Proyecto Final

Picarat Café – Manual Técnico

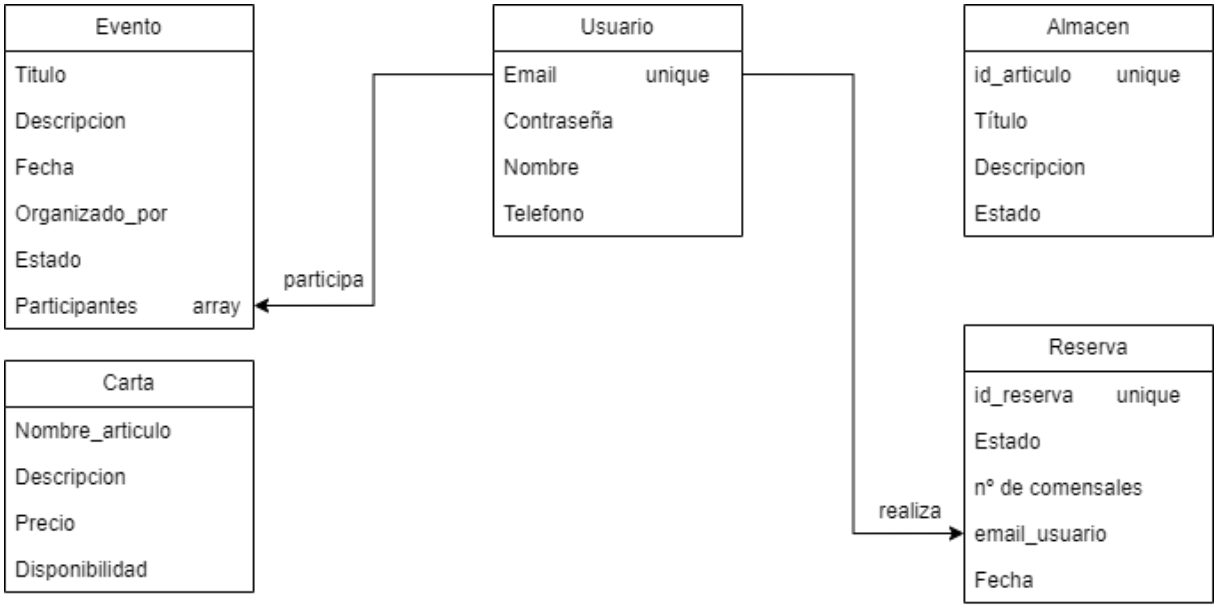
2º Desarrollo de Aplicaciones Web

Por César Rodríguez González

Contenido

1 Base de Datos

Tras toda el desarrollo de la página la base de datos a sufrido ligeros cambios y por lo tanto el modelo relacional también.



Para empezar, se han suprimido algunos de los registros de las colecciones a que en realidad no aplicaban como es el caso de 'Referencia' en la colección 'Artículos'

Otro punto a destacar en el cambio del modelo es la desaparición de ciertas 'relaciones' que en realidad no eran relaciones, soro eran acciones que el usuario/administrador realizaba sobre la base de datos. En su lugar se han especificado las relaciones reales que existen entre los usuarios, las reservas y los eventos. Esta relación se explicará más adelante.

2 Componentes de la aplicación

A continuación, se procederá a realizar un recorrido con su correspondiente explicación sobre los archivos más importantes que componen la aplicación. Cabe recalcar que en este documento se van a especificar sobre los archivos funcionales.

2.1 Server.js

Este archivo es el principal de la aplicación, encargado de todo lo relacionado con las funciones típicas de un servidor back-end. Dentro de la aplicación tiene las siguientes funcionalidades principales:

- Sirve el servidor que alojará nuestro proyecto. Esto se hace mediante la librería express, que nos permitirá además servir ficheros estáticos HTML con CSS y JS que es lo que alimenta al front-end

- Servirá los ficheros HTML, lo que nos permite sustituir las rutas absolutas de los ficheros (.html) con la de una ruta propia de una página web. Esto se puede hacer mediante los siguientes bloques de código que se pueden encontrar a lo largo del fichero:

```
// Página de inicio, al acceder se cargan los eventos
app.route('/').get((_req, res) => {
  return res.sendFile(path.join(__dirname, 'public', 'index.html'))
})
```

  En este caso, al acceder a la ruta principal se estará enviando el fichero correspondiente al índice o página principal

- Crea inicialmente una sesión vacía. Esta sesión nos permitirá almacenar la información del usuario cuando acceda o cuando se registre. Para ello se ha usado la librería express-session. Esta función corresponde a la siguiente parte del código:

```
app.use(session({
  secret: crypto.randomBytes(16).toString("hex"),
  resave: false,
  saveUninitialized: true,
  cookie: { secure: true }
}));
```

  Cabe destacar también la utilización de la librería incorporada crypto, que nos permitirá generar una clave aleatoria en hexadecimal específica para cada sesión.

