



7-3-2024

Node Js

Práctica 2: Frameworks



Julián Blanco González

UNIVERSIDAD DE EXTREMADURA

ÍNDICE

1. ¿Qué es Node JS?	3
2. ¿Por qué usar Node?	4
3. Instalación de Node Js	5
3.1. Página oficial	5
3.2. Administrador de Versiones (recomendada)	6
3.3. Node Package Manager (NPM)	7
4. Callbacks vs Promesas	7
4.1. Callbacks	7
4.2. Promesas	7
4.3. Diferencias entre ellos	8
4.4. Ejemplo de código	8
5. Páginas y proyectos que utilizan Node	9
6. Proyecto hecho con Node	10
6.1. Explicación del proyecto	10
6.2. Estructura del proyecto	11
6.3. Dependencias utilizadas	11
6.4. Capturas del proyecto	18
7. Bibliografía	22

ÍNDICE DE FIGURAS

Figura 1: Logo de Node Js	3
Figura 2: MERN (Mongo, Express, React, Node)	4
Figura 3: Importar dependencias/ archivos externos en Node	5
Figura 4: página oficial de Node Js	5
Figura 5: FNM	6
Figura 6: Node Package Manager (NPM)	7
Figura 7: Código con callbacks	8
Figura 8: Código con Promesas	9
Figura 9: Linked in (Node)	9
Figura 10: Netflix (Node)	10
Figura 11: Paypal (Node)	10
Figura 12: Dependencias del proyecto NBA	12
Figura 13: Uso de Bcrypt	12
Figura 14: Archivo .env (dotenv)	13
Figura 15: tabla dinámica de equipos con ejs	13
Figura 16: página inicial (express)	15
Figura 17: Creación del token	15
Figura 18: función para verificar un token	16
Figura 19: modelo de equipo (mongoose)	17
Figura 20: preset de las imágenes de jugadores (multer)	18
Figura 21: Login (pública)	18
Figura 22: registro (pública)	19
Figura 23: inicio (privada)	19
Figura 24: editar jugador (privada)	20
Figura 25: equipos (privada)	21
Figura 26: error no autorizado	21

1. ¿QUÉ ES NODE JS?

“Node.js® es un entorno de ejecución de JavaScript multiplataforma de código abierto.”, ésta es la descripción que te da la página web oficial.



Figura 1: Logo de Node Js

Ahora, para resumir, Node JS es un intérprete de JavaScript que se ejecuta en servidor, es decir, es un entorno donde puedes ejecutar el código de JavaScript. Está basado en el motor de JavaScript que utiliza Google Chrome (V8), escrito en C++. Sus características principales son:

- Tener el mismo lenguaje en cliente y servidor
 - Permite a cualquier persona desarrollar en backend o en frontend
 - Permite reusar código o incluso mover código de cliente a servidor o al revés
- Está orientado a eventos y utiliza un modelo asíncrono (propio de JavaScript).
- Al contrario que en el navegador, encontramos muchas llamadas asíncronas:
 - Llamadas a APIs
 - Lectura y escritura de ficheros
 - Ejecución de cálculos en el servidor
- Motor v8 y multiplataforma
 - Motor JavaScript de chrome, navegador donde se ejecuta muy rápido JavaScript.
 - Se ejecuta en windows, MacOS, TV, Nintendo Switch...

- Es monohilo
 - Utiliza un solo procesador

En resumen, al ser monohilo y estar programado a eventos, Node Js está en bucle esperando peticiones (eventos), pero mientras está ejecutando tareas, de aquí que sea asíncrono.

2. ¿POR QUÉ USAR NODE?

Hay muchas razones por las que usar Node. Voy a poner a continuación las más importantes:

- Demanda del Mercado: todas las empresas conocidas utilizan NodeJs, como Netflix, Visual Studio, Amazon



Figura 2: MERN (Mongo, Express, React, Node)

- Todos los conocimientos previos de JavaScript te sirven en Node, es decir, teniendo esta base, se puede ser productivo rápidamente.
- Se pueden crear APIs, páginas web, servidores...
- Comunidad gigantesca con el ecosistema más grande del mundo que es NPM.
- Es rápido, escalable, fácil de desplegar y barato, incluso gratis a poco nivel.

