



# Manual del administrador

Jose Antonio Gutiérrez Marcos

I.E.S. PORTADA ALTA

## Contenido

Introducción .....	2
Objetivos del proyecto .....	3
Planificación del proyecto .....	4
Análisis y diseño del sistema .....	5
Especificaciones del sistema .....	6
Especificaciones del software .....	8
Código fuente relevante.....	11
Conclusiones del proyecto .....	14
Bibliografía .....	15

## Introducción

MiColeccionCF es una aplicación web desarrollada bajo el objetivo de proporcionar una herramienta accesible, cómoda e intuitiva para los aficionados a las colecciones de cromos de La Liga EA Sports y La Liga Hypermotion de la temporada 2024/25.

Esta idea viene de la escasez que hay en el mercado de servicios de este tipo, siendo un nicho poco explotado y que los servicios que existen o son poco conocidos o están anticuados, además de que esta aplicación busca ser fácil de utilizar para aquellos aficionados que no están familiarizados con las tecnologías ya que este nicho lo conforman en una buena parte aficionados de una edad avanzada.

Lo que distingue a esta aplicación de otras que hay en el mercado es además de la interfaz sencilla e intuitiva, es que contará con chats en tiempo real y sugerencias de usuarios por cercanía, fomentando así que se contacten entre ellos para hablar de posibles intercambios.

## Objetivos del proyecto

El objetivo final de este proyecto es proporcionar una aplicación web que como he dicho, sea fácil de utilizar y que permita a los usuarios llevar al día sus colecciones de las máximas categorías del fútbol español y facilitar el intercambio con otros aficionados.

Los objetivos principales serían:

- Montar un sistema de registro/inicio de sesión mediante autenticación con JWT.
- Tener almacenados todos los datos sobre los usuarios y sus colecciones en una base de datos MySQL.
- Mantener una interfaz intuitiva para el usuario.
- Mostrar una lista de recomendaciones de usuarios para fomentar el contacto entre ellos.
- Establecer un sistema de mensajería en tiempo real de chats privados entre usuarios.

## Planificación del proyecto

En cuanto a los participantes del proyecto, el único soy yo.

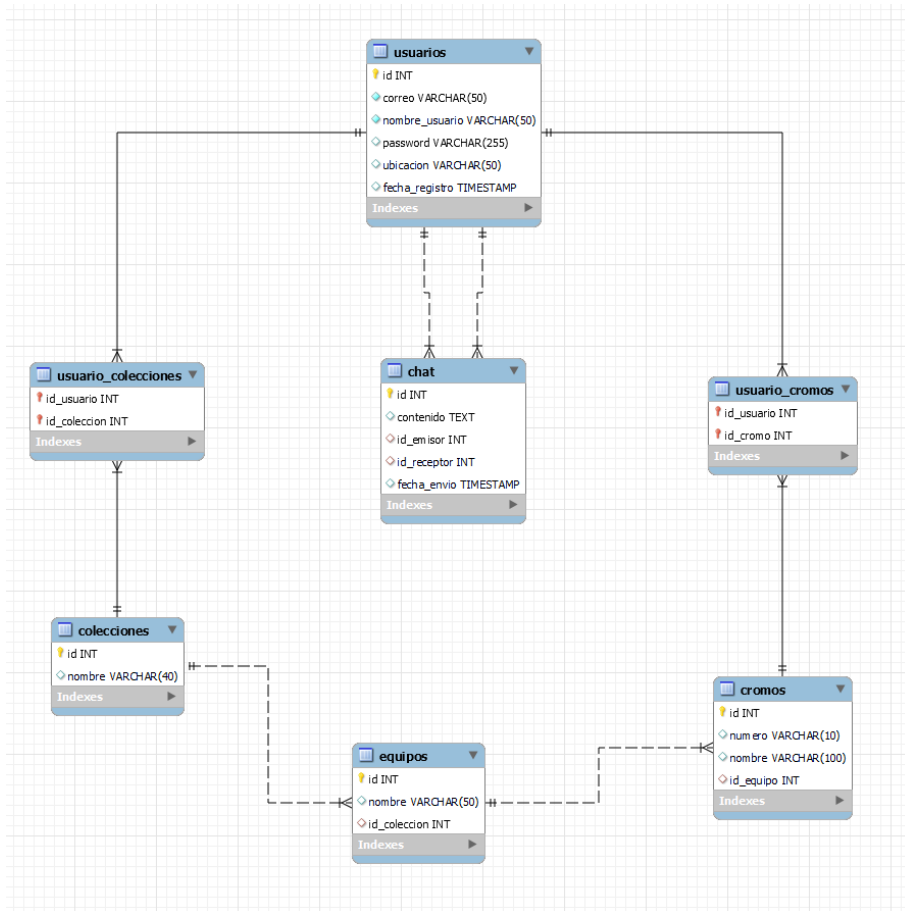
Tareas:

