

Documentación de API de Gestión de Restaurante

Daniel Calvar Cruz

7 de diciembre de 2025

Índice

1. Introducción	3
2. Estructura	3
2.1. Rutas Principales	3
2.2. Modelos de Datos	3
2.2.1. Modelo Grupo	3
2.2.2. Modelo Usuario	3
3. Base de Datos	5
4. Endpoints	6
5. Endpoints	6
5.1. Endpoints POST	6
5.1.1. POST /api/usuarios/create	6
5.1.2. POST /api/usuarios/read	6
5.1.3. POST /api/grupos/create	7
5.1.4. POST /api/grupos/read	7
5.2. Endpoints PUT	8
5.2.1. PUT /api/usuarios/update	8
5.2.2. PUT /api/grupos/update	8
5.3. Endpoints DELETE	9
5.3.1. DELETE /api/usuarios/delete	9
5.3.2. DELETE /api/grupos/delete	9
5.4. Endpoints GET	10
5.4.1. GET /api/usuarios/readall	10
5.4.2. GET /api/grupos/readall	10
6. Manejo de Errores	11
7. Ejemplos de Uso	11
8. Consideraciones Técnicas	12
8.1. Códigos de Estado HTTP	12
8.2. Stack tecnológico	12
8.3. Variables de entorno (.env)	12

9. Dockerizacion	13
9.1. Dockerfile	13
9.2. Archivo YAML	14
9.3. Pushear imagen a Dockerhub	16
10. Github Actions	18
10.1. Notificacion al pushear	19

1. Introducción

[Volver](#)

Esta API ofrece un servicio CRUD sobre una BBDD con dos colecciones, grupos y usuarios.

La API está construida con Express.js y utiliza MongoDB como base de datos.

2. Estructura

2.1. Rutas Principales

La API está organizada en dos apartados:

- Grupo
- Usuario

2.2. Modelos de Datos

2.2.1. Modelo Grupo

```
({
  nombre: {
    type: String
  },
  descripcion: {
    type: String
  },
  activo: {
    type: Boolean
  }
});
```

2.2.2. Modelo Usuario

```
({
  nombre: {
    type: String
  },
  apellido1: {
    type: String
  },
  apellido2: {
    type: String
  },
  edad: {
    type: Number
  }
});
```

```
    }  
});
```

3. Base de Datos

[**Volver**](#)

MongoDB (Atlas).

Estructura

- grupos: contiene los estados de los grupos.
- usuarios: contiene los estados de los usuarios.

4. Endpoints

[Volver](#)

5. Endpoints

[Volver](#)

5.1. Endpoints POST

5.1.1. POST /api/usuarios/create

Descripción: Crea un nuevo usuario en el sistema.

Body:

```
{  
  "nombre": "string",  
  "apellido1": "string",  
  "apellido2": "string",  
  "edad": "number"  
}
```

Respuestas:

- **200:** Usuario creado exitosamente
- **400:** Error en los datos proporcionados
- **500:** Error interno del servidor

5.1.2. POST /api/usuarios/read

Descripción: Busca usuarios según un filtro específico.

Body:

```
{  
  "filtroKey": "string",  
  "filtroValue": "string|number"  
}
```

Ejemplos de uso:

- Buscar por nombre: {"filtroKey": "nombre", "filtroValue": "Ana"}
- Buscar por edad: {"filtroKey": "edad", "filtroValue": 28}

Respuestas:

- **200:** Información encontrada con datos
- **404:** Usuarios no encontrados
- **500:** Error interno del servidor

5.1.3. POST /api/grupos/create

Descripción: Crea un nuevo grupo en el sistema.

Body:

```
{  
  "nombre": "string",  
  "descripcion": "string",  
  "activo": "boolean"  
}
```

Respuestas:

- **200:** Grupo creado exitosamente
- **400:** Error en los datos proporcionados
- **500:** Error interno del servidor

5.1.4. POST /api/grupos/read

Descripción: Busca grupos según un filtro específico.

Body:

```
{  
  "filtroKey": "string",  
  "filtroValue": "string|boolean"  
}
```

Ejemplos de uso:

- Buscar por nombre: {"filtroKey": "nombre", "filtroValue": "Administradores"}
- Buscar por estado: {"filtroKey": "activo", "filtroValue": true}

Respuestas:

- **200:** Información encontrada con datos
- **404:** Grupos no encontrados
- **500:** Error interno del servidor

5.2. Endpoints PUT

[Volver](#)

5.2.1. PUT /api/usuarios/update

Descripción: Actualiza la información de un usuario existente.