Y, una propia, añadida por mí, es que, se parece a Java en algunos aspectos:

Importar librerías o clases ajenas = usar dependencias o usar archivos externos.

```

const express = require('express');
const router = express.Router();
const multer = require('multer');
const fs = require('fs');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');
const verificarToken = require('./auth');

const Jugador = require('../model/jugador');
const Equipo = require('../model/equipo');
const Usuario = require('../model/usuario');

```

Figura 3: Importar dependencias/ archivos externos en Node

Y, por ejemplo, para exportar el archivo externo de equipo.js (model), se usa:

```
module.exports = mongoose.model('equipo', equipoSchema);
```

3. INSTALACIÓN DE NODE JS

Para instalar Node Js, hay 2 posibilidades

3.1. PÁGINA OFICIAL

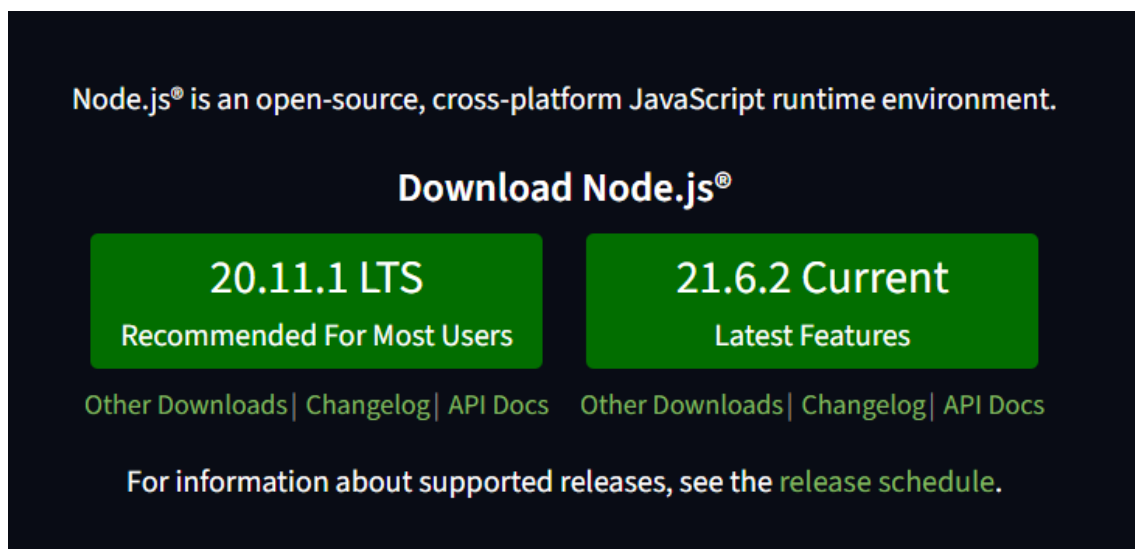


Figura 4: página oficial de Node Js

LTS (Long Term Support): versión estable de Node Js.

Current: Tiene todas las características de Node Js, pero puede dar errores.

3.2. ADMINISTRADOR DE VERSIONES (RECOMENDADA)

Esta es la mejor opción, porque, si estás trabajando en diferentes proyectos, y tienen distinta versión de Node, la forma anterior no valdría.

Para ello, se utiliza un administrador de versiones de Node. En este caso, he utilizado fnm.



Figura 5: FNM

La descripción de su página oficial es: “Fast and simple Node.js version manager, built in Rust”. Los pasos por seguir son:

Instalar fnm:

```
> scoop install fnm
```

Configurar fnm:

```
> fnm env --use-on-cd | Out-String | Invoke-Expression
```

Instalar cualquier versión de Node (en este caso, la de la imagen 4, la LTS):

```
> Fnm install 20.11.1
```

Usar esa versión de Node:

```
> Fnm use 20.11.1
```

Comprobar que se utiliza la versión de node escogida en fnm:

```
> Node -version
```

3.3. NODE PACKAGE MANAGER (NPM)



Figura 6: Node Package Manager (NPM)

Manejador de paquetes de node, es la herramienta por defecto de JavaScript para la tarea de compartir e instalar paquetes. Se compone de 2 partes:

- Un repositorio online para publicar paquetes de software libre para ser utilizados en proyectos Node.js
- Una herramienta para la terminal (command line utility) para interactuar con dicho repositorio que te ayuda a la instalación de utilidades, manejo de dependencias y la publicación de paquetes.