- 1. Planteamiento inicial de la aplicación (3 horas).**
  - a. Diseño de la base de datos (2 horas).
  - b. Calcular tecnologías a usar (1 hora).
- 2. Desarrollo del backend (24 horas).**
  - a. Creación y planteamiento de las rutas del api (6 horas).
  - b. Implementar el sistema de autenticación con JWT y cookies (4 horas).
  - c. Funcionalidad del sistema de mensajería en tiempo real (5 horas).
  - d. Modelos para hacer las consultas a la base de datos necesarias (6 horas).
  - e. Testing (3 horas).
- 3. Desarrollo del frontend (33 horas).**
  - a. Diseño y funcionalidad del sistema de registro/inicio de sesión (6 horas).
  - b. Diseño y funcionalidad de la página de inicio con las colecciones del usuario (8 horas).
  - c. Diseño y funcionalidad de la barra lateral de usuarios recomendados (2 horas).
  - d. Visualización del perfil del usuario al que se acceda desde las recomendaciones (3 horas).
  - e. Diseño y funcionalidad del sistema de mensajería en tiempo real (7 horas).
  - f. Creación de los servicios que se comunicarán con el backend. (3 horas)
  - g. Testing (4 horas).

El resultado final sería tener una aplicación web con un sistema funcional de registro e inicio de sesión autenticado con JWT, en la que se pueda gestionar las dos colecciones y con un chat privado entre usuarios.

## Análisis y diseño del sistema

Diagrama Entidad-Relación:



En la carpeta adjunto el script de la creación e inserción de algunos datos de la base de datos.

## Especificaciones del sistema

### Justificación de las tecnologías:

- Angular 17: Es un framework que hemos visto bastante en clase y que es una buena opción para construir el frontend y recibir los datos del backend.
- Node.js + Express: Al igual que angular lo hemos visto ya en clase y me parece una gran opción para crear el servidor web y generar los datos de la API REST, también con muchas opciones de librerías para añadir.
- Socket.io: Es la librería que utilizaré para que haya comunicación en tiempo real entre el cliente y el servidor ya que lo hemos visto también previamente en clase y será lo que me de la funcionalidad para el sistema de mensajería en tiempo real.
- MySQL: También lo hemos cursado y me parece la más cómoda para crear la base de datos relacional que necesita este proyecto.

### Instalación y configuración:

- Node.js se instala mediante el repositorio oficial de Ubuntu y Angular mediante npm.
- MySQL instalado localmente también desde el repositorio oficial.
- Librerías instaladas:
  - Bcrypt: sirve para hashear la contraseña y guardarla así en la base de datos.
  - Cookie-parser: sirve como middleware para que express lea las cookies que hay en las peticiones HTTP y dejarlas como un objeto en req.cookies.
  - Cors: lo utilizo como otro middleware para que el backend acepte las peticiones del frontend, que está en otro dominio.
  - Dotenv: para almacenar algunos datos importantes en un fichero .env y utilizarlos en el backend cuando sea necesario.
  - Express: es el framework que utiliza el backend para crear la API REST que enviará los datos necesarios al frontend.
  - Http: utilizaré esta librería en la creación del servidor y la integración con socket.io.
  - Jsonwebtoken: esta librería es para poder utilizar la autenticación con JWT a la hora de iniciar sesión con un usuario y guardar sus datos de sesión de forma segura.
  - MySQL2: esta librería la necesito para poder conectar el backend con la base de datos MySQL.
  - Socket.io: con socket.io, podré implementar el sistema de mensajería en tiempo real utilizando web sockets.
  - Jest: es el framework que utilizaré para los tests del backend.
  - Supertest: se utiliza junto con jest para simular las peticiones HTTP sin tener que tener el servidor ejecutándose.
  - Bootstrap: es el framework que he utilizado para el estilo de la interfaz de la aplicación.
  - Jasmine: es el framework que he utilizado para los tests del frontend.
  - Karma: ejecuta los tests en navegadores reales para probar como se comporta la aplicación en ellos.

