

La Pluma Digital

Marcos Arjona Comino

Índice:

1. Introducción
 - 1.1. Resumen del proyecto
 - 1.2. Explicación de la aplicación
 - 1.3. Tecnologías usadas
2. Requisitos
 - 2.1. Requisitos funcionales
 - 2.2. Requisitos no funcionales
3. Diagramas y Casos
 - 3.1. Casos de uso
 - 3.2. Diagrama entidad-relación
 - 3.3. Diagrama de clases del modelo
 - 3.4. Diagrama de secuencia
4. Implementaciones, Tecnologías y Código
5. Manual de usuario
6. Conclusiones

1. Introducción

1.1. Resumen del proyecto

La pluma digital consiste en una red social dedicada a los lectores y escritores. Hablando con personas lectoras, pude detectar la falta de una red social para personas lectoras que quieran debatir acerca de libros, o autores. Así como, la falta de una forma para conocer gente con sus mismos gustos en cuanto a lectura se refiere.

De ahí nace la idea de La Pluma Digital, encontrar un sitio donde poder conversar, conocer y descubrir cosas sobre cualquier libro.



1.2. Explicación de la aplicación

Esta aplicación será utilizada para ver comentarios de personas que quieran conocer a más personas ya que podrán seguir a otras e incluso conocer su correo electrónico si ambas se siguen. Así como tener escritores favoritos y poder ver los libros que han escrito. Otra cosa que los usuarios podrán hacer es ver los comentarios que hagan las personas sobre cualquier libro sobre sus libros preferidos o cualquiera que quieran ver.

Por otro lado también podrán ver los ranking de los libros más populares de la red social en cuanto a número de comentarios. Y por supuesto, podrán ver y buscar los libros que están en la base de datos para saber si existe ese libro, para conocer más de él o lo que quieran.

1.3. Tecnologías usadas

Para este proyecto usaré NodeJs para el backend, y Pug como herramienta para el frontend. He decidido usar NodeJs como backend porque es una herramienta de uso poco común que se basa en el lenguaje JavaScript. Este lenguaje, no tipado, presenta una prueba para determinar la lógica de programación de la

persona que lo programa. También he aprovechado la formación que he tenido en la empresa de prácticas (Nter tech Services)



2. Requisitos

2.1. Requisitos funcionales

Autenticación y Autorización

- El sistema permite el registro de nuevos usuarios.
- El sistema permite el inicio y cierre de sesión.
- El sistema valida el token JWT para proteger rutas privadas.
- El sistema muestra contenido dinámico según el rol del usuario (usuario, modder, admin).

Gestión de Libros y Autores

- El sistema lista todos los libros disponibles.
- El sistema lista todos los autores.
- El sistema permite subir libros (solo usuarios registrados).
- El sistema permite visualizar detalles de un libro o autor.

Ranking

- El sistema genera rankings de libros en función de cuántas veces han sido mencionados.
- El sistema genera rankings de autores del mismo modo.
- El sistema muestra rankings en formato de tabla, incluyendo nombre, foto y posición.

Perfil de Usuario

- El sistema permite a los usuarios ver su ranking de publicaciones.
- El sistema permite que los usuarios vean sus publicaciones.
- El sistema permite al usuario ver su descripción.

Publicaciones

- El sistema permite a los usuarios publicar contenido relacionado con libros/autores.
- Las publicaciones están ligadas a libros y autores (vía idLibro e idAutor).

2.2. Requisitos no funcionales

Seguridad

- El sistema cifra los tokens JWT con una clave secreta segura.
- Las contraseñas se almacenan cifradas (por ejemplo, usando bcrypt).
- Las cookies tienen opciones httpOnly y secure (en producción).

Rendimiento

- Las consultas de ranking no ser eficientes, usando agregaciones SQL y ventanas (RANK()).
- Las rutas protegidas no evitar accesos innecesarios a la base de datos si el token no es válido.

Usabilidad

- El sistema tiene una interfaz clara, con navegación sencilla entre libros, autores y rankings.
- El encabezado cambia dinámicamente según si el usuario está logueado o no (por ejemplo, "Iniciar sesión" ↔ "Cerrar sesión").

Mantenibilidad

- El sistema estructura su código en controladores, rutas, middlewares y vistas organizadas.
- Las vistas están basadas en plantillas reutilizables (por ejemplo, usando layout.pug).


Compatibilidad

- El sistema funciona en navegadores modernos (Chrome, Firefox, Edge).

3. Diagramas y Casos

3.1. Casos de uso

- Login y registro:

 Login y registro

- Libros:

 libro

- Ranking:

 ranking

- Autor:

 autor

- Publicaciones:

 publicaciones

- Sube tu libro:

 subetulibro

3.2. Diagramas Entidad-Relacion

 DiagramaEntidad Relacion

3.3. Diagrama de clases

 Clases

3.4. Diagrama de secuencia

- Registro:

 registro secuencia

- Login:

 login secuencia

- Libro

 libro secuencia

- Autores

 autores secuencia

- Publicaciones:



4. Implementaciones, Tecnologías y Código

El proyecto sigue una arquitectura MVC (Modelo-Vista-Controlador), separando de forma clara la lógica de negocio, la gestión de vistas y el acceso a datos:

- **Modelos:** Definidos de forma implícita en base a las tablas de la base de datos (Autores, Libros, Usuarios, etc.).
- **Vistas:** Usando **Pug** como motor de plantillas para generar HTML dinámico.
- **Controladores:** Manejando la lógica de las rutas y operaciones con la base de datos mediante **Node.js** y **MySQL**.