4. CALLBACKS VS PROMESAS

4.1. CALLBACKS

Un callback es una función que se pasa como argumento a otra función y se ejecuta después de que esa función ha completado su tarea. Los callbacks son una forma común de gestionar operaciones asíncronas en JavaScript.

4.2. PROMESAS

Una promesa es un objeto que representa el resultado de una operación asíncrona, que puede ser una resolución exitosa o un error. Las promesas introducen una sintaxis más limpia y legible para manejar operaciones asíncronas en comparación con los callbacks anidados.

4.3. DIFERENCIAS ENTRE ELLOS

Manejo de errores

En los callbacks, los errores a menudo se manejan mediante la comprobación de una variable de error en la función de callback o mediante la propagación de excepciones. Esto puede llevar a un código complicado y propenso a errores.

En cambio, las promesas proporcionan una forma más estructurada de manejar errores. Se puede usar el método `try catch()` para manejar cualquier error que ocurra dentro de la promesa.

Legibilidad del código

Las promesas suelen dar como resultado un código más limpio y fácil de leer, especialmente cuando se tienen múltiples operaciones asíncronas que deben realizarse en secuencia o en paralelo.

Async/await

Es una característica más reciente de JavaScript que simplifica aún más el manejo de operaciones asíncronas. Permite escribir código asíncronico de manera más similar a código síncrono, lo que mejora la legibilidad.

4.4. EJEMPLO DE CÓDIGO

```
//Callbacks: funciones que se ejecutan cuando una tarea termina
console.log("Leyendo el primer archivo...");
fs.readFile('./archivo.txt', 'utf-8', (err, text) => {
  console.log("Primer archivo: ", text)
});

console.log("Haciendo cosas mientras lee el archivo...");

console.log("Leyendo el segundo archivo...");
fs.readFile('./archivo2.txt', 'utf-8', (err, text) => {
  console.log("Segundo archivo: ", text)
});
```

Figura 7: Código con callbacks

```

const fs = require('node:fs/promises');
console.log("Leyendo el primer archivo...");
fs.readFile('./archivo.txt', 'utf-8')
  .then(text => {
    console.log("Primer texto: ", text);
  })

console.log("Haciendo cosas mientras lee el archivo...");

console.log("Leyendo el segundo archivo...");
fs.readFile('./archivo2.txt', 'utf-8')
  .then(text => {
    console.log("Segundo texto: ", text);
  })

```

Figura 8: Código con Promesas

5. PÁGINAS Y PROYECTOS QUE UTILIZAN NODE

Linkedin: comenzó a utilizar Node.js solo para su aplicación móvil y luego migró todo el código base a él. También registraron una velocidad de la aplicación 20 veces mayor que su iteración anterior con Ruby on Rails.



Figura 9: Linked in (Node)

Netflix: construyeron su interfaz de usuario (UI) con Node.js por su modularidad. Netflix experimentó un tiempo de inicio dos veces más rápido al utilizar Node.js.



Figura 10: Netflix (Node)

Paypal: Node.js ha proporcionado a PayPal un tiempo de carga de la aplicación web más rápido, que es casi el doble y contiene un 33% menos de líneas de código y un 40% menos de archivos que su aplicación inicial basada en Java.