### Especificaciones del hardware:

El proyecto estará desplegado en un VPS de OVHcloud, que tiene las siguientes características:

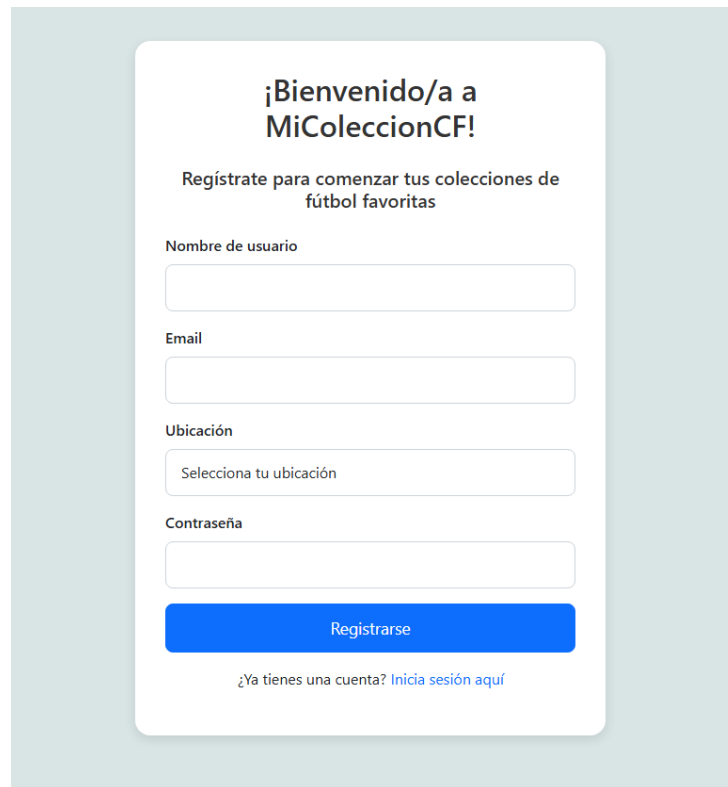
- Arquitectura del procesador: x86\_64.
- Modelo del procesador: Intel Core Processor (Haswell, no TSX).
- Memoria RAM: 2GB.
- Espacio de almacenamiento: 40GB.



## Especificaciones del software

Las operaciones que se llevarán a cabo en la aplicación serán las siguientes:

- Registro del usuario
  - Actor: nuevo usuario.
  - Resultado: se crea una cuenta nueva y se redirige al inicio de sesión.
  - Parámetros: nombre de usuario, correo, contraseña y ubicación.



¡Bienvenido/a a  
MiColeccionCF!

Regístrate para comenzar tus colecciones de fútbol favoritas

Nombre de usuario

Email

Ubicación

Contraseña

Registrarse

¿Ya tienes una cuenta? [Inicia sesión aquí](#)

- Inicio de sesión
  - Actor: usuario existente.
  - Resultado: accede a su página de inicio.
  - Parámetros: nombre de usuario y contraseña.



¡Bienvenido/a a  
MiColeccionCF!