- Nos servirá como intermediario entre la base de datos y la página. Para ello se han utilizado diversos end-points, cada uno con su funcionalidad específica, devolviendo errores cuando algo no ha salido bien y códigos de confirmación cuando todo ha ido correctamente

2.2 Mongodb.js

También alojado en la raíz del documento, este fichero nos permitirá conectar directamente con la base de datos. La función principal de este fichero es servir las funciones necesarias al back-end para manejar las conexiones a la base de datos y realizar acciones sobre la misma.

Este, por ende, será el único fichero que importará las funciones y clases necesarias desde la librería específica de MongoDB en NodeJS: 'mongodb', y la librería bcrypt ara el encriptado de contraseñas, que se explicará más adelante

La mayoría de las funciones dentro de este fichero tienen una estructura similar ,se conectan a la base de datos, realizan las acciones pertinentes, y se cierra la conexión. A continuación, se adjunta un ejemplo, en este caso es la función creada para encontrar elementos dentro de una base de datos con una query específica:

```
async function fetchWithQuery(query, coleccion){
    await MongoConnection.connect()
    const collection = bd.collection(coleccion)

    const result = collection.find(query)
    let results = []


    for await (const doc of result) {
        results.push(doc)
    }

    await MongoConnection.close()
    return results
}
```

Se requiere la query para buscar los datos, dada desde el servidor ya que esta función se aplica varias veces y con diferentes usos en el código, y la colección a la que nos queremos conectar. Cuando se ha realizado la búsqueda se recorre el puntero de la conexión para encontrar los documentos, se cierra la conexión y se devuelven los datos encontrados.

2.3 Mongo-init.js

Este archivo se podría decir que es un fichero de configuración para luego más tarde crear la base de datos. Este fichero contiene las funciones necesarias para crear la base de datos, las colecciones que constan dentro de la misma, la creación de los índices y la inserción de algunos datos de ejemplo.

Este fichero se usará cuando se lance el comando docker-compose

3 Archivos JavaScript públicos

En lo que al front-end respecta, los ficheros tienen una estructura muy similar. Para explicarlos se pondrá de ejemplo el fichero 'index.js' ya que realiza las funciones generales.

Lo primero que se hace es esperar a que los elementos de la página estén cargados. Una vez ha pasado eso se cargará la información necesaria para la funcionalidad de la página. Todo esto se hace mediante peticiones ajax

```
document.addEventListener('DOMContentLoaded', () => {

    $.ajax({
        url: '/',
        type: 'POST',
        success : async (response) => {
            await cargarEventos(response);
            await funcionalidadBotones();
            await cargarInfoModal();
            await getButton();
        },
        error: (xhr) => {
            Swal.fire({
                title: "Oh oh...",
                text: "Parece que estamos teniendo problemas para cargar la información de esta página",
                icon: "error"
            });
        }
    })
})
```

Para la carga de elementos se llama a la ruta de la página, si se requieren otras funcionalidades se llama a los en-point específicos. En este caso lo primero que se hace es llamar a la función cargarEventos, a la que se le enviará la información recogida sobre los mismos y se procesará la información para mostrarla en los recuadros correspondientes.

Una vez listos todos los eventos estáticos se cargan las funcionalidades de la página, como botones y otros elementos con , como el desplegable de la modificación de los elementos.

Por último, se llamará a un end-point específico que nos devolverá los botones de acción de los administradores únicamente si el usuario que está dado de alta en la página es administrador.

Si por alguna razón la llamada inicial falla, se interrumpirán el resto de procesos activando un pop-up de error.

Luego, cada fichero tiene sus acciones específicas, como las llamadas a los end-points que realizan las reservas en las páginas de reservas o la manera en la que se carga la información de manera específica para cada página.

Lo que sí coinciden son las llamadas a las funciones de modificación, borrado y añadido, las cuales se pueden encontrar a lo largo de todos los ficheros de JavaScript estáticos. Un ejemplo de estas llamadas puede ser el siguiente para el alta de un artículo en la base de datos

```
$('.btn_delete').click(() => {

    let nombre = document.getElementById('titulos_album').value;
    raw = {
        coll : 'articulos',
        data : {
            Titulo : nombre,
        }
    }
    $.ajax({
        url: '/deleteElement',
        type: 'POST',
        data: JSON.stringify(raw),
        contentType: 'application/json',
        error: function(xhr, status, error) {
            if (xhr.status === 500) {
                Swal.fire({
                    title: "¡Error!",
                    text: "No se ha podido eliminar el documento",
                    icon: "error"
                });
            }
        }
    }).done(() => {
        Swal.fire({
            title: "¡Elemento modificado!",
            text: "Los cambios se verán reflejados al refrescar la página",
            icon: "success"
      });
      window.modal1.close()
    })
})
```

Al activar el evento, se obtiene el valor del select que muestra todas las opciones disponibles, manda el título para hacer la query, la colección a modificar y realiza la llamada al end-point creado en el back que se dedica principalmente a mandar esos datos a borrar.

