if (novedad.img_id) {
      const imagen = cloudinary.image(novedad.img_id, {
        width: 100,
        height: 100,
        crop: 'fill'
      });
      return {
        ...novedad,
        imagen
      }
    } else {
      return {
        ...novedad,
        imagen: ''
      }
    }
  });

  res.render('admin/novedades', {
    layout: 'admin/layout',
    usuario: req.session.nombre,
    novedades
  });
});
```

## Paso 2

Finalmente en el archivo **view/admin/novedades.hbs** agregaremos en el bucle `each` la variable de la imagen con triple llave debido a que puede contener HTML

(la etiqueta img).

```
<tr>
  <th scope="col">#</th>
  <th scope="col">Título</th>
  <th scope="col">Imagen</th>
  <th class="text-center" scope="col">Acciones</th>
</tr>

<tr>
  <th scope="row">{{id}}</th>
  <td>{{ titulo }}</td>
  <td>{{ imagen }}</td>
</tr>
```

## Mostrar imágenes en el front

### Paso 1

El proceso para mostrar imágenes en el front es muy similar a lo que hicimos recientemente en el administrador, con la diferencia de que en este caso en vez de utilizar el método image utilizaremos el método url para obtener solo la ruta absoluta de la imagen y después poder usarla como sea necesario.

En caso de que la novedad no tenga una imagen asociada enviaremos como ruta al front la ubicación de una imagen por defecto que mostraremos a los usuarios.



```
var cloudinary = require('cloudinary').v2;
```

```
/* GET home page. */  
router.get('/', async function(req, res, next) {  
  novedades = await novedadesModel.getNovedades();  
  novedades = novedades.splice(0,5);  
  novedades = novedades.map(novedad => {  
    if (novedad.img_id) {  
      const imagen = cloudinary.url(novedad.img_id, {  
        width: 460,  
        crop: 'fill'  
      });  
      return {  
        ...novedad,  
        imagen  
      }  
    } else {  
      return {  
        ...novedad,  
        imagen: '/images/noimage.jpg'  
      }  
    }  
  });  
  res.render('index',{  
    novedades  
  });  
});
```

## Paso 2

Finalmente en el archivo **view/index.hbs** agregaremos dentro del bucle el código para que imprima las imágenes tanto en la lista de novedades como en el modal

```
<ul class="lista-novedades">
  {{#each novedades}}
    <li>
      <a href="javascript:void(0);" data-toggle="modal"
        data-target="#modal{{ id }}">
         {{ titulo}}>
      </a>
    </li>
  {{/each}}
</ul>

<div class="modal-body">
  <h2>{{ subtítulo }}</h2>
  
  <p>{{ cuerpo }}</p>
</div>
```

## Modificar una novedad con imagen

### Paso 1

Modificamos **views/admin/modificar.hbs** e incluimos el input para poder enviar la imagen. A diferencia del formulario de agregar, ahora incluiremos también un input

del tipo checkbox para permitir a los usuarios eliminar la imagen asociada a la novedad.

Es necesario también que incluyamos un input del tipo hidden para enviar junto con el formulario, el id de la imagen actual de la novedad en caso de haber una.

```
<div class="form-group">
  <p>
    <label>Imagen: <input type="file" name="imagen"
      id="imagen" /></label>
  </p>
  <p>
    <label><input type="checkbox" name="img_delete"
      id="img_delete" value="1">
      Eliminar imagen actual?</label>
    <input type="hidden" name="img_original"
      value="{{novedad.img_id}}">
  </p>
</div>
```

## Paso 2

En el archivo **routes/admin/novedades.js** agregamos el método **destroy** de cloudinary que nos permite eliminar imágenes del servicio. Al igual que hicimos con el uploader, utilizaremos la librería util para convertirlo en un método asíncrono.

El código para manejar la modificación de la imagen en la novedad puede parecer complejo a primera vista pero no lo es. Primero inicializamos la variable **img\_id** con el valor que haya venido del input hidden y la variable **borrar\_img\_vieja** que



usaremos para saber si tenemos que llamar al método destroy. De esta forma, pase lo que pase, al enviar el valor a la base de datos, preservaremos el valor original.

A continuación revisamos si el usuario decidió eliminar la imagen novedad, en cuyo caso asignamos null a `img_id` para limpiar el valor de la base de datos.

Si el usuario no tildó la casilla para eliminar la imagen actual repetimos el proceso que hicimos en agregar para ver si se envió una imagen junto con el formulario. En caso afirmativo la subimos a cloudinary, almacenamos el id publico en nuestra variable `img_id` y seteamos `borrar_img_vieja` a verdadero.

En el último paso revisamos el valor de `borrar_img_vieja` y en caso de ser verdadero, ya sea porque el usuario tildó la casilla o subió una imagen nueva, llamamos al método destroy para eliminar la imagen antigua de cloudinary.

Al finalizar todo este proceso, el valor de `img_id` contendrá una de tres opciones según el camino que haya seguido el código, null, el valor del input hidden o el id publico de una nueva imagen.



