

Diplomatura en programación web full stack con React JS

Módulo 6:

Administración, API Rest y despliegue

Unidad 1:

CRUD (parte 1)



Presentación

En esta unidad comenzamos con la creación del CRUD (Create, Read, Update y Delete) de novedades. Para esto vamos a crear una nueva tabla en nuestra base de datos y programaremos las funcionalidades de listar y dar de alta.



Objetivos

Que los participantes logren...

- Crear la tabla necesaria para poder manipular las novedades.
- Crear un listado de las novedades de la base de datos en el administrador.
- Implementar el código necesario para dar de alta las novedades..



Bloques temáticos

1. Listado de novedades.
2. Alta de novedades

1. Listado de novedades

Paso 1

Con nuestra base de datos ya definida en la anterior unidad vamos a crear la tabla **novedades** la cual nos va a servir para listar, modificar, agregar y eliminar novedades.

La tabla contará con 4 campos:

- **id:** un entero de 11 dígitos que será nuestra clave primaria.
- **titulo:** un varchar de máximo 250 caracteres.
- **subtitulo:** un text.
- **cuerpo:** otro text.

| Nombre | Tipo ? | Longitud/Valores ? |
|-----------|-----------|--------------------|
| id | INT ▼ | 11 |
| titulo | VARCHAR ▼ | 250 |
| subtitulo | TEXT ▼ | |
| cuerpo | TEXT ▼ | |

Cuando aceptemos estos valores podemos cargar como mínimo 2 noticias a nuestra tabla para poder empezar a trabajar.

Paso 2

Crearemos el archivo **models/novedades Model.js** donde iremos incorporando todas las funciones que nos permitirán consultar, crear, actualizar o eliminar las novedades.

La función **get Novedades** devuelve un array de filas de la tabla novedades.

```
var pool = require('./bd');

async function getNovedades() {
  var query = "select * from novedades order by id desc";
  var rows = await pool.query(query);
  return rows;
}

module.exports = { getNovedades }
```

Paso 3

En el archivo **routes/admin/novedades.js** importamos nuestro modelo de novedades y agregamos el código necesario para enviarlas como variable en el template.



```
var express = require('express');
var router = express.Router();
var novedadesModel = require('../models/novedadesModel');

router.get('/', async function (req, res, next) {
  var novedades = await novedadesModel.getNovedades();
  res.render('admin/novedades', {
    layout: 'admin/layout',
    usuario: req.session.nombre,
    novedades
  });
});

module.exports = router;
```

Paso 4

Modificamos también nuestro template **views/admin/novedades.hbs** para que liste las novedades recibidas desde el controlador. Para este fin, primero agregamos el código necesario para mostrar una fila estática (que no refleje datos de la base de datos), la cual podemos usar de ejemplo para ajustar nuestro diseño.



```
<div class="container" style="margin:50px auto">
  <div class="row">
    <div class="col">
      <p class="text-right">Hola {{ usuario }} !
      <a href="/admin/login/logout" class="btn btn-sm btn-danger">
        Cerrar sesión <i class="fa fa-sign-out"></i></a></p>
    </div>
  </div>
  <div class="row">
    <div class="col-7">
      <h3>Listado de Novedades</h3>
    </div>
    <div class="col-2 text-right">
      <a href="#" class="btn btn-success"> <i class="fa fa-plus">
        </i>Nuevo</a>
    </div>
  </div>
</div>
```



```
<div class="row">
  <div class="col">
    <table class="table">
      <thead class="thead-dark">
        <tr>
          <th scope="col">#</th>
          <th scope="col">Título</th>
          <th scope="col" class="text-center">Acciones</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <th scope="row">1</th>
          <td>va el titulo</td>
          <td class="text-center">
            <a href="#" class="btn btn-sm btn-secondary">
              <i class="fa fa-pencil"></i></a>
              <a href="#" class="btn btn-sm btn-secondary">
                <i class="fa fa-trash"></i></a>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>
```

