Práctica 2. Desarrollo de una aplicación web para la gestión de ligas de baloncesto con Express y Mongoose

Parte 2. Integración de vistas dinámicas y formularios con Nunjucks

En esta última parte de la práctica se transformará la API REST desarrollada en las partes anteriores en una aplicación web completa, incorporando **vistas HTML dinámicas, formularios y navegación entre páginas**, utilizando el motor de plantillas **Nunjucks**.

El objetivo es que la aplicación permita consultar y gestionar jugadores, equipos y partidos desde una interfaz web, reutilizando los servicios REST ya implementados.

2.1 Consideraciones generales

- Se mantiene el sistema de autenticación y gestión de roles desarrollado en la práctica anterior (admin, manager, user).
- Todas las vistas, formularios y rutas web deberán respetar las **mismas reglas de acceso** definidas para la API REST.
- Los servicios REST existentes deberán **adaptarse para su uso desde formularios web**:
    - Los formularios enviarán los datos mediante peticiones POST.
    - En caso de éxito, los controladores **redirigirán a una vista HTML**.
    - En caso de error, se deberá:
        - Renderizar la vista `error.njk`, o
        - Volver a renderizar el formulario mostrando mensajes de error claros.
- Recuerda que los formularios HTML **no permiten el uso directo de los métodos PUT ni DELETE**. Por eso, todas las operaciones de edición y eliminación se implementarán mediante POST, aunque conceptualmente correspondan a PUT o DELETE.

2.2 Pasos previos

Como punto de partida, realiza una copia del proyecto actual y aplica los siguientes cambios iniciales:

- Instala, carga y configura Nunjucks como motor de plantillas, configurándolo para buscar las vistas en una carpeta `/views` situada en la raíz del proyecto.
- Instala Bootstrap y configúralo para servir sus estilos desde la carpeta `/dist`.
- Configura un middleware para servir archivos estáticos desde una carpeta `/public`, que deberá incluir:
    - Un archivo **index.html**, que actuará como página de bienvenida de la aplicación.

    - Una subcarpeta `/images` para las imágenes del proyecto.
    - Opcionalmente, una subcarpeta `/css` para estilos propios.

- Añade una ruta en `/` que redirija a la página inicial (`/public/index.html`).

El archivo `package.json` deberá incluir, al menos, las siguientes dependencias:

- express
- mongoose
- nunjucks
- bootstrap
- dotenv

2.3 Jerarquía de vistas

Se deben definir las siguientes plantillas base:

- `base.njk` : incluirá título dinámico de la página, enlace a los estilos de Bootstrap y un bloque para el contenido principal.
- `menu.njk` :
    - Contendrá el menú de navegación principal de la aplicación
    - Debe incluir enlaces a listado de jugadores, listado de equipos, listado de partidos y formularios de alta (según se indica más adelante)

`menu.njk` debe incluirse dentro de `base.njk` , de modo que el menú esté presente en todas las vistas.

2.4 Vistas a definir

Todas las vistas deben heredar de `base.njk` .

2.4.1 Vista de error

`error.njk` mostrará un mensaje de error recibido desde el controlador. Se utilizará para errores de validación, búsquedas sin resultados o errores internos.

2.4.2 Jugadores (players)

- `players_list.njk` :
    - Mostrará el listado completo de jugadores registrados
    - Cada jugador incluirá enlaces para:
        - Ver detalles
        - Editar
        - Eliminar

- `player_detail.njk`:
  - Mostrará toda la información de un jugador concreto

2.4.3 Equipos (teams)

- `teams_list.njk`:
  - Mostrará el listado completo de equipos registrados
  - Cada equipo incluirá enlaces para:
    - Ver detalles
    - Eliminar
- `team_detail.njk`:
  - Mostrará la informacion completa del equipo y su roster activo

2.4.4 Partidos (matches)

- `matches_list.njk`: mostrará el listado de todos los partidos registrados.
- `match_detail.njk`: mostrará la información completa de un partido concreto.