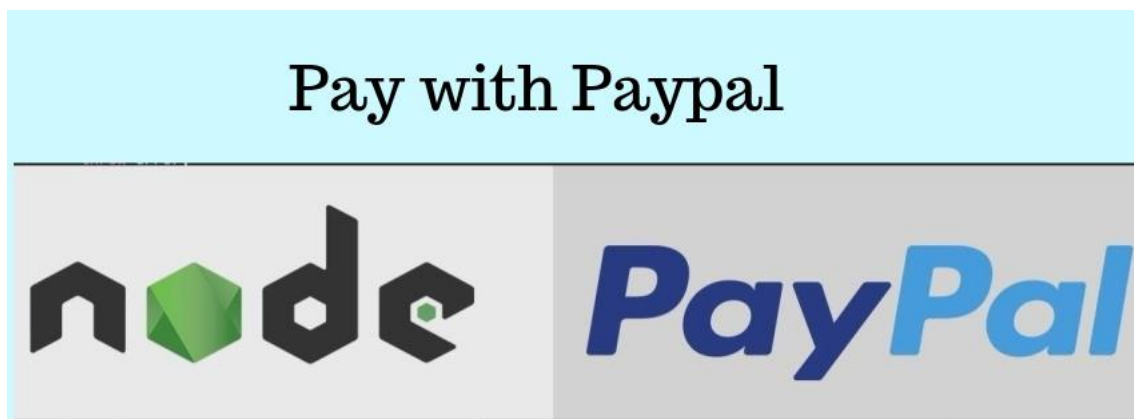


Figura 11: Paypal (Node)

6. PROYECTO HECHO CON NODE

6.1. EXPLICACIÓN DEL PROYECTO

El objetivo es realizar una página web que sea capaz de hacer las operaciones CRUD a una colección de jugadores (Mongo DB) a través de todas las consultas. También he implementado un sistema de registro y login con seguridad, el cual, no te deja acceder a todas las páginas que solo se pueden acceder cuando te logueas.

6.2. ESTRUCTURA DEL PROYECTO

El proyecto sigue el patrón de MVC (modelo, vista, controlador).

Modelo: clases base que tienen los campos de cada colección de la base de datos.

Vista: páginas web con extensión .ejs (dinámicas).

Controlador: funcionalidad de los eventos de la página web (botones, envío de formularios...) y redirección de cada página web a otra.

6.3. DEPENDENCIAS UTILIZADAS

Las dependencias son las librerías que se implementan con Node para conseguir nuevas funcionalidades, mejorar la salida por consola, hacer más eficiente el código..., es decir, un conjunto de módulos para resolver un problema mayor.

Hay 3 tipos:

- Locales: obligatorias para el proyecto.
- Desarrollador: no son obligatorias para el proyecto. Ayudan a optimizar el código, agregar estilos a salida por consola...
- Globales: disponibles para todos los proyectos en tu ordenador.

Para instalar las dependencias, hay que usar el siguiente comando:

```
> npm install [nombreDependencia]
```

Para ver todas las dependencias instaladas, hay que ver el archivo "package.json":

```

"dependencies": {
  "bcrypt": "^5.1.1",
  "dotenv": "^16.4.5",
  "ejs": "^3.1.9",
  "express": "^4.18.2",
  "express-session": "^1.18.0",
  "jsonwebtoken": "^9.0.2",
  "mongoose": "^8.2.0",
  "multer": "^1.4.5-lts.1"
},
"devDependencies": {
  "nodemon": "^3.1.0"
}

```

Figura 12: Dependencias del proyecto NBA

Dependencias locales

BCRYPT

Librería en Node.js que proporciona funciones para el cifrado seguro de contraseñas. En términos simples, ayuda a proteger las contraseñas de los usuarios en tu aplicación. La principal ventaja de utilizar bcrypt radica en su capacidad para «hashear» contraseñas de una manera segura y eficiente.

Proyecto: proteger las contraseñas a la hora de guardarlas en Mongo DB.

```

_id: ObjectId('65e6fdedd674a73a1d597f10')
nombre: "d"
password: "$2b$10$6sF3oXcfohNgHuvU2h0t5eJY2aDU9.9Ah3e7IdNDBCPST3CJghHJa"
__v: 0

```

```

_id: ObjectId('65e7026319a8104927c4243e')
nombre: "Julian"
password: "$2b$10$/kw4vZwe/6yjlz2wNhKqdunAkAx000Fd0/fb1BQbJ0kTQTqsEuHhu"
__v: 0

```

Figura 13: Uso de Bcrypt

DOTENV

Cargar variables de entorno desde un archivo .env y las asigna a una variable global llamada process.env. Su función principal es proteger información confidencial, como contraseñas, claves de API y credenciales de bases de datos.