Body:

```
{  
  "_id": "string",  
  "nombre": "string",  
  "apellido1": "string",  
  "apellido2": "string",  
  "edad": "number"  
}
```

Respuestas:

- **200:** Usuario actualizado exitosamente
- **400:** Error en los datos proporcionados
- **404:** Usuario no encontrado
- **500:** Error interno del servidor

5.2.2. PUT /api/grupos/update

Descripción: Actualiza la información de un grupo existente.

Body:

```
{  
  "_id": "string",  
  "nombre": "string",  
  "descripcion": "string",  
  "activo": "boolean"  
}
```

Respuestas:

- **200:** Grupo actualizado exitosamente
- **400:** Error en los datos proporcionados
- **404:** Grupo no encontrado
- **500:** Error interno del servidor

5.3. Endpoints DELETE

[Volver](#)

5.3.1. DELETE /api/usuarios/delete

Descripción: Elimina un usuario del sistema.

Body:

```
{  
  "_id": "string"  
}
```

Respuestas:

- **200:** Usuario eliminado exitosamente
- **400:** Error en los datos proporcionados
- **404:** Usuario no encontrado
- **500:** Error interno del servidor

5.3.2. DELETE /api/grupos/delete

Descripción: Elimina un grupo del sistema.

Body:

```
{  
  "_id": "string"  
}
```

Respuestas:

- **200:** Grupo eliminado exitosamente
- **400:** Error en los datos proporcionados
- **404:** Grupo no encontrado
- **500:** Error interno del servidor

5.4. Endpoints GET

[Volver](#)

5.4.1. GET /api/usuarios/readall

Descripción: Obtiene todos los usuarios del sistema.

Parámetros: Ninguno

Respuestas:

- **200:** Lista de usuarios obtenida exitosamente
- **404:** No hay usuarios registrados
- **500:** Error interno del servidor

5.4.2. GET /api/grupos/readall

Descripción: Obtiene todos los grupos del sistema.

Parámetros: Ninguno

Respuestas:

- **200:** Lista de grupos obtenida exitosamente
- **404:** No hay grupos registrados
- **500:** Error interno del servidor

6. Manejo de Errores

[Volver](#)

La API utiliza un sistema consistente de respuestas:

```
{  
  "type": "success|failure",  
  "message": "string descriptivo",  
  "data": "object (opcional)"  
}
```

7. Ejemplos de Uso

```
POST http://localhost:5000/api/usuarios/create  
Content-Type: application/json
```

```
{  
  "nombre": "Carlos",  
  "apellido1": "Martínez",  
  "apellido2": "Rodríguez",  
  "edad": 35  
}
```

```
POST http://localhost:5000/api/grupos/create  
Content-Type: application/json
```

```
{  
  "nombre": "Desarrolladores",  
  "descripcion": "Equipo de desarrollo de software",  
  "activo": true  
}
```

```
POST http://localhost:5000/api/usuarios/read  
Content-Type: application/json
```

```
{  
  "filtroKey": "nombre",  
  "filtroValue": "Carlos"  
}
```

```
POST http://localhost:5000/api/grupos/read  
Content-Type: application/json
```

```
{  
  "filtroKey": "activo",  
  "filtroValue": true  
}
```

8. Consideraciones Técnicas

[Volver](#)

8.1. Códigos de Estado HTTP

- **200 OK:** Operación exitosa
- **400 Bad Request:** Error en los datos enviados
- **404 Not Found:** Recurso no encontrado
- **500 Internal Server Error:** Error del servidor

8.2. Stack tecnológico

- **Versión Node.js:** 22.14
- **Base de datos:** MongoDB Atlas
- **Dependencias NPM:**
 - **cors:** 2.8.5
 - **dotenv:** 16.4.7
 - **express:** 4.21.1
 - **mongodb:** 6.12.0
 - **mongoose:** 8.13.2
 - **nodemon:** 3.1.10

8.3. Variables de entorno (.env)

- **PORT:** Puerto en el que se ejecutará el servidor.
- **MONGO_INITDB_ROOT_USERNAME:** Nombre del usuario de la BBDD.
- **MONGO_INITDB_ROOT_PASSWORD:** Password de la BBDD.
- **MONGO_INITDB_DATABASE:** Nombre de la BBDD.

