



GYM10

Eduardo Díaz Soria
IES Virgen del Carmen

DAM

2025/2026

CONTENIDO

01 MOTIVACIÓN Y OBJETIVOS

02 ANÁLISIS PREVIO Y METODOLOGÍA DE TRABAJO

03 DISEÑO: ARQUITECTURA, BASE DE DATOS E INTERFAZ

04 EJEMPLOS DE CÓDIGO.

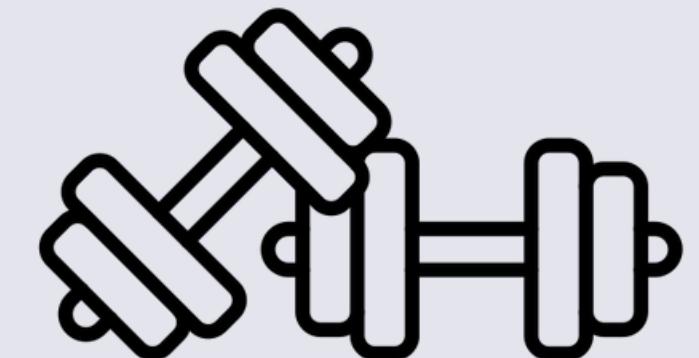
05 CONCLUSIÓN

06 BIBLIOGRAFÍA

Idea del Proyecto

Pensé en una aplicación de gimnasio ya que he probado varias y no me gustaba ni la interfaz o funcionamiento.

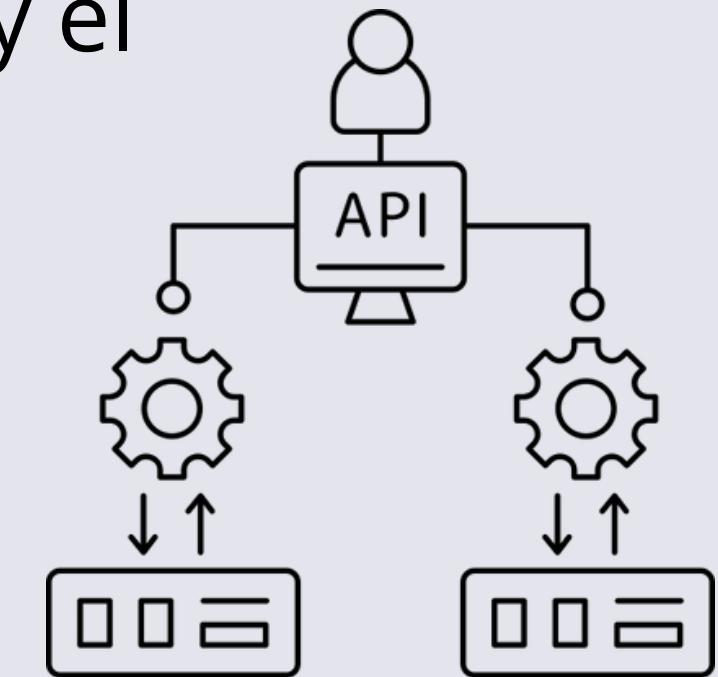
Como objetivo del funcionamiento era que fuera claro sus funcionalidades, que pudiera resolver un problema real a los administradores de un gimnasio, monitores y fuera útil para socios del gimnasio.



ANÁLISIS PREVIO AL DESARROLLO

Una vez clara la idea, elegí tecnologías para el desarrollo del proyecto, pero como innovación elegí un lenguaje de programación nuevo y arquitectura de diseño nueva.

Siguiendo una metodología de desarrollo incremental, donde el sistema se construyó microservicio por microservicio y el uso de REST API para comunicación entre servicios mediante peticiones HTTP

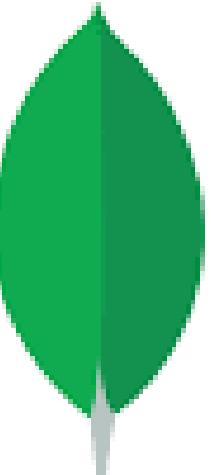


Tecnologías Backend

Tecnologías Principales

- **Backend:** Node.js con Express.js

Entorno de alto rendimiento y framework minimalista para crear APIs RESTful de manera eficiente.