Proyecto: guardar el puerto del servidor, la url de la base de datos y la clave secreta (autenticación por Token)

```
PORT = 5000
DB_URI = mongodb://localhost:27017/NodeProyecto
SECRET_KEY=dacsfdsfvef3f893gf45nfg59fn45v|
```

Figura 14: Archivo .env (dotenv)

EJS

Motor de plantillas que permite incrustar código JavaScript en archivos HTML o vistas. Permite crear páginas web dinámicas al mezclar HTML estático con lógica de programación en JavaScript.

Proyecto: todas las páginas que usan o manipulan los datos de Mongo uso ejs, y para reducir mucho código, tengo un header y un footer común para todas las páginas privadas.

Incluir el header y el footer en cualquier página:

```
<%- include('layout/header') %>
```

```
<%- include('layout/footer') %>
```

Tabla dinámica con ejs:

```
<tbody class="table-group-divider">
  <% equipos.forEach((row, index)=> { %>
    <tr>
      <!-- <td><%= index %></td> -->
      <style>td img {height: 100px !important;object-fit: contain;}</style>
      <td></td>
      <td><%= row.nombre %></td>
      <td><%= row.ciudad %></td>
      <td><%= row.estadio %></td>
    </tr>
  <% }) %>
</tbody>
```

Figura 15: tabla dinámica de equipos con ejs

EXPRESS

Express es un framework web minimalista y flexible para Node.js que proporciona un conjunto sólido de características para aplicaciones web y móviles.

1. Manejo de Rutas y Verbos HTTP:

- Express permite escribir **manejadores de peticiones** para diferentes **verbos HTTP** en diferentes **rutas URL**.
- Puedes definir rutas para manejar solicitudes GET, POST, PUT, DELETE, etc.

2. Middleware:

- Express utiliza una arquitectura basada en **middleware**.
- Puedes agregar funciones de middleware para procesar solicitudes antes de que lleguen a los manejadores finales.
- Esto es útil para tareas como autenticación, registro de acceso, compresión, etc.

3. APIs y Aplicaciones Web:

- Con miles de **métodos de utilidad HTTP** y middleware disponibles, crear una **API sólida** es rápido y sencillo.
- Express proporciona una **capa delgada de características fundamentales para aplicaciones web**, sin ocultar las funcionalidades de Node.js que ya conoces y amas.

4. Popularidad y Base para Otros Frameworks:

- Express es el **framework web más popular de Node.js** y sirve como base para muchos otros frameworks web populares en Node.
- Es ampliamente utilizado en la comunidad de desarrollo web

Hay 2 maneras de redireccionar con express:

Render: Combina un template (vista) con datos (contexto) y devuelve una página HTML completa.

Redirect: Redirige al cliente a otra ruta o URL.

Proyecto: para todas las redirecciones de mi página web, utilizo express, ya sea, para cargar la vista dinámica de equipos(render), o para cargar la pantalla de login, cuando el usuario se registre (redirect)

```
//pagina de inicio (login)
router.get('/', async (req, res) => {
  res.render('login', {
    titulo: "Login",
  });
});
```

Figura 16: página inicial (express)

EXPRESS-SESSION

Módulo que se utiliza con Express para manejar sesiones en aplicaciones web. Permite almacenar datos de sesión en el lado del servidor y admite varios atributos y opciones de cookies.

Proyecto: enviar al cliente los datos de los jugadores, equipos, y almacenar el token de login de sesión.

JSONWEBTOKEN

Permite crear, firmar y verificar tokens JWT. Estos tokens son útiles para autenticación y autorización en aplicaciones web y APIs. Sus funcionalidades principales:

Generación de JWT: Puedes crear un JWT con información específica (como el ID del usuario) y firmarlo con una clave secreta.

Verificación de JWT: Puedes verificar la validez de un token recibido en una solicitud.