```
const destroy = util.promisify(cloudinary.uploader.destroy);

router.post('/modificar', async (req, res, next) => {
  try{
    let img_id = req.body.img_original;
    let borrar_img_vieja = false;
    if (req.body.img_delete === "1") {
      img_id = null;
      borrar_img_vieja = true;
    } else {
      if (req.files && Object.keys(req.files).length > 0) {
        imagen = req.files.imagen;
        img_id = (await
          uploader(imagen.tempFilePath)).public_id;
        borrar_img_vieja = true;
      }
    }
    if (borrar_img_vieja && req.body.img_original) {
      await (destroy(req.body.img_original));
    }
  }
}
```

### Paso 3

En la variable obj agregamos img\_id para enviarlo a la base de datos junto con el resto de los campos.

```
var obj = {  
  titulo:req.body.titulo,  
  subtitulo:req.body.subtitulo,  
  cuerpo: req.body.cuerpo,  
  img_id  
}
```

## Para eliminar una imagen

### Paso 1

En el archivo **routes/admin/novedades.js** dentro del controlador de la ruta que elimina las novedades vamos a verificar primero si la novedad tiene una imagen, en cuyo caso, como vamos a eliminar la novedad de nuestra base de datos, la eliminamos también de cloudinary con el método destroy.



```
router.get('/eliminar/:id', async (req, res, next) => {  
  var id = req.params.id;  
  
  let novedad = await novedadesModel.getNovedadById(id);  
  if (novedad.img_id) {  
    await (destroy(novedad.img_id));  
  }  
  
  await novedadesModel.deleteNovedadesById(id);  
  res.redirect('/admin/novedades')  
});
```

## 2. Manipulación de imágenes

Como venimos viendo en los ejemplos podemos pedir a cloundinary versiones de nuestras imágenes recortadas. Esta es solo una de las opciones que ofrece la API de Cloudinary.

Las transformaciones de las imágenes se envían como un objeto con los parámetros que deseamos aplicar como propiedades. El ejemplo más común es el de recortar, que hemos venido usando en los ejemplos.

Esta manipulación en particular recibe como parámetros en ancho, el alto y el tipo de recorte que queremos hacer en la imagen. Los tipos de recorte (parámetro crop) mas comunes son:

- **fit:** Redimensiona la imagen para que quepa dentro de las medidas proporcionadas sin alterar las proporciones.
- **fill:** Redimensionar la imagen sin distorsionarla para que rellene el área especificada. Algunas áreas de la imagen pueden ser cortadas.
- **pad:** Redimensiona la imagen para que quepa dentro de las medidas proporcionadas sin alterar las proporciones y aplica padding en caso de ser necesario para que quepa la imagen completa.

Cloudinary cuenta además con varios efectos que podemos aplicar a las imágenes como escala de grises, contraste o brillo por mencionar algunos mediante el parámetro **effect**.

Podemos ver más efectos y ejemplos en [https://cloudinary.com/documentation/image\\_transformations#applying\\_image\\_effects\\_and\\_filters](https://cloudinary.com/documentation/image_transformations#applying_image_effects_and_filters).

Otra posibilidad sumamente útil es la de agregar imágenes o textos sobre nuestras imágenes, creando de esta forma muy sencilla marcas de agua mediante el

parámetro **overlay**. Para ver ejemplos de esta tecnica podemos visitar [https://cloudinary.com/documentation/image\\_transformations#image\\_and\\_text\\_overlays](https://cloudinary.com/documentation/image_transformations#image_and_text_overlays).

Cabe destacar que todas estas transformaciones de imagen son combinables entre sí, lo que nos brinda una amplia flexibilidad a la hora de utilizar imágenes en nuestros proyectos.

Para tener una guia completa y siempre actualizada de todas las opciones disponibles podemos visitar [https://cloudinary.com/documentation/image\\_transformations](https://cloudinary.com/documentation/image_transformations).



## Bibliografía utilizada y sugerida

### Artículos de revista en formato electrónico:

**Cloudinary.** Disponible desde la URL: <https://cloudinary.com/documentation/>

**Handlebars.** Disponible desde la URL: <https://handlebarsjs.com/>

**npmjs.** Disponible desde la URL: <https://www.npmjs.com/>