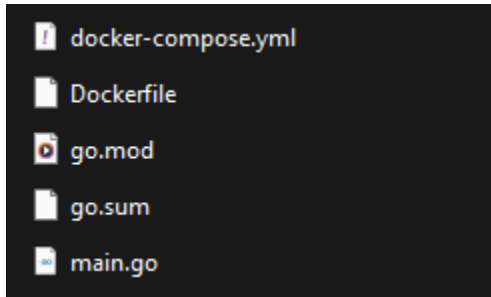


Documentación de uso – API REST de Gestión de Usuarios

Tecnologías empleadas

- Lenguaje: Go (Golang)
- Framework: Gorilla Mux (router HTTP)
- Base de datos: PostgreSQL
- Contenedores: Docker + Docker Compose
- Cliente sugerido para pruebas: Postman o curl

Estructura de la solución



Credenciales por defecto (en entorno local)

| Host = `go_db`

| Usuario = `postgres`

| Contraseña = `postgres`

| Base de datos = `go_crud_live`

Ejecución del proyecto

Requisitos

- Docker y Docker Compose instalados
- Sistema operativo: Windows, macOS o Linux

Pasos

1. Clonar o descomprimir el repositorio en el equipo local.
2. Abrir terminal en la raíz del proyecto.
3. Ejecutar:

docker compose up --build

4. Una vez desplegado, la API estará disponible en: <http://localhost:8080>

Funcionalidades

- Consultar todos los usuarios
- Consultar un usuario por ID
- Crear un nuevo usuario
- Actualizar información de un usuario
- Eliminar un usuario

Endpoint base

<http://localhost:8080>

Endpoints disponibles

- GET /users

Obtener la lista de todos los usuarios registrados.

- GET /users/{id}

Obtener la información de un usuario específico por su ID.

- POST /users

Crear un nuevo usuario en la base de datos. Se requiere enviar un cuerpo en formato JSON como el siguiente:

```
{  
  "username": "natalia123",  
  "email": "natalia@example.com"  
}
```

- PUT /users/{id}

Actualizar los datos de un usuario existente. Se requiere enviar un cuerpo en formato JSON como el siguiente:

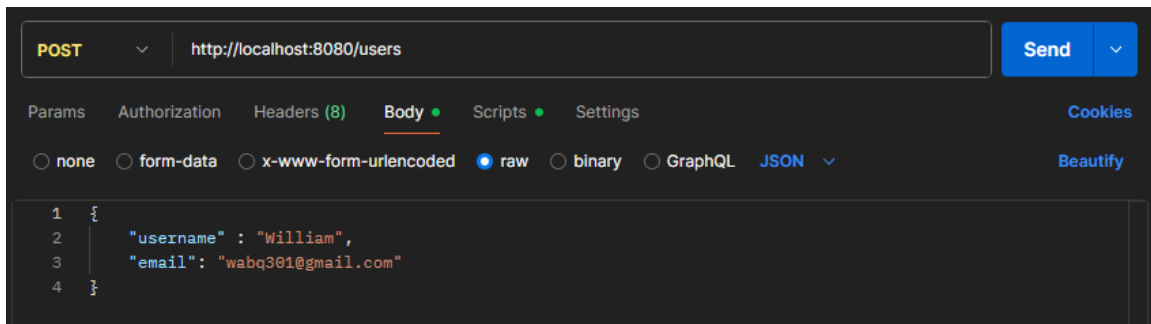
```
{
  "username": "usuario_actualizado",
  "email": "nuevo@example.com"
}
```

- DELETE /users/{id}

Eliminar un usuario existente por su ID.