2.5 Definición y validación de formularios

En esta parte se deberán implementar los siguientes formularios, incluyendo tanto la vista como la lógica de servidor necesaria para su funcionamiento.

Además, se deberán validar los datos enviados desde los formularios utilizando los mecanismos de validación definidos en los esquemas de Mongoose.

Cuando se produzca un error al guardar o actualizar un documento:

- Se deberán recoger los errores de validación generados por Mongoose
- Se deberán mostrar en la vista del formulario (encima de cada control) o bien en la vista de `error.njk`.
- Se mostrará un mensaje por cada error de validación detectado
- Se recomienda recorrer el objeto `error.errors` devuelto por Mongoose para construir el mensaje de error.

2.5.1 Alta de jugadores

- `player_add.njk`
  - Formulario para registrar nuevos jugadores.
  - Debe incluir todos los campos del modelo Player (excepto el ID).
- `GET /players/new`: renderiza el formulario de alta de jugadores.

- `POST /players` : procesa los datos enviados desde el formulario.
    - Si el alta es correcta, redirige al listado de jugadores.
    - Si hay errores (datos inválidos o duplicados), vuelve a renderizar el formulario mostrando mensajes de error específicos.

Añade un enlace en el **menú principal** para acceder a este formulario.

2.5.2 Edición de jugadores

- `player_edit.njk` : permite modificar los datos de un jugador existente. El formulario se cargará con los datos actuales del jugador.
- `GET /players/:id/edit` : renderiza el formulario de edición de jugadores.
- `PUT /players/:id` : procesa los datos enviados desde el formulario.
    - Si la actualización es correcta, redirige a la vista de detalle del jugador.
    - Si ocurre algún error, se vuelve a mostrar el formulario con los mensajes correspondientes.

El acceso a este formulario debe estar disponible:

- Desde el listado de jugadores.
- Desde la vista de detalle del jugador.

2.5.3 Búsqueda de jugadores por nombre

- Formulario de búsqueda por nombre parcial o completo.
- `GET /players/find?name=...` :
    - Si no se encuentran jugadores, se renderiza la vista de error con un mensaje personalizado.

El buscador deberá estar disponible:

- En el menú principal.
- En el listado de jugadores.

2.5.4 Alta de equipos

- `team_add.njk` : formulario para registrar nuevos equipos.
- `GET /teams/new` : renderiza el formulario de alta de equipos.
- `POST /teams` : procesa los datos enviados desde el formulario.
    - En caso de éxito, redirige al listado de equipos.
    - En caso de error, vuelve a mostrar el formulario con mensajes específicos.

2.5.5 Gestión del roster de un equipo

- `team_add_player.njk` : formulario para añadir un jugador existente al roster de un equipo. Permitirá seleccionar el jugador y la fecha de incorporación.
- `GET /teams/:id/roster/new` : renderiza el formulario para añadir un jugador a un equipo.
- `POST /teams/:id/roster` : procesa los datos enviados desde el formulario
  - Si la operación es correcta, redirige a la vista de detalle del equipo.
  - Si hay errores, se muestran mensajes adecuados.

2.5.6 Alta de partidos

`match_add.njk` : formulario para registrar un nuevo partido. Debe incluir todos los campos obligatorios del modelo Match.

- `GET /matches/new` : renderiza el formulario de alta de partidos.
- `POST /matches` : procesa los datos enviados desde el formulario
  - En caso de éxito, redirige al listado de partidos.
  - En caso de error, vuelve al formulario con los mensajes correspondientes.

Añade un enlace en el **menú principal** para este formulario.

2.5.7 Subida de imagen en jugadores (players)

Se añadirá la posibilidad de asociar una imagen a cada jugador (por ejemplo, una fotografía o avatar). Para ello, será necesario modificar el **modelo Player**.

En el archivo `models/player.js` se deberá añadir una nueva propiedad para almacenar el nombre del archivo de la imagen subida:

- image: de tipo String, no obligatoria

Formularios afectados

Se deberán modificar los formularios `player_add.njk` y `player_edit.njk` para incluir un campo de tipo file llamado image.

