

# UNIVERSIDAD DON BOSCO



## Diseño y Programación de Software Multiplataforma DPS441 G04T

CICLO I 2025

### ALUMNOS:

- |   |          |
|---|----------|
| • Bryan Alexander Realegeño García      | RG240109 |
| • Fátima Gisselle Cornejo Meléndez      | CM240108 |
| • Cristian Alexander Hernández Valiente | HV240081 |
| • Bryan Steven Hernández Polio          | HP240512 |
| • Mario Antonio Pacheco Guerrero        | PG240099 |

**Docente:** Kevin Jiménez    **Fecha de entrega:** 29/05/2025

<b>1. Introducción.....</b>	<b>3</b>
<b>2. Tecnologías utilizadas.....</b>	<b>3</b>
2.1 Frontend.....	3
2.2 Backend.....	3
2.3 Base de datos.....	3
2.4 Contenedorización.....	4
2.5 Orquestación.....	4
2.6 Control de versiones.....	4
<b>3. Instalación del backend.....</b>	<b>4</b>
3.1 Node.js.....	4
3.2 Express.js.....	5
3.2.1 Estructura del backend.....	5
3.2.2 Explicación de la estructura del backend.....	6
<b>4. Frontend, la interfaz de la aplicación.....</b>	<b>6</b>
4.1 Crear proyecto con React Native.....	7
4.1.1 Estructura del proyecto frontend.....	7
4.1.2 Explicación de la estructura del proyecto.....	8
<b>5. Base de datos.....</b>	<b>9</b>
5.1 Estructura de la base de datos.....	9
5.2 Diccionario de Datos.....	9
5.3 Diagrama de Entidad-Relación.....	10
5.4 Diagrama UML.....	11
<b>6. Docker y Google Container Registry (CGR).....</b>	<b>11</b>
6.1 Creación de archivo Dockerfile.....	11
6.2 Construcción de la imagen de Docker.....	12
6.3 Subir la imagen a Google Container Registry (CGR).....	12
6.4 Despliegue en Kubernetes.....	13

# MANUAL TÉCNICO

## Aplicación Móvil Coach GYM

### 1. Introducción

Coach Gym es una aplicación móvil multiplataforma diseñada para ayudar a los usuarios a gestionar rutinas de ejercicios y planes de alimentación personalizados.

El manual técnico está dirigido a desarrolladores, integradores y técnicos encargados del despliegue y mantenimiento del sistema. Se describe la arquitectura del sistema, tecnologías utilizadas, estructura del código fuente, procesos de instalación y configuración del backend y del frontend.

### 2. Tecnologías utilizadas

#### 2.1 Frontend

Para el frontend se ocupó React Native para la interfaz móvil de la aplicación Coach Gym. Los usuarios pueden acceder a sus rutinas, planes alimenticios y ejercicios a través de una aplicación instalada en su dispositivo Android o iOS.

#### 2.2 Backend

Para el Backend se ocupó Node.js y el web framework Express.js. Sirve para construir la lógica del servidor, entre ellos, la autenticación, gestión de usuarios, rutinas, planes alimenticios y comunicación con la base de datos. El backend responde las solicitudes del frontend mediante una API REST.

#### 2.3 Base de datos

Se utiliza la imagen oficial de MYSQL. La aplicación almacena información estructurada como usuarios y rutinas en MySQL. Las tablas se relacionan usando claves primarias y foráneas

## 2.4 Contenedorización

Para la contenedorización se usa Docker, el cual cada componente del frontend, backend y la base de datos se puede ejecutar en su propio contenedor, lo que permite pruebas y despliegues consistentes entre entornos. Para el frontend se define un Dockerfile, igualmente para el backend y para la base de datos se usa Docker Hub para crear un contenedor que gestione la base de datos

## 2.5 Orquestación

- **Despliegue**
  - Se utiliza Kubernetes para orquestar múltiples contenedores (frontend, backend, base de datos) y se asegura que se gestione el despliegue, la escalabilidad y la disponibilidad de la aplicación.
- **Archivos YAML**
  - Se crea archivos YAML para definir los servicios, deployments y pods necesarios para desplegar la aplicación en un clúster de Kubernetes
  - Los servicios expondrán la API del backend y la base de datos, permitiendo la comunicación entre los diferentes componentes.
  - Los deployments garantizarán la replicación de los pods, asegurando la disponibilidad y escalabilidad de la aplicación.
  - Se configurarán los pods para que se comuniquen entre sí, mediante los servicios creados en los archivos YAML.