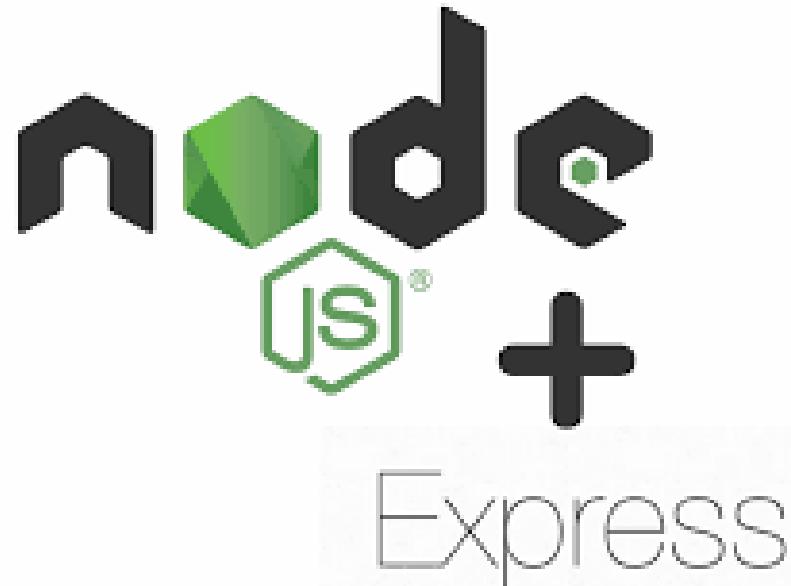
mongoDB®

- **Lenguaje:** TypeScript

Añade tipado estático a JavaScript para mayor robustez y mantenibilidad del código.

- **Base de Datos:** MongoDB con Mongoose

Base de datos NoSQL flexible y escalable, modelada mediante esquemas y validación con Mongoose.

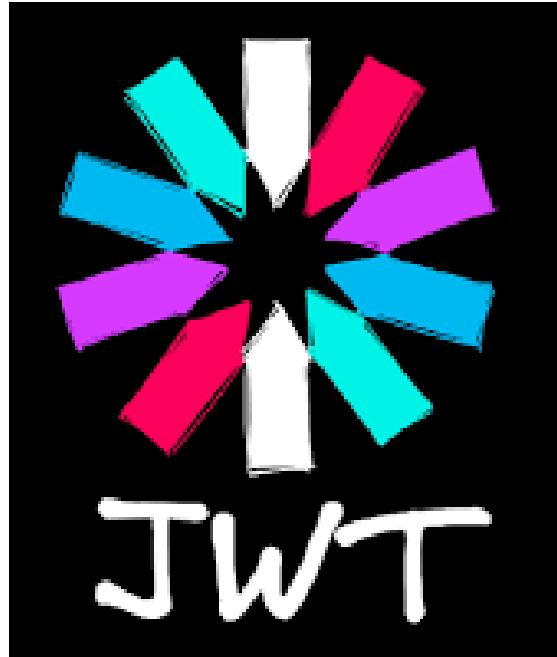


Tecnologías Backend

Seguridad, Utilidades y Despliegue

- **Autenticación & Seguridad:**

JWT: Autenticación sin estado entre servicios.



bcryptjs: Cifrado seguro de contraseñas.

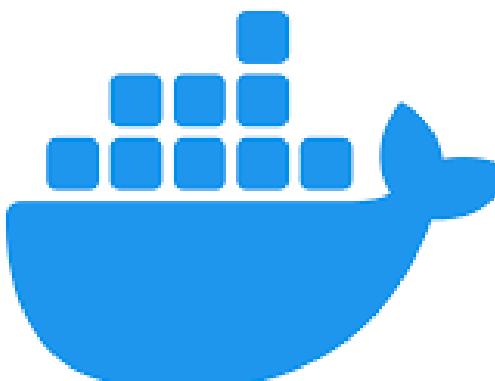
Helmet & CORS: Protección de headers HTTP y gestión de orígenes cruzados.

- **Utilidades:**

qrcode: Generación de códigos QR para acceso al gimnasio.