Proyecto: crear un token cada vez que se loguea el usuario y destruirlo cuando se desloguee. Gracias a esto, no se pueden acceder a las rutas privadas poniendo su url.

```
const payload = { nombre: req.body.nombre };
const secretkey = process.env.SECRET_KEY;
const options = { expiresIn: '1h' };

const token = jwt.sign(payload, secretkey, options);
req.session.token = token; //guardo el token en la sesion
req.session.message = {
  type: 'success',
  message: `Bienvenido, ${req.body.nombre}!`,
};
res.redirect("/inicio")
```

Figura 17: Creación del token


```

const jwt = require('jsonwebtoken');

function verificarToken(req, res, next) {
  const token = req.session.token;
  //console.log("Contenido cabecera: ",token)
  if (!token) {
    return res.status(403).json({ message: 'Token no proporcionado' });
  }
  jwt.verify(token, process.env.SECRET_KEY, (err, decoded) => {
    if (err) {
      return res.status(401).json({ message: 'Token inválido' });
    }
    req.usuario = decoded;
    next(); // Pasar al siguiente middleware si la verificación es exitosa
  });
}

module.exports = verificarToken;

```

Figura 18: función para verificar un token

MONGOOSE

herramienta de modelado de objetos para MongoDB. Diseñada para trabajar en un entorno asíncrono, facilita la creación, manipulación y validación de documentos MongoDB. Sus funciones principales son:

Conexión a MongoDB: Puedes establecer una conexión a la base de datos MongoDB utilizando `mongoose.connect`.

Definición de modelos: Mongoose permite crear modelos de datos basados en esquemas.

Interacción con documentos: Puedes realizar operaciones CRUD (crear, leer, actualizar, eliminar) en documentos MongoDB utilizando los modelos definidos.

Proyecto: conexión a la base de datos de Mongo, operación CRUD y creación de los modelos de cada colección.

```

const mongoose = require('mongoose');
const equipoSchema = new mongoose.Schema({

  nombre: {
    type: String,
    required: true,
  },
  escudo: {
    type: String,
    required: true,
  },
  ciudad: {
    type: String,
    required: true,
  },
  estadio: {
    type: String,
    required: true,
  },
},
  { collection: 'equipo' });

module.exports = mongoose.model('equipo', equipoSchema);

```

Figura 19: modelo de equipo (mongoose)

MULTER

Su función principal es la carga de archivos: Multer facilita la carga de archivos en aplicaciones Express. También permite el manejo de formularios multipartes: Permite procesar formularios que contienen campos de texto y archivos adjuntos.

Proyecto: a la hora de editar, subir o borrar un jugador, uno de sus campos, es una imagen. Para poder manipular la imagen, hace falta el multer, es decir, si se borra el jugador, debe borrarse la imagen, si se actualiza, debe borrarse la imagen anterior y guardar la nueva.

```
const jugadorStorage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, './uploads/jugadores');
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + '-' + file.originalname);
  }
});
const jugadorUpload = multer({ storage: jugadorStorage });
```

Figura 20: preset de las imágenes de jugadores (multer)

Dependencias de desarrollador

NODEMON

Automatizar el reinicio de la aplicación Node automáticamente cuando detecta cambios en los archivos del directorio. Esto mejora significativamente la productividad del desarrollo al evitar la necesidad de reiniciar el servidor manualmente.

Proyecto: ver los cambios directamente sin tener que reiniciar el servidor.

