

# 1. Introducción del Proyecto

Inicialmente, consideramos dockerizar una aplicación sencilla de React (basada en Create React App), por familiaridad con HTML, JavaScript y React. Sin embargo, al avanzar con las consignas del trabajo práctico, notamos que al no contar con una base de datos, no sería posible cumplir con todos los requisitos del TP.

Por eso, decidimos cambiar de rumbo y dockerizar un proyecto que ya conocíamos en profundidad: una aplicación desarrollada por nosotras en la materia Arquitectura de Software 1. Esta aplicación cuenta con:

- **Frontend:** desarrollado con React + Vite + JavaScript.
- **Backend:** desarrollado en GoLang.
- **Base de datos:** SQL (utilizamos MySQL según configuración).
- **Objetivo:** Dockerizar la aplicación completa y prepararla para su despliegue en entornos QA y Producción.

## 2. Proceso de Dockerización

### Preparación del Entorno Local

1. **Clonación del Repositorio:** Clonamos el repositorio del proyecto en una carpeta local llamada `tp2-docker`.
2. **Instalación de Dependencias:** Nos dirigimos a la carpeta del frontend y ejecutamos `npm install` para instalar todas las dependencias del proyecto.
3. **Verificación de la Aplicación:** Corrimos en la carpeta del backend `go run main.go` no sin antes haber declarado las variables de entorno como `DB_USER`, `DB_PASSWORD`, `DB_HOST`, `DB_PORT` y `DB_NAME`. Estas pueden definirse en un archivo `.env` o directamente en la terminal antes de ejecutar el comando.

```
sofis@DellOlibia MINGW64 ~/IngSoft3-Cervellini-Oliveto/TP2-Docker/backend (master)
$ export DB_USER=root
export DB_PASSWORD=44898366
export DB_HOST=localhost
export DB_PORT=3306
export DB_NAME=dbarquisoft1
go run main.go
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
- using env:   export GIN_MODE=release
- using code:  gin.SetMode(gin.ReleaseMode)

Connection Established
Finishing Migration Database Tables
```

Y posteriormente en la carpeta del frontend ejecutamos `npm run dev` consiguiendo levantar el back en el puerto 8080 y el front en el navegador (por defecto en <http://localhost:5173>).

```
sofis@DellOlibia MINGW64 ~/IngSoft3-Cervellini-Oliveto/TP2-Docker/frontend (master)
$ npm run dev

> frontend@0.0.0 dev
> vite

VITE v5.2.13 ready in 551 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

4. **Configuración de Docker Hub:** Nos aseguramos de tener Docker Desktop o el entorno de Docker funcionando y luego iniciamos sesión en Docker Hub con el comando `docker login` usando nuestro usuario y contraseña.

```
sofis@DellOlibia MINGW64 ~/IngSoft3-Cervellini-Oliveto/TP2-Docker (master)
$ docker login
Authenticating with existing credentials... [Username: sofioliveto]

Info → To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded
```

## Creación del Dockerfile

Luego de validar el funcionamiento local, procedimos a construir los Dockerfiles del backend y frontend para su posterior dockerización y despliegue (por separado y luego orquestados con Docker Compose).

Dockerfile de /backend

Etapa de construcción	
FROM golang:1.21-alpine AS builder	Esta imagen incluye el compilador Go y herramientas necesarias para construir el binario. Se elige la variante Alpine por ser liviana.
WORKDIR /app	Organiza los archivos dentro del contenedor.
COPY . .	Copia todos los archivos del proyecto.

<code>RUN go mod download</code>	Instala las dependencias definidas en go.mod.
<code>RUN go build -o main .</code>	Compila la app Go en un binario llamado main.
Etapa de producción	
<code>FROM alpine:latest</code>	Es una imagen mínima que no incluye herramientas de desarrollo. Ideal para ejecutar el binario sin sobrecarga.
<code>COPY --from=builder</code>	Solo copiamos el binario, no el código fuente.
<code>EXPOSE 8080</code>	Indica que la app escucha en el puerto 8080.
<code>CMD [ "./main" ]</code>	Especifica cómo correr la app.

Dockerfile de /frontend

Etapa de construcción	
<code>FROM node:20-alpine</code>	Incluye Node.js y npm, necesarios para instalar dependencias y compilar el frontend. La variante Alpine reduce el peso.
<code>RUN npm install</code>	Instala dependencias del package.json.
<code>RUN npm run build</code>	Crea los archivos estáticos de producción (/dist).
Etapa de producción	
<code>FROM nginx:stable-alpine</code>	NGINX es un servidor web eficiente y ampliamente usado para servir archivos estáticos. La versión Alpine mantiene la imagen liviana.
<code>COPY --from=builder /app/dist</code>	Solo copiamos los archivos finales, no el código fuente.
<code>EXPOSE 80</code>	Expone el puerto del servidor web.

Multistage build: primero se construye el proyecto en Node y luego se sirve con Nginx. Nginx expone la app en el puerto 80.

La separación en etapas responde a una práctica recomendada llamada **multi-stage build**. Esta técnica permite:

- **Optimizar el tamaño de la imagen final:** eliminando herramientas innecesarias como compiladores o dependencias de desarrollo.

- **Mejorar la seguridad:** al no incluir el código fuente ni utilidades que podrían ser explotadas.
- **Acelerar despliegues:** las imágenes más livianas se transfieren y ejecutan más rápido.

## 3. Creación y Despliegue de la Imagen

### Construcción de la Imagen

Dentro de los directorios correspondientes construimos cada imagen con los siguientes comandos:

```
docker build -t sofioliveto/backend-app:dev .
```

- `docker build`: Es el comando para construir una imagen.
- `-t`: Es la opción para nombrar y etiquetar la imagen (tag). Le dimos el nombre `sofioliveto/backend-app` y la etiqueta `:dev` para indicar que es una versión de desarrollo.
- `.`: Indica que el Dockerfile se encuentra en el directorio actual.