Paso 5

Finalmente en el archivo **view/admin/novedades.hbs** agregaremos el bucle each de Handlebars para recorrer el array de novedades que recibimos.



```
<tbody>
  {{#each novedades}}
  <tr>
    <th scope="row">{{id}}</th>
    <td>{{titulo}}</td>
    <td class="text-center">
      <a href="#" title="Editar" class="btn btn-sm btn-secondary">
        <i class="fa fa-pencil"></i></a>
      <a href="#" title="Eliminar" class="btn btn-sm btn-secondary">
        <i class="fa fa-trash"></i></a>
    </td>
  </tr>
  {{/each}}
</tbody>
```



1. Alta de novedades

Paso 1

En el archivo **models/novedades Model.js** agregaremos la función necesaria para dar de alta una novedad.

```
async function insertNovedad(obj) {  
  try {  
    var query = "insert into novedades set ? ";  
    var rows = await pool.query(query, [obj]);  
    return rows;  
  } catch (error) {  
    console.log(error);  
    throw error;  
  }  
}  
  
module.exports = { getNovedades, deleteNovedadById,  
  insertNovedad }
```

Paso 2

Modificamos nuestro archivo **views/admin/novedades.hbs** creamos un link a una nueva ruta que utilizaremos para mostrar el formulario donde ingresaremos las novedades.

```
<div class="col-2 text-right">
  <a href="/admin/novedades/agregar" class="btn btn-success">
    <i class="fa fa-plus"></i>Nuevo</a>
</div>
```

Paso 3

En el archivo **routes/admin/novedades.js** creamos el nuevo manejador de ruta que mostrará el formulario de alta.

```
router.get('/agregar', (req, res, next) => {
  res.render('admin/agregar', {
    layout: 'admin/layout'
  });
});
```



Paso 4

Creamos el archivo **views/admin/agregar.hbs** con el código necesario para el formulario. Dirigimos el action a /admin/novedades/agregar.

```
<div class="container" style="margin:100px auto">
  <div class="row">
    <div class="col-6 offset-3">
      <h4>Agregar una nueva novedad</h4>
      <form action="/admin/novedades/agregar" method="post">
        <div class="form-group">
          <input type="text" class="form-control"
            placeholder="Titulo" name="titulo">
        </div>
        <div class="form-group">
          <textarea type="text" class="form-control"
            placeholder="Subtitulo"
            name="subtitulo"></textarea>
        </div>
        <div class="form-group">
          <textarea type="text" class="form-control"
            placeholder="Cuerpo" name="cuerpo">
          </textarea>
        </div>
        <button type="submit" class="btn btn-
          primary">Guardar</button>
      </form>
    </div>
  </div>
</div>
```

Paso 5

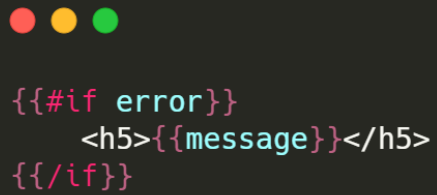
De vuelta en el archivo **routes/admin/novedades.js**, agregaremos el controlador necesario para capturar los datos enviados por el formulario. Una vez recibidos, validamos que tengamos todos los datos necesarios y se los pasamos a la función del model para guardarlos en la base de datos.

En caso de no tener todos los datos necesarios o que se produzca un error en el grabado de la información volveremos a mostrar el mismo template pasándole una variable que indique que hubo un error y una descripción del mismo.

```
router.post('/agregar', async (req, res, next) => {
  try {
    if (req.body.titulo !== "" && req.body.subtitulo !== "" &&
        req.body.cuerpo !== "") {
      await novedadesModel.insertNovedad(req.body);
      res.redirect('/admin/novedades')
    } else {
      res.render('admin/agregar', {
        layout: 'admin/layout',
        error: true, message: 'Todos los campos son requeridos'
      })
    }
  } catch (error) {
    console.log(error)
    res.render('admin/agregar', {
      layout: 'admin/layout',
      error: true, message: 'No se cargo la novedad'
    });
  }
});
```

Paso 6

En el archivo **view/admin/agregar.hbs** agregamos el `if` para que imprima el error en caso de haber uno.



```
{{#if error}}  
  <h5>{{message}}</h5>  
{{/if}}
```




Bibliografía utilizada y sugerida

Artículos de revista en formato electrónico:

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

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