## 2.6 Control de versiones

Se usa github como control de versiones para almacenar el código fuente de la aplicación. El equipo hace uso de ramas, commits y pull requests para colaborar y revisar los cambios que se hacen.

# 3. Instalación del backend

Requisitos previos:

- **Sistema Operativo:** Windows, Linux o macOS
- **Conexión a internet**
- **Editor de código:** Visual Studio Code (recomendado, puede usar otro)
- **Node.js**

### 3.1 Node.js

- Visitar el sitio oficial de node.js <https://nodejs.org/>
- Descargar la versión LTS (Long Term Support)
- Instalar siguiendo los pasos del asistente de instalación.
- Abrir la consola (CMD o Terminal) y verificar que se haya instalado poniendo los siguientes comandos:

- node -v
- npm -v

Si aparece la versión significa que si está instalado correctamente.

- Instalar las dependencias para ello se usa el comando “npm install”

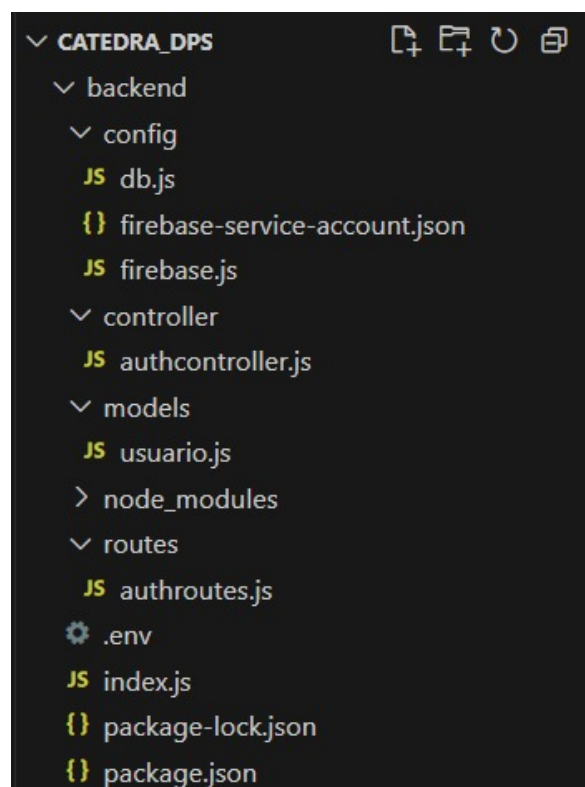
## 3.2 Express.js

- Se crea la carpeta “backend” dentro del proyecto
- Se coloca el siguiente comando “npm init -y”, para generar el archivo package.json que contiene la configuración básica del proyecto.
- Se instala express y otras dependencias con el siguiente comando:  
“**npm install bcryptjs cors dotenv express firebase-admin mysql2**”