- **Despliegue & Contenedores:**

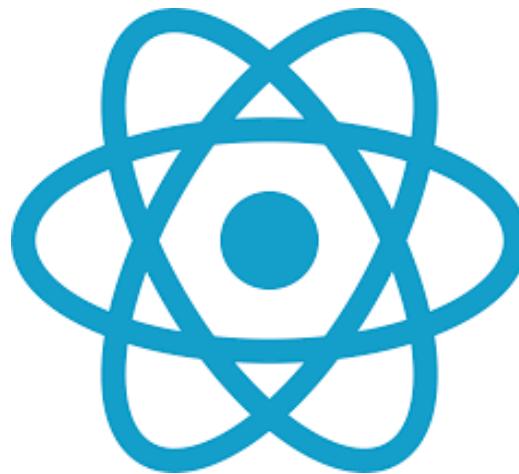
Docker: Contenerización para garantizar consistencia y aislamiento



Tecnologías frontend

Tecnologías Principales

- **React 18:** Biblioteca base para construir la interfaz de usuario mediante componentes.
- **TypeScript:** Tipado estático para mayor robustez y consistencia con el backend.
- **Vite:** Herramienta de construcción para un desarrollo rápido y empaquetado eficiente.
- **React Router DOM:** Manejo del enrutamiento y protección de rutas por roles.
- **Axios:** Cliente HTTP para consumir las APIs de los microservicios, configurado con autenticación.
- **react-qr-code:** Componente para renderizar códigos QR de acceso al gimnasio.



A X 1 O S

Ejemplos de Código



```
1  /**  
2   * Middleware para verificar si el usuario es Admin  
3   */  
4  export const isAdmin = (  
5    req: AuthRequest,  
6    res: Response,  
7    next: NextFunction  
8  ): void => {  
9    if (req.userRole !== UserRole.ADMIN) {  
10      res.status(403).json({  
11        success: false,  
12        message: 'Requiere permisos de administrador'  
13      });  
14      return;  
15    }  
16    next();  
17  };
```



```
1  export interface ApiResponse<T = unknown> {  
2    success: boolean;  
3    message?: string;  
4    data?: T;  
5    count?: number;  
6  }
```



```
1  /* Rutas de Rutinas */  
2  <Route  
3    path="/routines"  
4    element={  
5      <ProtectedRoute allowedRoles={[UserRole.SOCIO]}>  
6        <Routines />  
7      </ProtectedRoute>  
8    }  
9  />
```

Ejemplos de Código



```
1 // Interfaz que define la estructura del documento de usuario en MongoDB
2 export interface IUserDocument extends Document {
3   email: string;
4   password: string;
5   name: string;
6   role: UserRole;
7   dni?: string;
8   phone?: string;
9   birthDate?: Date;
10  isActive: boolean;
11  qrToken?: string; // Token unico para el QR de acceso
12  qrGeneratedAt?: Date; // Fecha de generacion del QR (para validar expiracion de 24h)
13 }
```



```
1 /* Titulo principal */
2 .qr-title {
3   font-size: 2rem;
4   color: #ffffff;
5   margin-bottom: 0.5rem;
6 }
7
8 /* Subtitulo */
9 .qr-subtitle {
10  color: #cccccc;
11  margin-bottom: 2rem;
12  font-size: 1.1rem;
13 }
```



```
1 // Generar token JWT
2 const payload: JWTPayload = {
3   id: String(user._id),
4   email: user.email,
5   role: user.role
6 };
```



```
1 // POST /api/bookings - Crear reserva
2 router.post('/', verifyToken, createBooking);
3
4 // GET /api/bookings/my-bookings - Mis reservas - Ruta especifica primero
5 router.get('/my-bookings', verifyToken, getMyBookings);
6
7 // PUT /api/bookings/:id/cancel - Cancelar reserva
8 router.put('/:id/cancel', verifyToken, cancelBooking);
```

```
1 // Verificar que el usuario tiene una suscripción activa
2 try {
3     const token = req.headers.authorization?.split(' ')[1];
4     const subscriptionResponse = await axios.get(
5         `${PAYMENT_SERVICE_URL}/api/payments/me/active`,
6     {
7         headers: {
8             Authorization: `Bearer ${token}`
9         }
10    }
11 );
12
13 if (subscriptionResponse.data.success && subscriptionResponse.data.data) {
14     const hasActiveSubscription = subscriptionResponse.data.data.hasActiveSubscription;
15     if (!hasActiveSubscription) {
16         res.status(403).json({
17             success: false,
18             message: 'Necesitas una suscripción activa para reservar clases'
19         });
20         return;
21     }
22 }
23 } catch (error: any) {
24 // Si el servicio de pagos no está disponible, denegar la reserva por seguridad
25     res.status(503).json({
26         success: false,
27         message: 'No se pudo verificar tu suscripción. Por favor, intenta más tarde.'
28     });
29     return;
30 }
```

```
1 // Enum para tipos de feedback
2 export enum FeedbackType {
3     QUEJA = 'queja',
4     VALORACION = 'valoracion',
5     DUDA = 'duda'
6 }
7
8 // Enums para rutinas y ejercicios
9 export enum MuscleGroup {
10    PECHO = 'pecho',
11    ESPALDA = 'espalda',
12    HOMBROS = 'hombros',
13    BICEPS = 'biceps',
14    TRICEPS = 'triceps',
15    ANTEBRAZOS = 'antebrazos',
16    CUADRICEPS = 'cuadriceps',
17    ISQUIOTIBIALES = 'isquiotibiales',
18    GEMELO = 'gemelo',
19    GLUTEOS = 'gluteos',
20    LUMBARES = 'lumbares',
21    ABDOMINALES = 'abdominales',
22    CARDIO = 'cardio',
23    FULL_BODY = 'full_body'
24 }
25
26 // Enum de días de la semana
27 export enum DayOfWeek {
28    LUNES = 'lunes',
29    MARTES = 'martes',
30    MIERCOLES = 'miercoles',
31    JUEVES = 'jueves',
32    VIERNES = 'viernes',
33    SABADO = 'sabado',
34    DOMINGO = 'domingo'
35 }
```