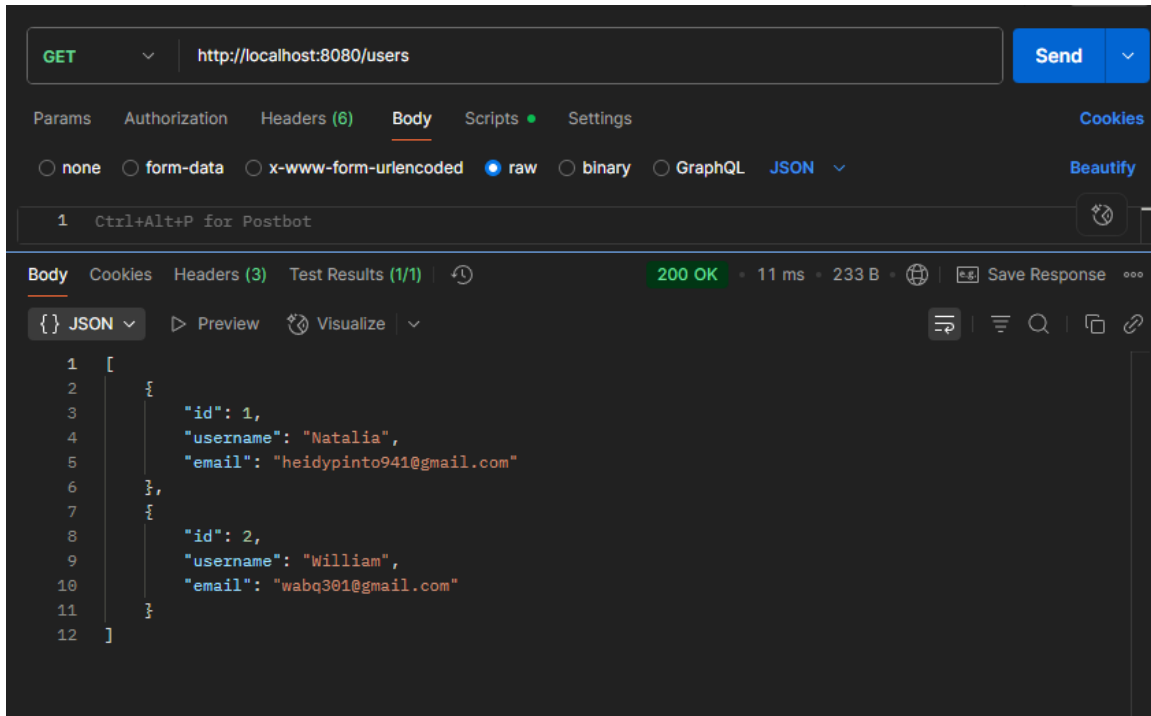
Ejemplos específicos con Postman

POST



```
{
  "id": 2,
  "username": "William",
  "email": "wabq301@gmail.com"
}
```

GET



GET ▼ http://localhost:8080/users Send ▼

Params Authorization Headers (6) **Body** Scripts ● Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼ Beautify

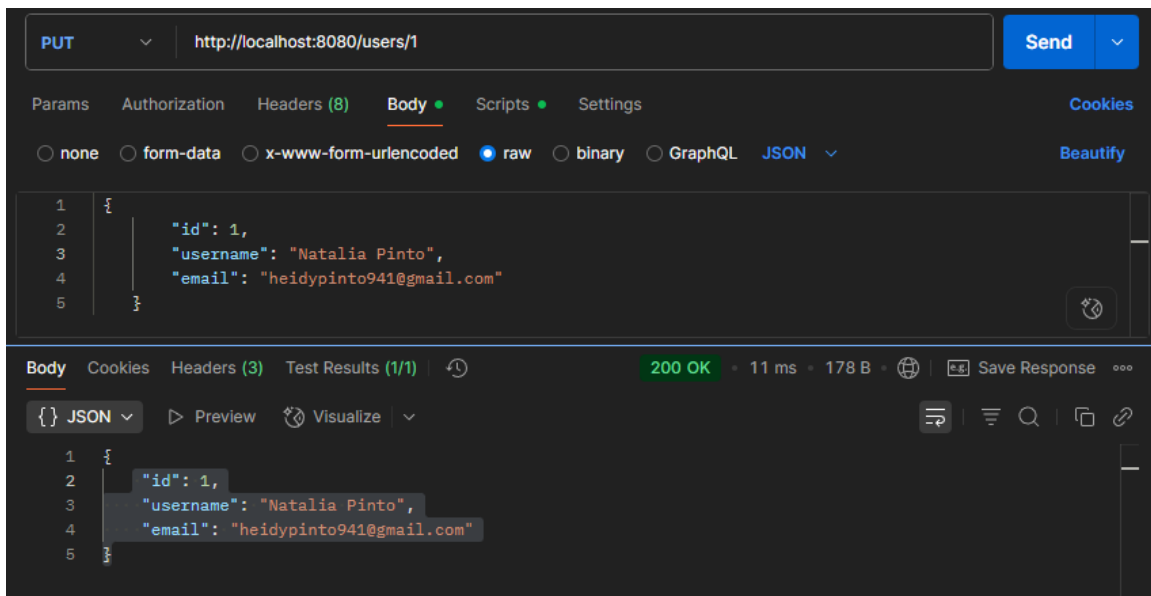
1 Ctrl+Alt+P for Postbot

Body Cookies Headers (3) Test Results (1/1) 🕒 200 OK 11 ms 233 B 🌐 📄 Save Response ⋮

{} JSON ▼ ▶ Preview 🔗 Visualize ▼ 🔍 📄 🔗

```
1  [
2    {
3      "id": 1,
4      "username": "Natalia",
5      "email": "heidypinto941@gmail.com"
6    },
7    {
8      "id": 2,
9      "username": "William",
10     "email": "wabq301@gmail.com"
11   }
12 ]
```

PUT (Se actualiza Username: Natalia a “Natalia Pinto”)



PUT ▼ http://localhost:8080/users/1 Send ▼

Params Authorization Headers (8) **Body** ● Scripts ● Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼ Beautify

```
1  {
2    "id": 1,
3    "username": "Natalia Pinto",
4    "email": "heidypinto941@gmail.com"
5  }
```

Body Cookies Headers (3) Test Results (1/1) 🕒 200 OK 11 ms 178 B 🌐 📄 Save Response ⋮

{} JSON ▼ ▶ Preview 🔗 Visualize ▼ 🔍 📄 🔗

```
1  {
2    "id": 1,
3    "username": "Natalia Pinto",
4    "email": "heidypinto941@gmail.com"
5  }
```

DELETE

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8080/users/1
- Buttons:** Send
- Tabs:** Params, Authorization, Headers (8), Body, Scripts, Settings, Cookies
- Body Type:** raw (selected), none, form-data, x-www-form-urlencoded, binary, GraphQL
- Body Content:** (Empty)
- Response Status:** 204 No Content
- Response Time:** 6 ms
- Response Size:** 96 B
- Response Headers:** (Empty)
- Response Body:** (Empty)
- Buttons:** Save Response, Beautify