6.4. CAPTURAS DEL PROYECTO

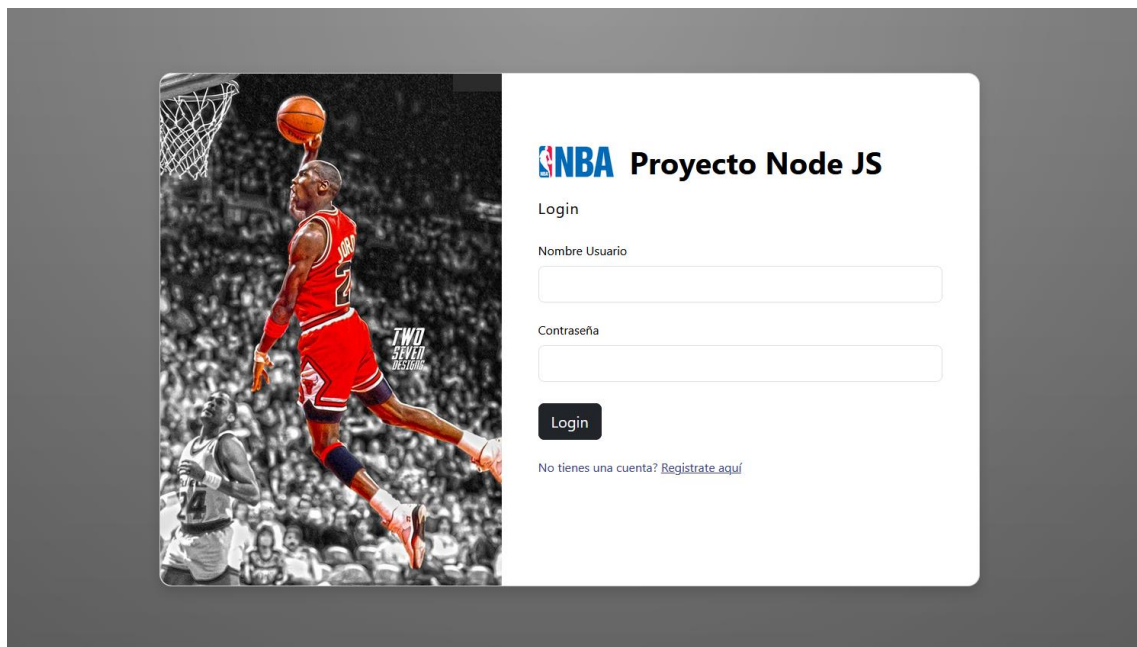


Figura 21: Login (pública)

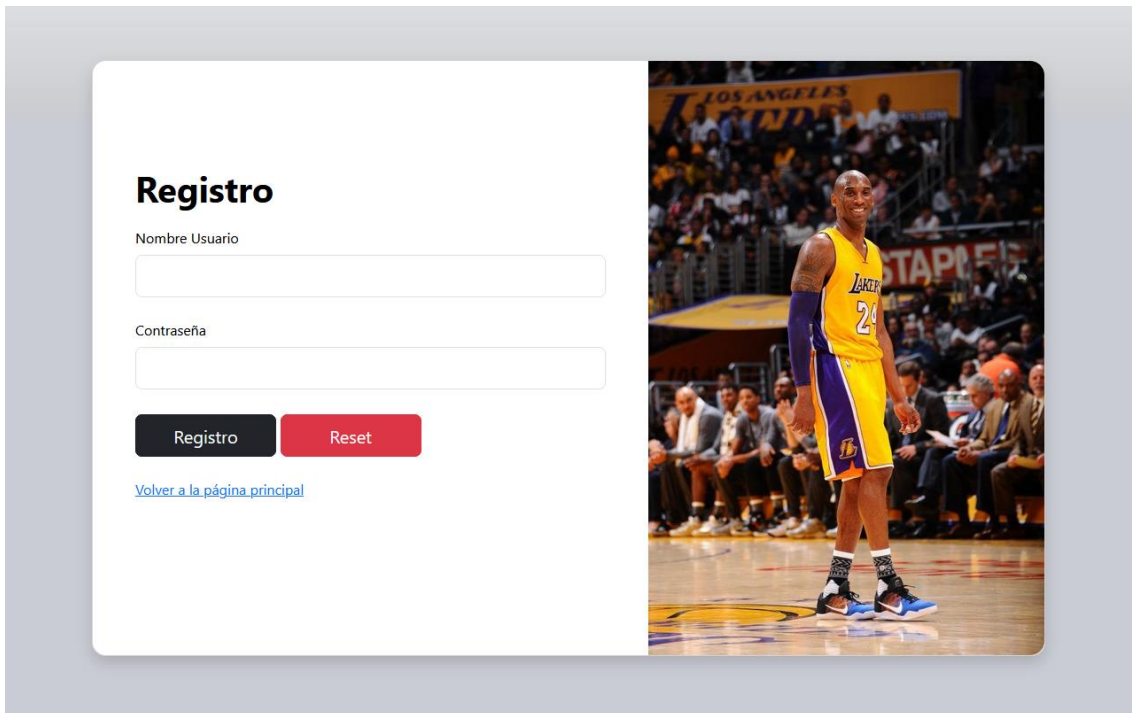


Figura 22: registro (pública)