9. Dockerizacion

[Volver](#)

9.1. Dockerfile

```
FROM node:20

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install

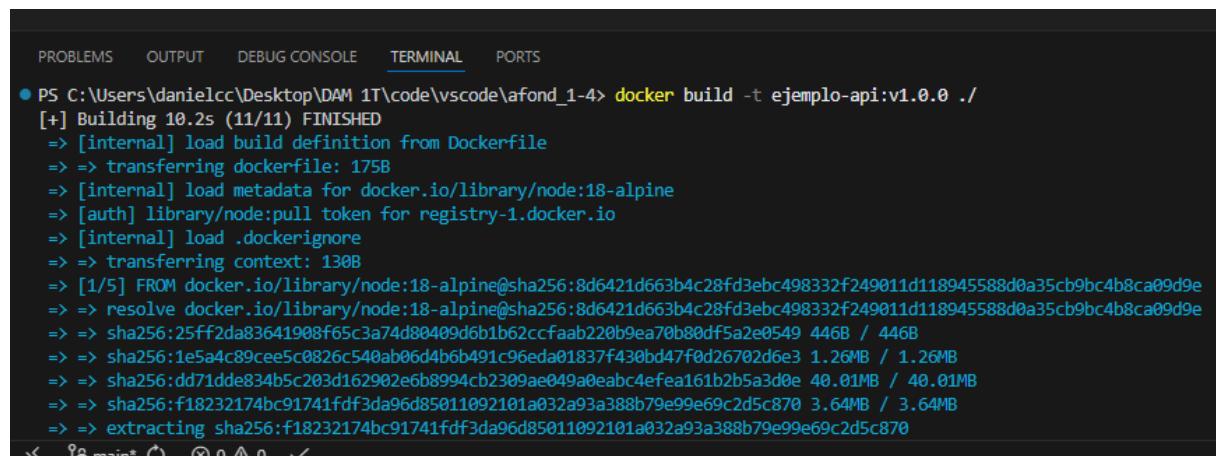
COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

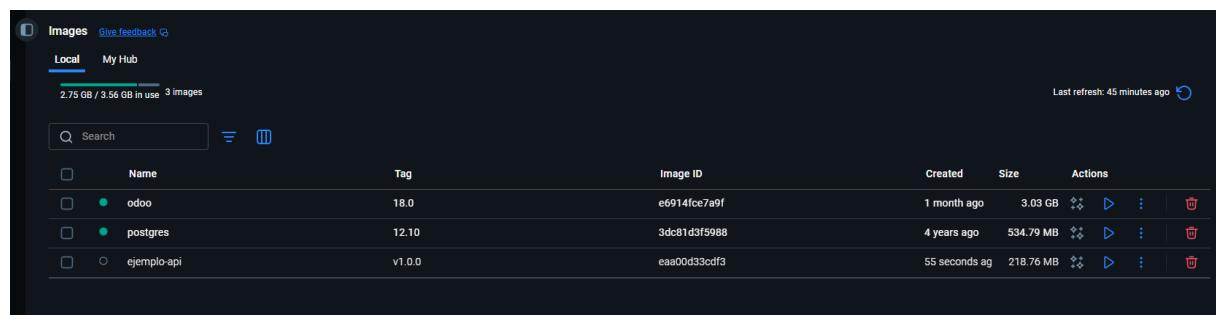
Usamos Node.js, versión 20, definimos el directorio para la imagen, instalamos las dependencias de node, copiamos proyecto, iniciamos programa.

Comando para levantar imagen: `docker build -t ejemplo-api:v1.0.0 ./`



The screenshot shows the VS Code interface with the terminal tab selected. The command `PS C:\Users\danielcc\Desktop\ DAM 1T\code\vscode\afond_1-4> docker build -t ejemplo-api:v1.0.0 ./` is being run. The output shows the build process starting with "[+] Building 10.2s (11/11) FINISHED" and listing various steps such as transferring the Dockerfile, loading metadata from docker.io, and extracting layers from the Docker image.

Imagen creada:



Comprobación de imagen, comando: `docker run -p 5000:5000 -e PORT=5000 -e MONGO_INITDB...`

```

PS C:\Users\danielcc\Desktop\DM 1T\code\vscode\afond_1-4> docker run -p 5000:5000 -e PORT=5000 -e MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=root -e MONGO_INITDB_DATABASE=bbdd_test -e MONGO_URI="mongodb://root:root@mongodb:27017/bbdd_test?authSource=admin" ejemplo-api:v1.0.0
> crudform-express@1.2.0 start
> node api/server.js