Si todo ha ido correctamente, termina y da un mensaje de confirmación, de lo contrario, muestra un mensaje de error.

Estos mensajes se han realizado con la herramienta SweetAlert, de la que hablaremos más adelante.

4 Implementación de Docker

En este punto se explicarán los ficheros relevantes a la Dockerización del proyecto

4.1 .dockerignore

Lo primero que se ha implementado es un fichero .dockerignore en el que, en este caso, añadiremos la carpeta '/node-modules', para así agilizar la construcción de las imágenes. Además de otros archivos que solo son útiles en local como pueden ser los archivos de diagramas o de texto

4.2 Dockerfile

Cada parte tendrá su propio archivo dockerfile. En este caso, una para la parte de NodeJs y otra para la parte de MongoDB.

Para preparar la imagen de Mongo simplemente se importa una imagen común de Mongo, se expone en el puerto predeterminado y se especifica que al iniciarse la imagen debe levantarse Mongo.

La que tiene algo más de complicación pero también es bastante sencilla es la construcción de la imagen de NodeJS. En este caso los pasos a seguir de la aplicación son los siguientes:

- Se descarga la imagen de node
- Se pasan las dependencias del directorio local al Workdir '/app'
- Se instalan las dependencias
- Se copia el resto del directorio actual a /app
- Se expone el servidor en el puerto 3030
- Se especifica que al levantarse la imagen se debe ejecutar el servido con un 'npm start'

4.3 Docker-compose

Este archivo docker-compose está preparado tanto como para levantar los contenedores necesario como para construir las imágenes necesarias para ello.

El flujo para los dos contenedores es el mismo:

- Se construye la imagen
- Se selecciona el nombre de la red, la imagen a usar y el nombre del contenedor
- Se especifican los puertos
- Se indican los nombre de los hosts dentro de la red

Luego, cada contenedor tendrá sus propias configuraciones. Por ejemplo, en el contenedor de NodeJS se especifica que el archivo que debe coger para el entorno virtual es el archivo de .env dentro de la raíz.

Por otro lado, para el contenedor de Mongo se le lanza una instrucción que se especifica que cuando se vaya a crear el contenedor se lance un fichero de configuraci´´on. En este caso ese fichero contendrás las funciones necesarias para iniciar nuestra base de datos, darle estructura e insertar algunos datos, ese es el fichero mongo-inti.js

4.4 Docker workflows

Por último, se ha creado un fichero de workflow que conecta el repositorio de GitHub con el repositorio de Docker Hub. Con la configuración actual del fichero, el flujo de trabajo para cada imagen sería el siguiente

- Se hace un push a la rama main del repositorio
- Se inicia sesión en Docker Hub
- Se construye la imagen
- Se suben los cambios al repositorio

5 Aplicaciones y Librerías usadas

A lo largo del proyecto se han utilizado varias tecnologías para desarrollar el mismo. Muchas de estas tecnologías he tenido que aprenderla sobre la marcha. A continuación, dejo como manifiesto aquellas más significativas y la repercusión que han tenido en el proyecto.

5.1 Desarrollo General

Para el desarrollo general del proyecto como el desarrollo de código, control de versiones, visualización de base de datos y construcción de imágenes se han usado las tecnologías generales para ello. En orden:

- Visual Studio Code
- Git Hub (apoyado con la extensión para VSCode, GitGraph)
- MongoDB Compass
- Docker Desktop

5.2 MongoDB

Como ya se ha comentado en otras partes de la documentación, la bse de datos utilizada es MongoDB. Esta se complementa muy bien a nivel de usuario con NodeJS y otorga una documentación sencilla de seguir y bastante accesible:

https://www.mongodb.com/docs/drivers/node/current/

5.3 Sweet Alert 2

Esta ha sido la librería utilizada par arealizar de manera más sencilla las notificaciones de alerta devueltas por las peticiones del proyecto. Con una gran capacidad de personalización lo hace perfecto para adornar un poco las respuestas como ha sido mi caso.

La documentación ha sido la oficial del proyecto, la cual se puede encontrar en el siguiente enlace:

https://sweetalert2.github.io/

5.4 Bcrypt

Con la necesidad de encriptar las contraseñas se encontró la librería diseñada para NodeJS, bcrypt. Esta librería permite al desarrollador convertir texto plano en un ‘hash’ configurable y completamente seguro.

Una vez encriptada la contraseña se puede comparar mediante las funciones que ofrece la librería para saber si una contraseña pertenece al ‘hash’ .

El enlace a la documentación seguida es el siguiente:

https://www.npmjs.com/package/bcrypt

5.5 Figma

Herramienta desconocida hasta la fecha que me ha permitido visualizar de manera clara todos los elementos que conformarían parte de la aplicación.