<div> <div> Inicio Añadir Jugador Equipos </div> <div>Perfil</div> </div>						
ID	Imagen	Nombre	Equipo	Dorsal	Posición	Acción
0		Don Stephen	Golden State Warriors	30	Base	
1		Le Bron James	Los Angeles lakers	8	AlaPivot	
2		Luka Doncic	Dallas Mavericks	77	Alero	
3		Kyrie Irving	Dallas Mavericks	2	Base	
4		Jason Tatum	Boston Celtics	0	Alero	
5		Zach Lavine	Chicago Bulls	8	Escolta	

Figura 23: inicio (privada)

Editar Jugador (Don Stephen)

Nombre del Jugador

Don Stephen

Equipo

Golden State Warriors

Posición

Base


Dorsal

30

Seleccionar Imagen

Examinar...

No se ha seleccionado ningún archivo.



Actualizar

Restablecer

Figura 24: editar jugador (privada)

EQUIPOS NBA

Escudo	Nombre	Ciudad	Estadio
	Los Angeles lakers	Los Angeles, California	Crypto.com Arena
	Dallas Mavericks	Dallas, Texas	American Airlines Express
	Golden State Warriors	San Francisco, California	Chase Center
	Boston Celtics	Boston, Massachuttes	TD Garden
	Chicago Bulls	Chicago, Illinois	United Center

Figura 25: equipos (privada)

Si se intenta acceder a una ruta privada poniendo su url, gracias a jsonwebtoken, te muestra un mensaje de que no se puede acceder.

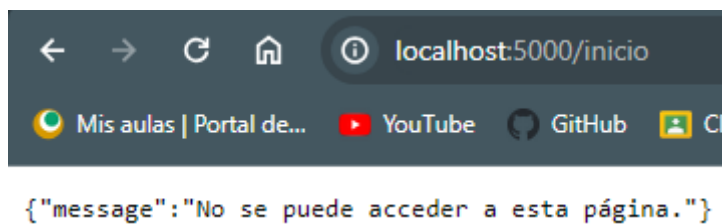


Figura 26: error no autorizado

7. BIBLIOGRAFÍA

Introducción a Nodejs

- <https://juanda.gitbooks.io/webapps/content/javascript/node.html>
- <https://learn.microsoft.com/es-es/training/modules/intro-to-nodejs/2-what>
- https://www.youtube.com/watch?v=yB4n_K7dZV8

FNM

- <https://github.com/Schniz/fnm>

NPM

- <https://www.freecodecamp.org/espanol/news/que-es-npm/>

Empresas que usan Node

- <https://kinsta.com/es/blog/node-js-aplicaciones/>

Callbacks / Promesas

- <https://www.freecodecamp.org/espanol/news/javascript-asincrono-explicacion-de-callbacks-promesas-y-async-await/>
- <https://keepcoding.io/blog/diferencias-entre-callbacks-y-promesas/>

Dependencias

- [Encriptación de password en NodeJS y MongoDB: bcrypt \(izertis.com\)](#)
- [dotenv - npm \(npmjs.com\)](#)
- [Cómo usar EJS para crear una plantilla de su aplicación Node | DigitalOcean](#)
- [Express - Node.js web application framework \(expressjs.com\)](#)
- [express-session - npm \(npmjs.com\)](#)
- [jsonwebtoken - npm \(npmjs.com\)](#)

- [mongoose - npm \(npmjs.com\)](#)
- [¿Qué es Multer en Express y Node.js? | KeepCoding Bootcamps](#)
- [nodemon - npm \(npmjs.com\)](#)