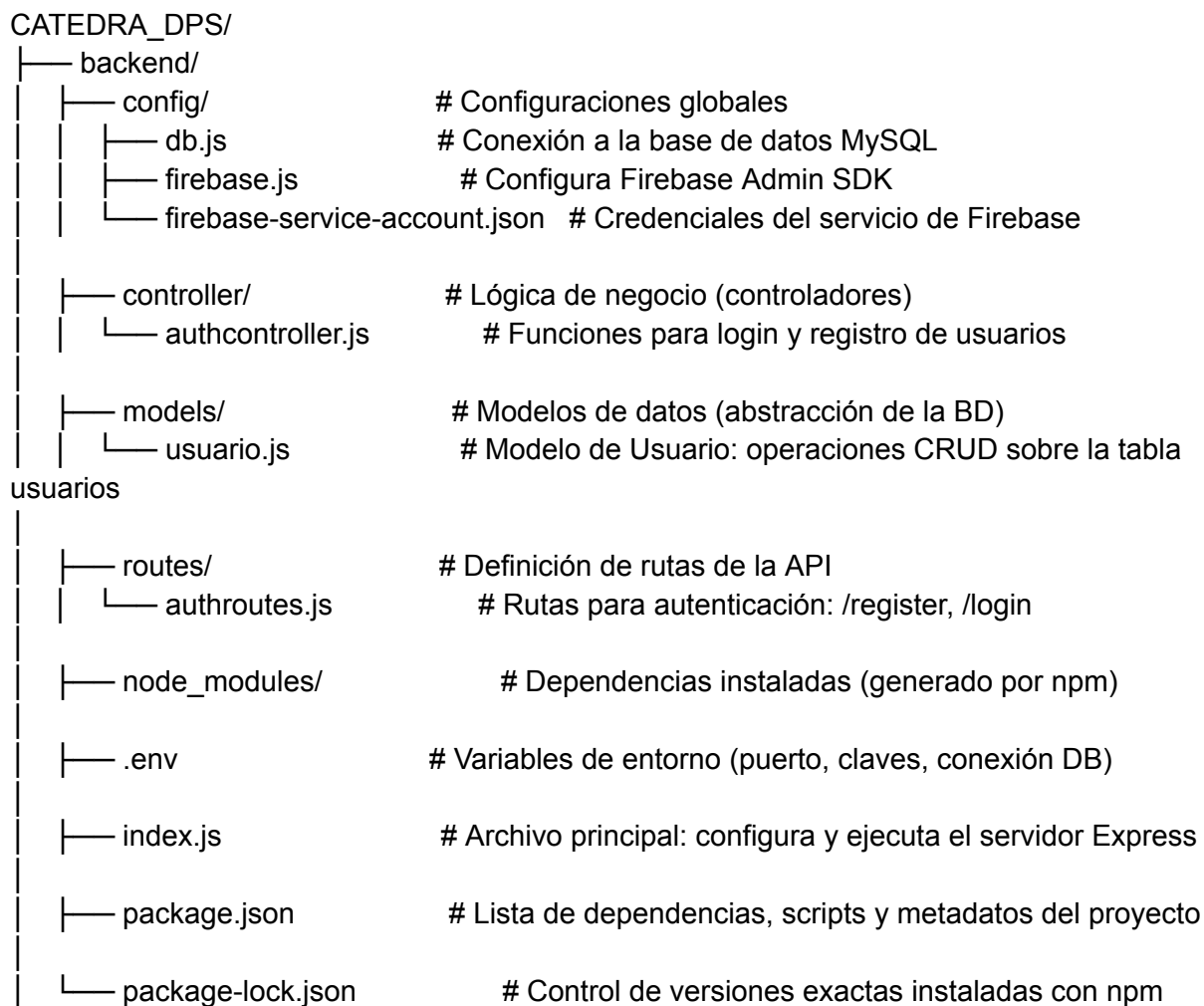
Las librerías que se instalan por este comando son:

- **express:** El framework minimalista de Node.js para crear el servidor y definir rutas.
- **cors:** Middleware que permite gestionar las solicitudes entre dominios diferentes.
- **dotenv:** Permite usar variables de entorno definidas en un archivo .env.
- **mysql2:** Cliente de MySQL para Node.js. Se usa para conectar y hacer consultas a la base de datos.
- **bcryptjs:** Biblioteca para encriptar contraseñas. Más ligeras y rápidas que bcrypt para algunos entornos.
- **firebase-admin:** SDK de administración de Firebase (no es necesario si no se usa Firebase para autenticación o notificaciones).