Inicia sesión para continuar con tus colecciones favoritas

Nombre de usuario

Contraseña

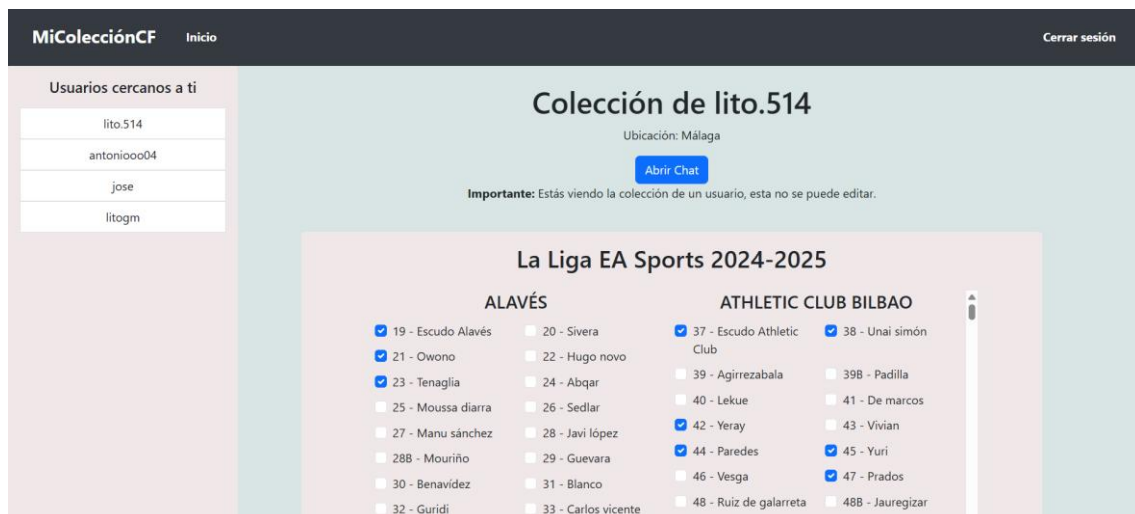
Iniciar sesión

¿No tienes una cuenta? [Regístrate aquí](#)

- Gestión de las colecciones
  - Actor: usuario que ha iniciado sesión.
  - Resultado: el usuario añade o borra cromos de sus colecciones.
  - Parámetros: id del cromo y de la colección en cuestión y el id del usuario.



- Acceder al perfil de otro usuario
  - Actor: el usuario que ha iniciado sesión.
  - Resultado: visualiza la información y colecciones de un usuario.
  - Parámetros: nombre del usuario que se está visualizando.



- Chatear con un usuario
  - Actor: el usuario que ha iniciado sesión.
  - Resultado: el usuario envía un mensaje a otro usuario.
  - Parámetros: contenido del mensaje, id del usuario que envía el mensaje y el id del usuario que lo recibe.



## Código fuente relevante

Como código que yo considero relevante incluiré el siguiente:

En este código que hay en el backend, creo un middleware que verifica el token de JWT para evitar los accesos que no están autorizados en el inicio de sesión de la aplicación.

```
app.use((req, res, next) => {
  const token = req.cookies.access_token;
  req.session = { user: null};
  try {
    const data = jwt.verify(token, process.env.JWT_SECRET);
    req.session.user = data;
  } catch {}

  next();
});
```

Aquí creo una ruta que sirve para confirmar dicho acceso en el frontend.

```
app.get("/check", (req, res) => {
  if(req.session.user) {
    return res.json({ autenticado: true, usuario: req.session.user });
  }
  return res.status(401).json({ autenticado: false });
});
```

Esto es un “guard” que utilizo para admitir el acceso en el inicio de sesión dentro de las rutas de angular.

```
export const autenticacionGuard: CanActivateFn = () => {
  const http = inject(HttpClient);
  const router = inject(Router);

  return http.get('http://localhost:3000/check', { withCredentials: true })
    .pipe(
      map(() => true),
      catchError(() => {
        router.navigate(['/login']);
        return of(false);
      })
    );
};
```