Estructura de un microservicio(auth-service)



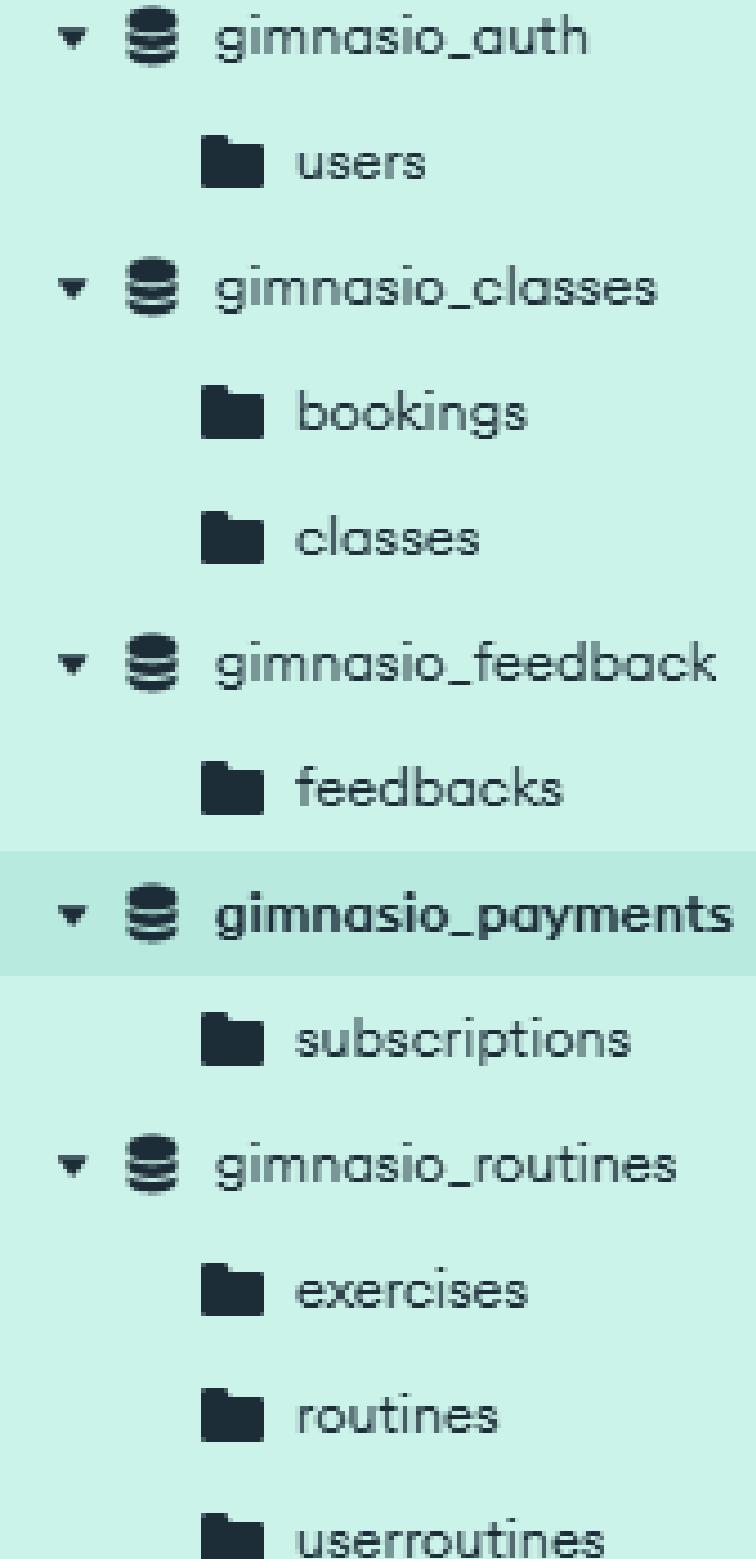
```
1 `src/config/` - Configuración de base de datos  
2 - `src/controllers/` - Controladores de autenticación y administración  
3 - `src/middleware/` - Middleware de autenticación JWT y validación  
4 - `src/models/` - Modelos (User)  
5 - `src/routes/` - Definición de rutas de la API  
6 - `src/scripts/` - Scripts de inicialización y utilidades  
7 - `src/types/` - Definiciones de tipos TypeScript
```