### 3.2.1 Estructura del backend



### 3.2.2 Explicación de la estructura del backend



## 4. Frontend, la interfaz de la aplicación

Para crear el proyecto de React Native, debemos cumplir ciertos requisitos previos

- Node.js.
- Expo CLI (opcional, pero recomendado para principiantes).
- Editor de código Visual Studio Code (recomendado, puede usar otro).
- Emulador de Android o dispositivo móvil con la app de Expo Go instalada.

## 4.1 Crear proyecto con React Native

Para crear el proyecto de React Native, usando CMD o Terminal, lo primero es verificar si tenemos instalado Node.js, si no lo tiene hay que instalarlo.

- Visitar el sitio oficial de node.js <https://nodejs.org/>
- Descargar la versión LTS (Long Term Support)
- Instalar siguiendo los pasos del asistente de instalación.
- Abrir la consola (CMD o Terminal) y verificar que se haya instalado poniendo los siguientes comandos:
  - node -v
  - npm -v

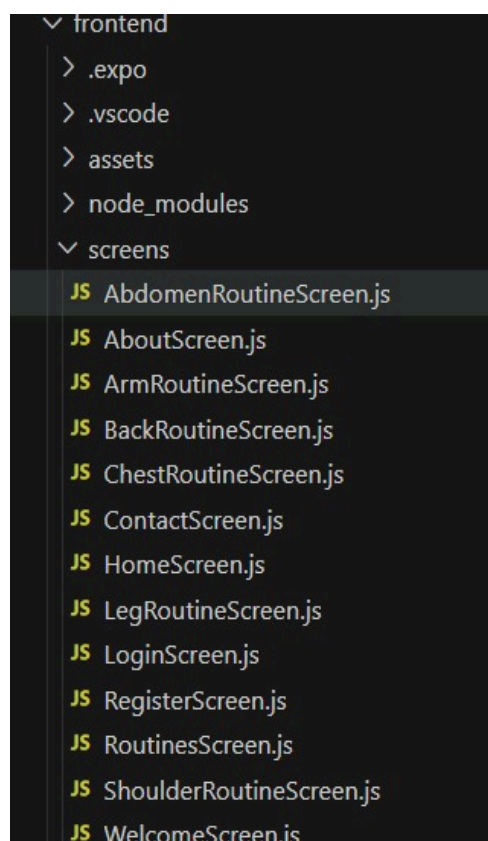
Si aparece la versión significa que si esta instalado correctamente.

En la terminal, en el directorio que usted quiera, lo recomendable es que sea en el escritorio colocar el siguiente comando:

**“npx create-expo-app frontend”**

- **npx**: Ejecuta un paquete sin necesidad de instalarlo globalmente
- **create-expo-app**: Herramienta oficial de Expo que genera la estructura inicial de una aplicación React Native optimizada para desarrollo rápido.
- **frontend**: Es el nombre del proyecto y carpeta donde se genera el código fuente del frontend.

### 4.1.1 Estructura del proyecto frontend



### 4.1.2 Explicación de la estructura del proyecto

Luego de crear el proyecto aparecerá una estructura sencilla por predeterminado:

```
frontend/  
├── App.js  
├── package.json  
├── babel.config.js  
├── node_modules/  
└── assets/
```

En la imagen mostrada se puede ver la siguiente estructura, el cual la screens son las vistas de la aplicación y App.js es nuestro archivo principal y también se encuentra la rutas de navegación entre pestañas.

```
/frontend  
├── .expo/           # Configuración interna de Expo  
├── .vscode/         # Configuración del editor (opcional)  
├── assets/          # Recursos como imágenes, íconos, fuentes  
├── node_modules/    # Dependencias del proyecto  
├── screens/         # Pantallas individuales de la aplicación  
│   ├── AbdomenRoutineScreen.js  
│   ├── ArmRoutineScreen.js  
│   ├── BackRoutineScreen.js  
│   └── ChestRoutineScreen.js
```



