Prácticas DAI

0. Práctica 0

- Instalamos node.js, docker y mongodb. Herramientas para el desarrollo de las prácticas.

- Configuramos la base de datos en un docker-compose.yml:
    - Base de datos NoSQL mongoDB puerto 27017.
    - Cliente gráfico Mongo Express puerto 8081.

1. Práctica 1

- Instalamos los distintos drivers para generar el package.json y el directorio node_modules.
    - **npm i mongodb**
    - **npm i -D @types/node**
- Analizando el código del guion, vamos a incluir los datos en la Base de datos empleando el fetch nativo de node, con una función asíncrona. Al final del seed.js podemos ver la inserción consecutiva con .then() y .catch().
    - Es necesario: **npm i node-fetch**. (Librería que permite realizar el fetch)
- User y password de la BD en un .env, evitando que estén en el código.

- Método “insertMany” para incluir los datos en las colecciones.
- Mongodump para hacer la copia de seguridad (backup)

2. Práctica 2

2.1. Práctica 2.1

- En esta práctica partimos de la BD anterior y vamos a realizar la tienda online.
    - Framework: Express. Comando para instalar: **npm i express**.
    - Interacción con BD / Esquemas: Mongoose. Comando para instalar: **npm i mongoose**.
    - Motor de plantillas: Nunjucks. Comando para instalar: **npm i nunjuks chokidar**. Se instala también chokidar ya que es una librería que permite monitorear cambios en archivos y directorios en tiempo real.
- Generamos la estructura de carpetas que indica el guion copiando el código que nos aporta.
    - Dentro de model tenemos dos archivos que representan el Modelo del proyecto. El primero define la conexión que mongoose hace con la base de datos. El segundo el esquema “ODM” que define mongoose para acceder a los datos de la BD.
    - Dentro de public situamos archivos estáticos como CSS e imágenes.
    - Dentro de routes encontramos el Controlador, que hace el routing.
    - Dentro de views se sitúa la Vista, y es la tarea para esta práctica. Empleamos Bootstrap para el CSS y tenemos en cuenta herencia en las páginas. (Partimos de una página base.html y generamos las demás a partir de ella con bloques de contenido {%block contenido %} {% endblock %}

2.2. Práctica 2.2

- En esta práctica partimos de la tienda creada en la práctica anterior y configuramos las búsquedas y el carrito.
- Respecto al formulario de búsqueda, tenemos que añadir el urlencoded en el servidor, ya que HTTP es un protocolo ASCII.
- Para guardar la información de las compras en el carrito, empleamos las sesiones de express. Comando para instalar: npm i express-session. En **Express**, las sesiones se gestionan

mediante la librería **express-session**, que permite almacenar información del usuario en el servidor entre solicitudes. Cuando un usuario se conecta, se crea una sesión identificada por un **ID único** almacenado en una **cookie** en el navegador del cliente. Esta cookie se envía con cada solicitud, lo que permite recuperar la sesión en el servidor y mantener el estado (como el inicio de sesión) a lo largo de la navegación.

3. Práctica 3

3.1. Práctica 3.1

- En esta práctica vamos a implementar la autentificación de usuarios en nuestra tienda online, basándonos en Tokens JWT.
- En primer lugar definimos un nuevo router como parte del controlador que se encargará de la gestión de los Usuario, como nos indica el guion. Para el login, el proceso es el siguiente:
    - **1. Autenticación**: El usuario envía su nombre de usuario y contraseña. Si son correctos, el servidor genera un **JWT** y lo envía al cliente como una **cookie HTTP-only**, que no puede ser accesible por JavaScript para mayor seguridad.
    - **2. Middleware de autenticación**: Cada vez que el cliente realiza una solicitud, el servidor verifica la cookie `access_token` con `jwt.verify()`. Si el token es válido, extrae la información del usuario y la agrega a la solicitud (`req.username`).
- **Importancia del middleware**: El middleware asegura que solo los usuarios con un **token válido** puedan acceder a rutas protegidas, validando la autenticación en cada solicitud.
- Para poder realizar esta práctica necesitamos dos bibliotecas:

    - 1. Una **cookie** es un pequeño archivo de texto que almacena información en el navegador del usuario para mantener el estado y la persistencia entre visitas a un sitio web. Para gestionar todas las cookies en la práctica empleamos cookie-parser. Comando de instalación: **npm i cookie-parser**.
    - 2. Para gestionar los token JWT empleamos jsonwebtoken. Comando de instalación: **npm i jsonwebtoken**.

3.2. Práctica 3.2

- Esta práctica es bastante simple, y añadimos a autorización a determinados usuarios.
- Dado que la BD es NoSQL podemos añadir campos libremente a cada usuario. En esta práctica añadiremos el campo admin a algunos usuarios, actualizando el esquema que tenemos hecho con moongose para obtener los datos de la BD.
- Tenemos que realizar unas modificaciones en el código, incluyendo el campo admin en el middleware de autentificación en el servidor y en el router de usuarios a la hora del login.
- Los usuarios que inicien sesión como admin podrán realizar cambios en la página del producto, pudiendo cambiar título y precio.
- Se establecerán validaciones en el esquema, como bien indica el guion. Se define un método post con el método “findByIdAndUpdate” para obtener el producto actualizado y recargar la página con el nuevo producto.

4. Práctica 4

- En esta práctica vamos a hacer un API para modificar el rating de cada producto desde SPAs o móviles.
- Habilitamos los sencillos endpoints que indica el guion. Para ello, debemos crear un nuevo router en el controlador que se encargará de la gestión de estos endpoints sobre los ratings.
- Para el testeo del API, empleamos la extensión de VS Code “Rest Client”.
- Como punto opcional, definimos un logger empleando windows, redirigiendo la salida de cada proceso registrado a un fichero combined.log.

5. Práctica 5

5.1. Práctica 5.1

- En esta práctica vamos a definir un DOM que hará llamadas fetch al API creado en la práctica anterior.

- Analizando el código que nos proporciona el guion, tenemos que añadir la visualización de las estrellas de cada producto. Basándonos en la documentación, nos redirecciona a una librería que podemos emplear para pintar las estrellas.
- Definiremos un método que genere las estrellas y, partiendo del rating obtenido con la llamada fetch al API, se generarán y mostrarán las correspondientes estrellas.

5.2. Práctica 5.2

- En esta práctica se mejora el resultado de la práctica anterior permitiendo hacer click en las estrellas y votar cada producto.
- Para ello, tenemos que basarnos en el PUT del API, que refleja una orden UPDATE.
- Debemos definir un método que nos permita votar. Para ello, definimos una método agregar manejador, que será el que nos permita hacer click en cada una de las estrellas. Al hacer click, se lanzará el método Vota, definido parcialmente en el guion. Este método lanzará una llamada fetch al API actualizando el rating del producto.
- Además, cambiará el color de las estrellas en la página en función de la puntuación actualizada del producto.

6. Práctica 6

- En esta práctica hacemos un proyecto con Vite + React.
- Usamos la librería UI React, Tailwind como CSS y daisyUI como librería de componentes sobre Tailwind.
- Para obtener los datos se usa la librería SWR.