Estructura del Frontend



```
1 - `components/` - Componentes reutilizables (Navbar, ProtectedRoute)  
2 - `contexts/` - Context API para gestión de estado global (AuthContext)  
3 - `hooks/` - Custom hooks (useAuth)  
4 - `pages/` - Páginas de la aplicación (Login, Register, Dashboard, etc.)  
5 - `services/` - Servicios para comunicación con APIs (authService, classService, etc.)  
6 - `styles/` - Archivos CSS por componente  
7 - `types/` - Definiciones de tipos TypeScript compartidos  
8 - `utils/` - Utilidades y helpers (errorHandler)
```

MongoDB Compass



¿De que se encarga cada uno?



- 1 - **Auth Service (Puerto 3001)**: Gestiona usuarios, autenticación y autorización
 - 2 - Base de datos: `gimnasio_auth`
- 3
- 4 - **Class Service (Puerto 3002)**: Gestiona clases y reservas
 - 5 - Base de datos: `gimnasio_classes`
- 6
- 7 - **Payment Service (Puerto 3003)**: Gestiona pagos y suscripciones
 - 8 - Base de datos: `gimnasio_payments`
- 9
- 10 - **Feedback Service (Puerto 3004)**: Gestiona feedback, quejas y valoraciones de los socios
 - 11 - Base de datos: `gimnasio_feedback`
- 12
- 13 - **Routine Service (Puerto 3005)**: Gestiona rutinas de ejercicios y ejercicios personalizados
 - 14 - Base de datos: `gimnasio_routines`
- 15
- 16 La comunicación entre servicios se realiza mediante JWT tokens que contienen la información del usuario.

CONCLUSIÓN

He aprendido sobre arquitectura de microservicios, uno de los tipos de arquitectura más utilizados actualmente en la industria de software.

He aprendido a crear y ser capaz de diseñar y organizar aquellos servicios independientes, donde cada uno de ellos tiene su propia responsabilidad y base de datos, además de garantizar la escalabilidad y la mantenibilidad del proyecto.

Tambien mi proyecto podría haber tenido otros microservicios pero ya queda en la selección del cliente o necesidades de uno mismo, un ejemplo de mejora un microservicio de inventario o de progreso de los socios

BIBLIOGRAFIA

Algunas paginas o videos que he usado:

Para iconos y fotos:

<https://www.flaticon.es>

<https://www.freepik.es/fotos/gym>

Para el desarrollo:

<https://react.dev/>

<https://www.youtube.com/watch?v=4W3UWjyyVkJQ>

https://www.tiktok.com/@dev_flash

<https://www.typescriptlang.org/>

