Práctica 1. Desarrollo de una API REST para la gestión de una clínica de fisioterapia con Express y Mongoose

Parte 3. Autenticación y pruebas completas

En esta parte de la práctica añadiremos un modelo para gestionar los usuarios registrados en nuestra aplicación. Para ello, utilizaremos los mecanismos de autenticación basada en tokens y roles de acceso para controlar las operaciones permitidas para cada tipo de usuario. Finalmente, se realizará una batería de pruebas para comprobar el correcto funcionamiento de la aplicación.

Instala *jsonwebtoken* en la aplicación y crea una carpeta *auth* donde definiremos un archivo con todas las funciones necesarias para la gestión de la autenticación (generar token, validar token, etc.).

Nota: se valorará el uso de variables de entorno para gestionar información sensible, empleando la librería *dotenv*.

1. Modelo de usuario

En el archivo *models/users.js* se registrará la información de los **usuarios** autorizados en la aplicación. De ellos guardaremos estos campos (todos obligatorios):

- **login:** string, mínimo de 4 caracteres.
- **password:** string, mínimo 7 caracteres.
- **rol:** define el tipo de usuario (*admin*, *physio* o *patient*), que tendrá diferentes permisos en la aplicación.

2. Autenticación

La gestión de la autenticación la dividiremos en tres pasos:

2.1 Funciones auxiliares de autenticación

Define en la carpeta **auth** un archivo llamado *auth.js* que contenga las funciones necesarias para gestionar la autenticación basada en tokens. Este archivo debe incluir al menos las siguientes funciones:

- **Función para generar un token:** crea un token JWT para un usuario válido.
- **Función para verificar el token:** comprueba si el token es válido y decodifica los datos del usuario.
- **Función para proteger rutas:** restringe el acceso solo a usuarios con un token válido y verifica el rol para permitir el acceso a rutas específicas.
    - Si el usuario no tiene permisos para acceder al recurso solicitado, independientemente de las credenciales, debe enviar un código 403

(Acceso prohibido), junto al mensaje “Acceso no autorizado” (campo error)

2.2 Enrutador para usuarios

Este enrutador lo ubicaremos en el fichero *routes/auth.js* y responderá a URIs que comiencen por ***/auth***. Se pide implementar los siguientes servicios:

- **Loguear a un usuario**
  - **Endpoint:** `POST /auth/login`
  - **Descripción:** se obtendrá del cuerpo de la petición el *login* y el *password* del usuario y, si es correcto, se devoverá un código 200 con el token de acceso correspondiente (campo result). En caso contrario se enviará un código de error 401 (no autorizado) con el mensaje “login incorrecto” (campo error).

2.3 Protección de rutas y gestión de roles de acceso

Debe protegerse cada ruta según el rol:

- **admin:** acceso completo a toda la API.
- **physio:** permisos para gestionar pacientes y expedientes.
- **patient:** acceso restringido a sus propios datos únicamente.

A continuación se detalla una tabla resumen de accesos por rol:

Endpoint	Roles		
	admin	physio	patient
PATIENTS			
`GET /patients`	SÍ	SÍ	❌
`GET /patients/:id`	Sí	Sí	Sí (solo su propio ID)
`GET /patients/find`	Sí	Sí	❌
`POST /patients`	Sí	Sí	❌
`PUT /patients/:id`	Sí	Sí	❌
`DELETE /patients/:id`	Sí	Sí	❌
PHYSIOS			
`GET /physios`	Sí	Sí	Sí
`GET /physios/:id`	Sí	Sí	Sí
`GET /physios/find`	Sí	Sí	Sí
`POST /physios`	Sí	❌	❌
`PUT /physios/:id`	Sí	❌	❌
`DELETE /physios/:id`	Sí	❌	❌
RECORDS			
`GET /records`	Sí	Sí	❌
`GET /records/:id`	Sí	Sí	Sí (solo su propio ID)
`GET /records/find`	Sí	Sí	❌

`POST /records`	Sí	Sí	❌
`POST /records/:id/appointments`	Sí	Sí	❌
`DELETE /records/:id`	Sí	Sí	❌

3. Pruebas

La aplicación se probará con distintos casos de prueba de los diferentes servicios. Se os proporcionarña una base (simple) de ejemplos de prueba que podéis usar en vuestro desarrollo aunque posteriormente se probará con más casos.

4. Criterios de calificación

La calificación final de la práctica se realizará según estos criterios:

1. Estructura general de la aplicación (carpetas *models*, *routes* y *auth* con el contenido adecuado, fichero *package.json* con las dependencias correctamente instaladas y fichero *index.js* con el programa principal): **0,25 puntos**.

   Se incluye en este apartado la gestión de un fichero de configuración de variables de entorno .env que, al menos, almacene y utilice correctamente las siguientes variables de entorno:

   - Puerto de conexión
   - URL de acceso a la base de datos
   - Palabra secreta para cifrar los tokens
2. Definición de modelos y esquemas (parte 1): **2,25 puntos**, repartidos como sigue:
   - Esquema y modelo de paciente: **0,5 puntos**
   - Esquema y modelo de fisioterapeuta: **0,5 puntos**
   - Esquema y modelo de expediente médico (incluyendo relación con paciente y un array de subdocumentos de citas médicas) **1,25 puntos**
3. Enrutadores (parte 2): **4,5 puntos**. Se calificará cada servicio con **0,25 puntos**.
4. Autenticación (parte 3): **1,75 puntos**, repartidos de este modo:
   - Enrutador para usuarios: **0,25 puntos**
   - Funciones auxiliares de autenticación correctas: **0,75 puntos**
   - Protección de los servicios necesarios con el middleware de autenticación: **0,75 puntos**
5. Servidor principal: **0,25 puntos** (secuenciación correcta de los pasos para poner en marcha el servidor)
6. Pruebas: **1 puntos**. Se probará la aplicación con el conjunto simple de pruebas que se os facilitará en esta etapa, más otras pruebas adicionales que se añadirán.

Penalizaciones y requisitos a tener en cuenta para superar la práctica

Será condición indispensable para superar la práctica alcanzar al menos la mitad de puntuación en los apartados 2, 3 y 6 de los criterios anteriores. Además, en caso de que el profesor/a considere oportuno realizar una prueba oral a cualquier alumno/a sobre su práctica, será necesario responder correctamente al menos a la mitad de las preguntas que el profesor pueda hacer.

Además, se aplicarán las siguientes **penalizaciones** sobre la nota final:

- Si algún servicio no recibe o devuelve los atributos con el nombre que se indica, o no responde a la URI indicada, se calificará con 0 puntos, sin tener en cuenta si su código es o no correcto.
    - **Ejemplo 1:** si el servicio POST /patients espera recibir el atributo name y recibe un atributo nombre en su lugar, se calificará con un 0.
    - **Ejemplo 2:** si el servicio GET /patients devuelve la respuesta en un campo resultado en lugar de result, el servicio se calificará con un 0.
    - **Ejemplo 3:** si el servicio de borrado (DELETE) atiende a la URI patients/delete/:id en lugar de /patients/:id, también se calificará con un 0.
- Si no se sigue la estructura de proyecto propuesta (nombres de archivos y carpetas) y no se justifica debidamente esta decisión, se penalizará la calificación global de la práctica hasta un 50% de la nota.