Los formularios deberán:

- Incluir el atributo `enctype="multipart/form-data"`
- Permitir subir una única imagen por jugador
- Seguir funcionando correctamente aunque no se seleccione ninguna imagen

Gestión de la subida de archivos

Para gestionar la subida de imágenes se deberá utilizar la **librería multer** y configurarla en el archivo del enrutador correspondiente (`routes/players.js`), definiendo:

- La carpeta de destino: `public/uploads`
- El nombre del archivo: concatenando un timestamp (`Date.now()`) con el nombre original del archivo, para evitar sobreescrituras.

Servicios afectados

Los servicios que gestionan la creación y edición de jugadores deberán integrar el middleware de subida de archivos:

- `POST /players`
- `POST /players/:id/edit`

Estos servicios deberán:

- Procesar la imagen subida, si existe
- Guardar el nombre del archivo en la propiedad image del jugador
- Mantener el funcionamiento normal del servicio si no se sube ninguna imagen

En el caso de edición, conservar la imagen anterior si no se proporciona una nueva

Visualización de la imagen

La vista `player_detail.njk` deberá mostrar la imagen del jugador desde la ruta `/public/uploads` , si existe. Si no tiene imagen, no se mostrará nada.

2.6 Autenticación. Formulario de login

Se debe implementar un formulario de autenticación que permita a los usuarios acceder a la aplicación web utilizando el sistema de autenticación basado en tokens desarrollado en la práctica anterior.

2.6.1 Vista de login

- `login.njk` : formulario con los campos login y password.
- `GET /login` : renderiza el formulario de login.
- `POST /login` : procesa las credenciales enviadas desde el formulario.
    - Si el login es correcto:
        - Se genera el token JWT utilizando la lógica ya implementada.
        - El usuario queda autenticado en la aplicación web.
        - Se redirige a la página principal de la aplicación.
    - Si las credenciales no son válidas:

- Se vuelve a renderizar el formulario de login, mostrando un mensaje de error claro (por ejemplo: "Login incorrecto").

2.6.2 Acceso a la aplicación

Todas las vistas de la aplicación, excepto la página de login, deben requerir autenticación. Si un usuario no autenticado intenta acceder a cualquier otra ruta, será redirigido automáticamente a la vista de login.

El menú principal debe mostrar:

- Enlace a Login cuando el usuario no esté autenticado.
- Información básica del usuario autenticado (login y rol).
- Opcionalmente, un enlace de Logout.

De forma opcional, se podrá implementar una ruta `/logout` que elimine el token de autenticación y redirija al formulario de login.

ANEXO I. Persistencia del token JWT en la aplicación web (cookie httpOnly)

Para que la autenticación funcione correctamente en una aplicación web, no basta con generar el token JWT en el login: es necesario que el navegador pueda enviarlo automáticamente en las siguientes peticiones. La forma más sencilla en una web con formularios es almacenar el token en una **cookie httpOnly**.

En este anexo se describe qué cambios deben realizarse y en qué archivos, de forma clara y guiada.

1. ¿Qué es una cookie httpOnly?

Una cookie httpOnly es una cookie que:

- Se guarda en el navegador.
- Se envía automáticamente al servidor en cada petición.
- No puede ser leída ni modificada desde JavaScript (mejora de seguridad).

En esta práctica, no debes gestionar manualmente el token en el frontend (ni usar localStorage, ni añadir headers con JavaScript) sino que el token se gestionará en el servidor mediante cookies.

2. Detalles de implementación

Añadir dependencia extra

- Instala el middleware **cookie-parser** para leer cookies: `npm install cookie-parser`
- En `index.js`, añade el siguiente código:

```
const cookieParser = require('cookie-parser');
app.use(cookieParser());
```

- Este middleware debe cargarse **antes de definir las rutas**.

Guardar el token en la cookie (POST /login)

En el servicio que procesa el login web (`POST /login`), una vez que las credenciales son correctas y se ha generado el JWT, se debe guardar el token en una cookie.

```
// Archivo routes/auth.js
res.cookie('token', token, {
  httpOnly: true,
  sameSite: 'lax'
});
res.redirect('/players');
```

Con esto, el navegador almacenará el token y lo enviará automáticamente en las siguientes peticiones.