		ShoulderRoutineScreen.js	
		LegRoutineScreen.js	
		AboutScreen.js	
		ContactScreen.js	
		HomeScreen.js	
		LoginScreen.js	
		RegisterScreen.js	
		RoutinesScreen.js	
		WelcomeScreen.js	
		.gitignore	# Archivos ignorados por Git
		App.js	# Archivo principal y para la navegación entre pestañas
		app.json	# Configuración del proyecto Expo
		index.js	# Entry point (gestiona registro de la app)
		package.json	# Lista de dependencias y scripts
		package-lock.json	# Bloqueo de versiones de dependencias

## 5. Base de datos

### 5.1 Estructura de la base de datos

El sistema maneja tres tipos de tablas que son las principales que ayudan a almacenar los datos en la aplicación

- **Usuarios:** Contiene la información de cada usuario registrado.
- **Planes alimenticios:** Almacena los planes de alimentación asignados a cada usuario.
- **Rutinas:** Guarda las rutinas de ejercicio asociadas a cada usuario.

### 5.2 Diccionario de Datos

Los tres tipos de tablas contienen los siguientes datos:

#### Usuarios

- **id\_usuario:** Identificador único del usuario, número entero autoincremental.
- **Nombre:** nombre del usuario, cadena de texto de hasta 100 caracteres. No puede quedar vacío.
- **email:** Correo electrónico, también de hasta 100 caracteres. Es único en la base de datos.
- **contraseña:** Almacena la clave de acceso encriptada, con un límite de 255 caracteres. No puede estar vacío.
- **rol:** Tipo de usuario (administrador, usuario normal, etc.), almacenado como ENUM.
- **fecha\_registro:** Fecha y hora en que se registró el usuario, se guarda automáticamente.