Aquí muestro como lo aplico en las rutas:

```
const routes: Routes = [
```

```

{ path: 'login', component: LoginComponent },
{ path: '', redirectTo: '/login', pathMatch: 'full' },
{ path: 'register', component: RegisterComponent },
{ path: 'home', component: HomeComponent, canActivate:
[autenticacionGuard] },
{ path: 'users/:nombre_usuario', component: PerfilComponent,
canActivate: [autenticacionGuard] },
{ path: 'chat/:nombre_usuario', component: ChatComponent, canActivate:
[autenticacionGuard] }
];

```

En este código establezco la conexión de socket.io con el frontend desde el backend, creando “salas” uniendo los id del emisor y del receptor (ordenados, para que el chat entre el usuario 1 y el usuario 2 siempre sea el mismo).

```

io.on("connection", (socket) => {
  socket.on("abrirChat", ({ id_emisor, id_receptor }) => {
    const nombreChat = [id_emisor, id_receptor].sort().join("-");
    socket.join(nombreChat);
  });

  socket.on("enviarMensaje", async (mensaje) => {
    const nombreChat = [mensaje.id_emisor,
mensaje.id_receptor].sort().join("-");
    io.to(nombreChat).emit("recibirMensaje", mensaje);
    try {
      await Chat.guardarMensaje(mensaje.id_emisor,
mensaje.id_receptor, mensaje.contenido);
    } catch (error) {
      console.error("error:", error);
    }
  })
});

```

Y en el cliente lo conecto de la siguiente manera:

```

conectarSocket(): void {
  this.socket = this.chatService.conectar();

  this.socket.emit('abrirChat', { id_emisor: this.idEmisor,
id_receptor: this.idReceptor });
  this.socket.on('recibirMensaje', (mensaje: Mensaje) => {
    mensaje.fecha_envio = new
Date(mensaje.fecha_envio).toLocaleString('es-ES', {
      timeZone: 'Europe/Madrid',
      day: '2-digit',
      month: '2-digit',
      year: 'numeric',

```

```
        hour: '2-digit',
        minute: '2-digit'
    })
    this.mensajes.push(mensaje);
    this.scrollAbajo();
  })
}
```

En el frontend también tengo este código en el que me encargo de que cuando se navegue a otra ruta de la aplicación desde el chat se eliminen todos los eventos que se están escuchando y se desconecte el socket.

```
ngOnDestroy(): void {
  if (this.socket) {
    this.socket.disconnect();
  }
}
```

## Conclusiones del proyecto

La aplicación final permite que los usuarios puedan crearse cuentas, iniciar sesión, llevar al día sus colecciones de cromos, ver los usuarios recomendados por cercanía (que vivan en su misma ciudad) y poder ponerse en contacto con los mismos gracias al sistema de mensajería en tiempo real. Esto último me parece lo más interesante ya que promueve la interacción entre los coleccionistas para ayudarse a completar sus colecciones y no tener que buscar a otros aficionados para intercambiar a través de otros métodos que pueden llegar a ser más tediosos.

Respecto a la temporalización estimada que puse en la anterior entrega, finalmente ha aumentado bastante el tiempo final que he dedicado al proyecto sobre todo debido al chat en tiempo real, los tests y retoques que he ido haciendo.

En cuanto a mi opinión personal, con este proyecto además de aprender cosas nuevas como el sistema de autenticación de usuario o uso de sockets para un chat único entre 2 usuarios, por ejemplo, considero que también estoy aprendiendo bastante sobre el diseño y estructura del backend. Además de que también es un proyecto al que una vez esté finalizado le podré dar un uso personal y seguro que gente que conozco también podrá hacerlo.

## Bibliografía

[v17.angular.io/docs](https://v17.angular.io/docs)

[Expressjs.com](https://expressjs.com)

[Jwt.io](https://jwt.io)

[Dev.mysql.com](https://dev.mysql.com)

[Npmjs.com/package/cors](https://npmjs.com/package/cors)

[socket.io/docs/v4/](https://socket.io/docs/v4/)

[jestjs.io/docs/getting-started](https://jestjs.io/docs/getting-started)

[github.com/ladjs/supertest](https://github.com/ladjs/supertest)