En cuanto a tecnologías, he usado algunas como:

Tecnología	Uso Principal
Node.js	Backend y servidor principal
Express.js	Framework para la gestión de rutas y middleware
MySQL	Base de datos relacional para persistencia
Pug	Motor de plantillas para vistas
Moment.js	Manejo y formato de fechas
JWT	Autenticación y autorización de usuarios
bcryptjs	Encriptación de contraseñas

En cuanto a la explicación de código, voy a explicar un tipo de archivo por cada categoría:

- **Controller:**

```
const db = require('../db.js');
const moment = require('moment');

exports.ranking = (req, res) => {
  db.query(
    'SELECT RANK() OVER (ORDER BY COUNT(p.idLibro) DESC) as ranking,
    COUNT(*) as contador, l.Titulo, l.Foto, p.idLibro as idLibro FROM
    Publicaciones p JOIN Libros l ON p.idLibro = l.idLibro GROUP BY p.idLibro,
    l.Titulo, l.Foto ORDER BY COUNT(p.idLibro) DESC'
    , (err, rank)=> {
      if (err) {
        return res.send('Fallo a la hora del ranking' + err)
      }
      if (rank.length === 0) {
        return res.send('Fallo en la respuesta')
      }
      db.query(
```

```

        'SELECT RANK() OVER (ORDER BY COUNT(p.idAutor) DESC) as
        ranking, COUNT(*) as contador, a.Nombre, a.Foto, p.idAutor as idAutor FROM
        Publicaciones p JOIN Autores a ON a.idAutor = p.idAutor GROUP BY p.idAutor,
        a.Nombre, a.Foto ORDER BY COUNT(p.idAutor) DESC',
        (err, rankA) => {
            if(err){
                return res.send('Fallo a la hora del ranking autores' +
err)
            }
            if(rankA.length === 0) {
                return res.send('Fallo en la respuesta')
            }
            res.render('ranking/ranking', {rankings: rank, rankA:
rankA})
        }
    )
}
}
}

```

En este ejemplo de código podemos ver como la estructura de este controller se basa en los resultados de un aquery, buscando obtenerlos, transformarlos y responder según lo que obtenga. Busco conseguir un contador de publicaciones.

- **Router:**

```

const express = require('express');
const router = express.Router();
const rankingController = require('../controllers/rankingController');

router.get('/', rankingController.ranking)

module.exports = router;

```

En este caso lo que hacer el Router es indicar como tiene que responder el proyecto basandore en algunas rutas.

- **View:**

```

extends ../templates/layout
block head

block content
    div.centro
        div.tamano
            h1 Registrare con nosotros
            form(action="/register", method="post")
                label(for="UserName") Username:
                input(type="text", name="UserName")

```

```
br  
br  
label(for="Nombre") Nombre:  
input(type="text", name="Nombre")  
br  
br  
label(for="Descripcion") Descripcion:  
input(type="text", name="Descripcion")  
br  
br  
label(for="Email") Email:  
input(type="text", name="Email")  
br  
br  
label(for="Contraseña") Contraseña:  
input(type="password", name="Contraseña")  
br  
br  
label(for="Foto") Foto (URL):  
input(type="text", name="Foto")  
br  
br  
input.enlace-registrar(type="submit", value="Registrate!!")  
br  
br  
a.enlace-enviar(href='/login') ¿Ya tienes cuenta? Logueate aquí!!
```

Esta es una vista creada para poder registrarse.

5. Manual de usuario

Para usar la web. Puedes entrar directamente a ver las publicaciones que hay, y a posteriorer si quieres hacer algo más, tendrás que loguearte, aunque si no tienes cuenta, puedes registrarte ingresando tus datos. Posteriormente, cuando te logueas puedes llegar a acceder a los rankings o al listado de autores, libros o quizas si tu quieres, puedes mandar tus libros, que ya hayan sido publicados. O saber un poco más de los creadores.

6. Conclusiones

En conclusión, este proyecto me ha llevado a poder jugar un poco más con javascript y aumentar bastante mi destreza usando este tipo de lenguaje. También es una enseñanza de paciencia, tranquilidad y esfuerzo a largo plazo.

Empecé sin saber muy bien ni que quería hacer ni como quería hacerlo. Por lo que, solo las tardes después del trabajo donde intentaba buscar como hacer cosas y cuando mi pareja hacía de cliente y me corregía la estética, podemos saber como **La Pluma Digital** se ha ido forjando junto a esfuerzo y dedicación continua.

Este proyecto, que como objetivo tiene que los lectores puedan tener unaa forma más de conocer personas, nace en busqueda también de enseñar un mundo algo perdido y oculto que se va perdiendo poco a poco, la escritura.