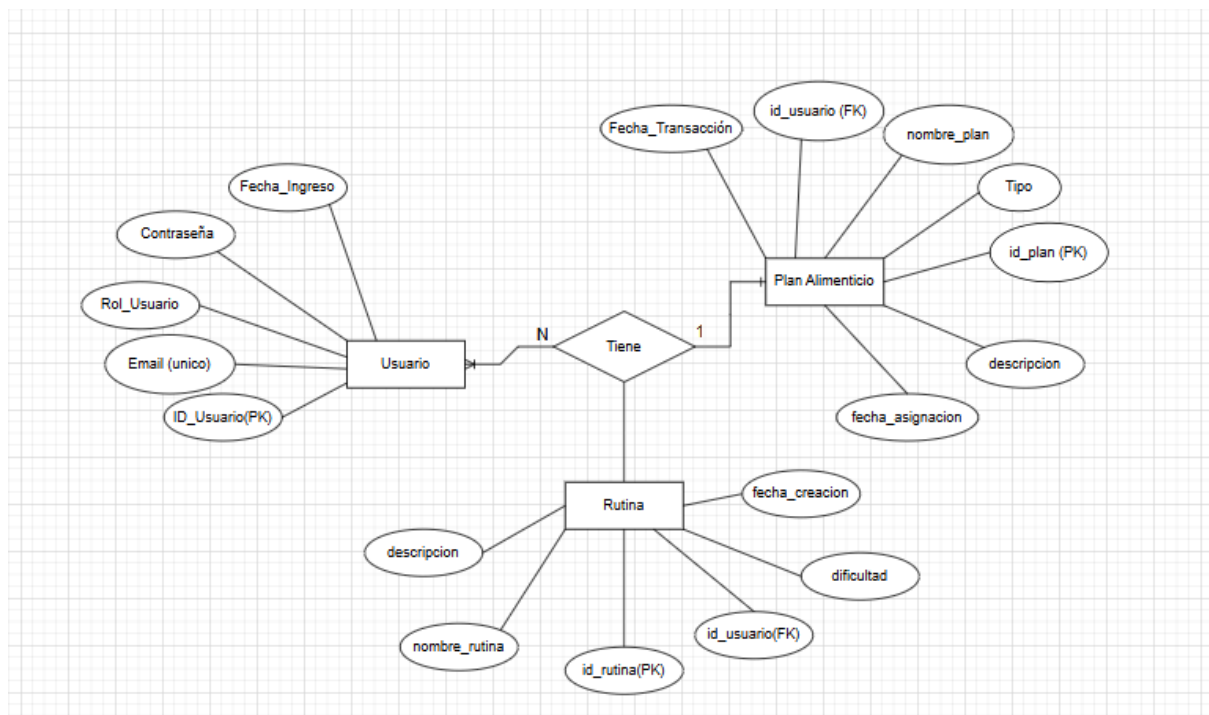
#### Planes Alimenticios

- **id\_plan:** Identificador único del plan, número entero autoincremental.
- **id\_usuario:** Usuario al que pertenece el plan, clave foránea que referencia a "Usuarios".
- **nombre\_plan:** Nombre del plan, texto de hasta 100 caracteres. No puede quedar vacío.
- **descripcion:** Detalles del plan de alimentación, tipo TEXT. Puede quedar vacío.
- **tipo:** Tipo de plan, almacenado como ENUM.
- **fecha\_asignacion:** Fecha en que se asignó el plan, generada automáticamente.