Fourella no porto: 5000

```

Ln 8, Col 19 Spaces: 4 UTF-8 CRLF

9.2. Archivo YAML

```

services:
  api:
    image: ejemplo-api:v1.0.0
    container_name: api-test
    restart: unless-stopped
    ports:
      - "5000:5000"
    environment:
      - MONGO_URI=mongodb://root:root@mongodb:27017/bbdd_test?authSource=admin
    depends_on:
      - mongodb
  networks:
    - app-network

  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: root
      MONGO_INITDB_DATABASE: bbdd_test
    volumes:
      - mongodb_data:/data/db
    networks:
      - app-network

volumes:
  mongodb_data:
    driver: local

networks:
  app-network:
    driver: bridge

```

Importante:

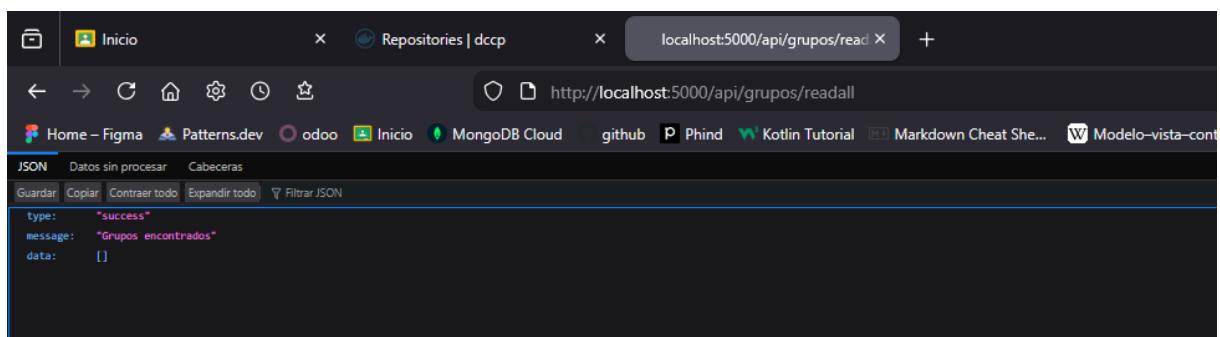
- **image: ejemplo-api:v1.0.0:** El nombre de nuestra imagen local.
- **ports: - "5000:5000" :** puerto de la API.

- environment: - MONGO_URI=mongodb://root:root@mongodb:27017/bbdd_test?authSource=admin
- variables de entorno.

Levantar contenedor: docker-compose up -d

```
PS C:\Users\danielcc\Desktop\DM 1T\code\vscode\afond_1-4> docker-compose up -d
✓ 46a33c43d8ef Pull complete
✓ a0ca46313493 Pull complete
✓ 9d906841bd13 Pull complete
✓ eef62aa7c052 Pull complete
✓ d97335354a5b Pull complete
✓ 20043066d3d5 Pull complete
✓ fdc0018776b0 Pull complete
✓ a801c4a80002 Pull complete
[+] Running 4/4
✓ Network afond_1-4_app-network    Created
✓ Volume "afond_1-4_mongodb_data"  Created
✓ Container mongodb                Started
✓ Container api-test               Started
○ PS C:\Users\danielcc\Desktop\DM 1T\code\vscode\afond_1-4>
```

Éxito:



9.3. Pushear imagen a Dockerhub

Build: docker build -t dccp/ejemplo-api:v1.0.0 ./ Nombre de usuario, nombre de imagen, tag, ruta.

Push: docker push dccp/ejemplo-api:v1.0.0 Nombre de usuario, nombre de imagen, tag.

Resultado del pusheado:

The screenshot shows the Docker Hub interface for the repository `dccp/ejemplo-api`. The repository has two tags: `latest` and `v1.0.0`. Both tags were pushed 9 days ago and pulled 9 days ago. The `v1.0.0` tag was pushed 18 days ago. A sidebar for `buildcloud` is visible, advertising Docker Build Cloud.

Uso de la imagen pusheado en el `docker-compose.yml`:

```
docker-compose.yml
 1  services:
 2    api:
 3      image: dccp/ejemplo-api:v1.0.0
 4      container_name: api-test
 5      restart: unless-stopped
 6      ports:
 7        - "5000:5000"
 8      environment:
```

Despues de `docker-compose up -d`, resultado en Docker Desktop:

Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
practicaododocker	-	-	-	0%	26 days ago	
afond_1-4	-	-	-	0.46%	1 minute ago	
mongodb	79130baa3461	mongo:latest	27017:27017	0.46%	5 minutes ago	
api-test	3c3b7f19790a	dccp/ejemplo-api:v1.0.0	5000:5000	0%	1 minute ago	

Funcionamiento correcto:

The screenshot shows the Visual Studio Code interface with several tabs open: 'readme.tex I, M', 'localhost:5000/auxfuncts/...', and 'docker-compose.yml M'. In the center, there's a 'Send' button and a 'Response' tab displaying a JSON object. The response content is as follows:

```
1  {
2    "type": "success",
3    "message": "Usuarios encontrados",
4    "data": [
5      {
6        "id": "690f516edf958c30de8a38d",
7        "nombre": "Carlos",
8        "apellido1": "Martinez",
9        "apellido2": "Rodriguez",
10       "edad": 35,
11       "_v": 0
12     }
13   ]
14 }
```

Below the main editor area, the 'PROBLEMS' and 'OUTPUT' tabs are visible. The 'OUTPUT' tab shows the command 'docker-compose up -d' being run in a PowerShell terminal, with logs indicating container creation and starting processes. The 'TERMINAL' tab is also visible.

```
st:String) [NewItem], IOException
+ FullyQualifiedId : DirectoryExist,Microsoft.PowerShell.Commands.NewItemCommand
PS C:\Users\dc-cr\Desktop\DM 2\code\vscode\afond_1-4> docker-compose up -d
✓ api Pulled
[+] Running 2/3
✓ Network afond_1-4_app-network Created
✓ Container mongodb Started
- Container api-test Starting
Error response from daemon: ports are not available: exposing port TCP 0.0.0.0:5000 -> 127.0.0.1:0: listen tcp 0.0.0.0:5000: bind: Only one usage of each socket address (protocol/network address/port) is normally permitted.
PS C:\Users\dc-cr\Desktop\DM 2\code\vscode\afond_1-4>
```

10. Github Actions

[Volver](#)

Con este archivo `yml` creado en `.github/workflows` nos permite usar ese workflow manualmente desde el repositorio además de cuando se hace un push:

```
name: Build and Push Push Push

on:
  push:
    branches:
      - main
      - master
    paths:
      - 'api/**'
      - '.github/workflows/docker-push.yml'
  workflow_dispatch: # Permite ejecutar manualmente

env:
  DOCKER_IMAGE_NAME: ${{ secrets.DOCKER_USERNAME }}/ejemplo-api
  DOCKER_TAG: latest

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout código
        uses: actions/checkout@v4

      - name: Configurar Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login a Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Construir y subir imagen Docker
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}
          cache-from: type=registry,ref=${{ env.DOCKER_IMAGE_NAME }}:buildcache
```

```

cache-to: type=inline

- name: Mostrar información de la imagen
  run: |
    echo "Imagen construida y subida exitosamente:"
    echo " - Imagen: ${{ env.DOCKER_IMAGE_NAME }}:${{ env.DOCKER_TAG }}"
    echo " - Docker Hub: https://hub.docker.com/r/${ secrets.DOCKER_USERNAME }"

```

Después tan sólo tenemos que configurar los `secrets` en el repo, `Settings`, `Secrets and Variables`. Además, ya que el archivo está configurado para acceder a DockerHub mediante token, generamos la token (`Write & Read`), recordando que hay que copiarla nada más crearla.

10.1. Notificacion al pushear

He intentado meterle notificación por email pero la cuenta alternativa de gmail que uso no me dejaba generar un App password, de manera que he usado las notificaciones nativas de Github.

```

notify-on-push:
  runs-on: ubuntu-latest
  needs: build-and-push
  if: always()
  steps:
    - name: Notify on Success
      if: needs.build-and-push.result == 'success'
      run: |
        echo "Docker image built and pushed successfully!"
        echo "Image: ${ env.DOCKER_IMAGE_NAME }:${ env.DOCKER_TAG }"

    - name: Notify on Failure
      if: needs.build-and-push.result == 'failure'
      run: |
        echo "Docker build failed!"
        exit 1

```

La notificación puede verse en el repositorio, la pestaña de Actions, clickeamos el último commit, click en `notify-on-push`.