Leer el token desde la cookie (middleware)

El middleware de autenticación debe obtener el token desde la cookie:

```
// Archivo: auth/auth.js
const token = req.cookies.token;
```

Para mantener compatibilidad con herramientas como Postman o EchoAPI, es recomendable aceptar también el token enviado por cabecera:

```
const token =
  req.cookies.token ||
  req.headers.authorization?.split(' ')[1];
```

A partir de este token, se validará el JWT y se obtendrá la información del usuario (login y rol) como ya se hacía.

Rutas de login y logout

En el mismo archivo `routes/auth.js` se definirán también las rutas web:

- `GET /login` : renderiza `login.njk`

- `POST /login`: procesa el formulario de login
- `GET /logout`: elimina la cookie y redirige a `/login`

Para cerrar sesión basta con eliminar la cookie del token:

```
res.clearCookie('token');
res.redirect('/login');
```

Proteger las rutas (archivos routes/*.js)

Todas las rutas definidas en los siguientes archivos deberán requerir autenticación:

- `routes/players.js`
- `routes/teams.js`
- `routes/matches.js`

La protección se realizará utilizando el middleware de autenticación definido en `auth.js`.

Las únicas rutas públicas serán:

- `GET /` (página de bienvenida)
- `GET /login` y `POST /login`

Si un usuario no autenticado intenta acceder a cualquier otra ruta, será redirigido automáticamente a la vista de login.

3. Criterios de calificación

- **Configuración inicial y estructura del proyecto: 1 punto**
  - Integración correcta de Nunjucks (views, motor configurado): 0,4
  - Configuración de estáticos (`/public`, Bootstrap): 0,4
  - Redirección correcta desde `/` a `index.html`: 0,2
  - Si no arranca o no renderiza vistas: 0 en este bloque.
- **Plantillas base y navegación: 1 punto**
  - `base.njk` bien definida (bloques, Bootstrap): 0,4
  - `menu.njk` integrado en todas las vistas: 0,3
  - Menú dinámico según autenticación (login / usuario / rol): 0,3
- **Vistas de listado y detalle: 1,5 puntos**
  - Players (list + detail): 0,6
  - Teams (list + detail + roster): 0,6
  - Matches (list + detail): 0,3

    - Se valorará renderizado correcto y navegación funcional entre vistas

- **Formularios y lógica de servidor: 2 puntos**
    - Alta y edición de jugadores: 0,8
    - Alta de equipos y gestión de roster: 0,8
    - Alta de partidos: 0,4
    - Incluye rutas GET/POST correctas, redirecciones en éxito, mensajes de error

- **Subida de imágenes en jugadores: 1,5 puntos**
    - Campo image añadido al *modelo Player*: 0,25
    - Formularios con *multipart/form-data* y campo *file*: 0,5
    - Configuración correcta de multer (`public/uploads` , nombre de archivo): 0,5
    - Visualización de la imagen en `player_detail.njk` : 0,25

- **Validación de formularios con Mongoose: 1,5 puntos**
    - Validadores definidos en esquemas de Mongoose: 0,75
    - Mensajes de error personalizados en los esquemas: 0,5
    - Gestión y renderizado correcto de errores (*error.errors* → `error.njk`): 0,25

- **Autenticación, roles y protección de rutas: 1 punto**
    - Login funcional con JWT: 0,4
    - Persistencia del token en cookie httpOnly: 0,3
    - Protección correcta de rutas + redirecciones a `/login` : 0,3
    - Si la app no protege rutas, suspenso directo de este bloque.

- **Gestión de errores y restricciones de negocio: 0,5 puntos**
    - Uso correcto de `error.njk`
    - Mensajes claros en: duplicados, restricciones de negocio y accesos no autorizados

- **Código, orden y coherencia general: - 0,5 puntos si no se cumple**
    - Estructura clara
    - Código legible
    - Sin “restos” de JSON/API antigua