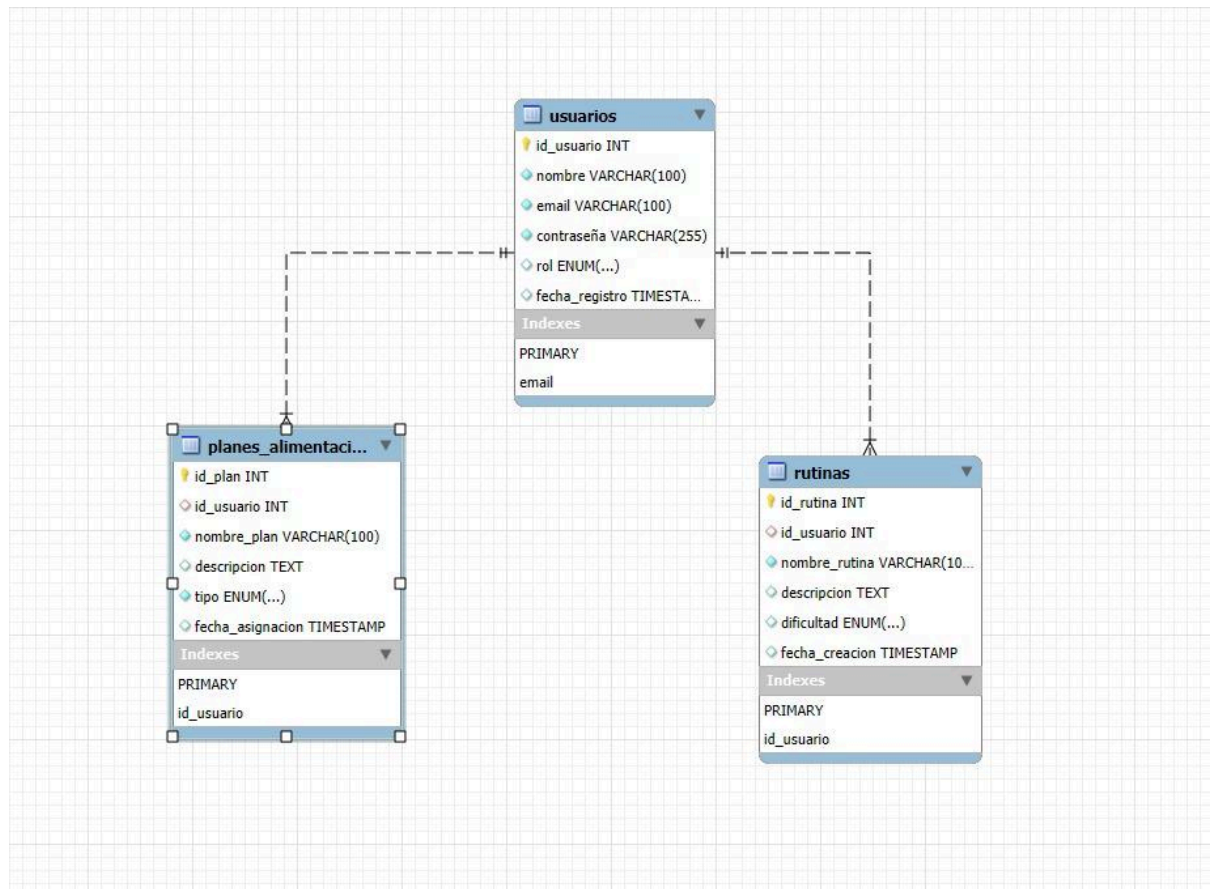
## Rutinas

- **id\_rutina:** Identificador único de la rutina, número entero autoincremental.
- **id\_usuario:** Usuario al que pertenece la rutina, clave foránea hacia "Usuarios".
- **nombre\_rutina:** Nombre de la rutina, texto de hasta 100 caracteres. No puede quedar vacío.
- **descripcion:** Detalles de la rutina, tipo TEXT. Puede quedar vacío.
- **dificultad:** Nivel de dificultad, almacenado como ENUM.
- **fecha\_creacion:** Fecha en la que se creó la rutina, generada automáticamente.

## 5.3 Diagrama de Entidad-Relación



## 5.4 Diagrama UML



## 6. Docker y Google Container Registry (CGR)

### 6.1 Creación de archivo Dockerfile

Hay que colocar en la raíz de la carpeta Backend

```
# Usa la imagen oficial de Node.js como base
FROM node:18
```

```
# Establece el directorio de trabajo dentro del contenedor
WORKDIR /app
```

```
# Copia los archivos del proyecto al contenedor
COPY package*.json ./
RUN npm install
```

```
# Copia el resto del código
COPY . .
```

# Expone el puerto (debe coincidir con el que usa Express)  
EXPOSE 5000

# Comando que se ejecutará al iniciar el contenedor  
CMD ["node", "index.js"]

**Asegúrate de usar .dockerignore para evitar copiar archivos como node\_modules, .env y otros sensibles.**

## 6.2 Construcción de la imagen de Docker

Primero se construye la imagen Docker con el siguiente comando:

**docker build -t gcr.io/autenticaciondps/autenticaciondps-backend:v1 .**

¿Qué hace este comando?

- **docker build:** construye una imagen Docker a partir del archivo Dockerfile en el directorio actual (.).
- **-t:** etiqueta la imagen con un nombre y una versión (v1).
- **gcr.io/autenticaciondps/autenticaciondps-backend:v1:** nombre completo de la imagen incluyendo el registro (GCR), el proyecto (autenticaciondps), el nombre del contenedor (autenticaciondps-backend) y su versión (v1).

## 6.3 Subir la imagen a Google Container Registry (CGR)

Requisitos previos

- Tener acceso al proyecto en Google Cloud Platform (GCP)
- Haber ejecutado:  
**gcloud auth login**  
**gcloud auth configure-docker**

Si ya cumples con los requisitos previos puedes poner el siguiente comando

**docker push gcr.io/autenticaciondps/autenticaciondps-backend:v1**

¿Qué hace este comando?

- **docker push:** envía (sube) la imagen creada a un registro remoto.
- **gcr.io/...:** URL del Container Registry de Google Cloud.

Luego verificar si puedes ver la imagen en:

<https://console.cloud.google.com/gcr/images>

## 6.4 Despliegue en Kubernetes

Deberá tener la imagen Docker creada y subirlo a un registro de contenedores como GCR. Como se menciona, para los Kubernetes se crean archivos .yaml para exponer el backend. Se define un deployment.yaml que indica:

- Cuantos pods correr.
- Qué imagen usar.
- Qué variables de entorno se necesitan.
- Conexión de la base de datos con un contenedor con MySQL mediante variables.

Se define un service.yaml para exponer esos pods:

- Si un pod falla, se reinicia.
- Si hay más carga, crea más pods.