```
sofis@DelloIolibia MINGW64 ~/IngSoft3-Cervellini-Oliveto/TP2-Docker/backend (master)
$ docker build -t sofioliveto/backend-app:dev .
[+] Building 204.2s (16/16) FINISHED                                docker:desktop-linux
=> [internal] load build definition from dockerfile                  0.4s
=> => transferring dockerfile: 668B                                0.2s
=> [internal] load metadata for docker.io/library/alpine:latest     5.6s
=> [internal] load metadata for docker.io/library/golang:1.21-alpine 5.8s
=> [auth] library/golang:pull token for registry-1.docker.io        0.0s
=> [auth] library/alpine:pull token for registry-1.docker.io        0.2s
=> [internal] load .dockerignore                                     0.2s
=> => transferring context: 2B                                       0.0s
=> [builder 1/5] FROM docker.io/library/golang:1.21-alpine@sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f 78.7s
=> => resolve docker.io/library/golang:1.21-alpine@sha256:2414035b086e3c42b99654c8b26e6f5b1b1598080d65fd03c7f409552ff4 0.2s
```

```
docker build -t sofioliveto/frontend-app:dev .
```

- `-t`: Es la opción para nombrar y etiquetar la imagen (tag). Le dimos el nombre `sofioliveto/frontend-app` y la etiqueta `:dev` para indicar que es una versión de desarrollo.

```
sofis@DelloIolibia MINGW64 ~/IngSoft3-Cervellini-Oliveto/TP2-Docker/frontend (master)
$ docker build -t sofioliveto/frontend-app:dev .
[+] Building 341.5s (15/15) FINISHED                                docker:desktop-linux
=> [internal] load build definition from dockerfile                  0.2s
=> => transferring dockerfile: 640B                                0.1s
=> [internal] load metadata for docker.io/library/nginx:stable-alpine 5.3s
=> [internal] load metadata for docker.io/library/node:20-alpine    5.7s
=> [auth] library/node:pull token for registry-1.docker.io          0.0s
=> [auth] library/nginx:pull token for registry-1.docker.io         0.0s
=> [internal] load .dockerignore                                     0.1s
=> => transferring context: 2B                                       0.0s
=> [stage-1 1/2] FROM docker.io/library/nginx:stable-alpine@sha256:8f2bcf97c473dfe311e79a510ee540ee02e28ce1e6a64e1ef8 26.6s
```

Pruebas de que las imágenes fueron creadas

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
sofiioliveto/frontend-app  dev                13929dd36378       22 minutes ago     49MB
sofiioliveto/backend-app   dev                53a140e5d153       32 minutes ago     27.6MB
```

## Subida al Repositorio

Finalmente, subimos las imágenes al repositorio de Docker Hub ejecutando:

```
docker push sofiioliveto/backend-app:dev
```

```
docker push sofiioliveto/frontend-app:dev
```

Pruebas de que las imágenes fueron subidas a docker hub

Name	Last Pushed <a href="#">↑</a>	Contains	Visibility
sofiioliveto/backend-app	3 minutes ago	IMAGE	Public
sofiioliveto/frontend-app	9 minutes ago	IMAGE	Public

## Estrategia de Versionado

Implementamos una estrategia de **etiquetado** para manejar las diferentes versiones de la imagen. Esto nos permite diferenciar entre etapas del ciclo de vida del software, como **desarrollo** y **producción**.

- **sofiioliveto/sofiioliveto/backend-app:dev**: Esta etiqueta indica una versión en desarrollo.
- **sofiioliveto/sofiioliveto/backend-app:v1.0**: Esta etiqueta marca una versión estable y lista para producción.

Este enfoque nos permite reutilizar la misma imagen para diferentes entornos (QA, producción, etc.) sin necesidad de reconstruirla cada vez.

Como por ahora nuestras imágenes están tagueadas con dev significan que aun no son estables como para pasar a producción.

## 4. Integrar una base de datos en contenedor

Se eligió MySQL porque es una base de datos con la cual estamos acostumbradas a trabajar, tiene excelente compatibilidad con GoLang a través del ORM GORM. La imagen oficial en Docker Hub (`mysql:8`) simplifica la configuración y el mantenimiento del entorno de base de datos.

La creación de los contenedores de mysql para QA y PROD con sus respectivos volúmenes para la persistencia de datos se hicieron en el `docker-compose.yml`.

La app se conecta a la db en backend/[db-clients.go](#) con

```
func ConnectDatabase() {
    dbUsername := os.Getenv("DB_USER")
    dbPassword := os.Getenv("DB_PASSWORD")
    dbHost := os.Getenv("DB_HOST")
    dbPort := os.Getenv("DB_PORT")
    dbSchema := os.Getenv("DB_NAME")
    dsn :=
"%s:%s@tcp(%s:%s)/%s?charset=utf8mb4&parseTime=True&loc=Local"
```

## 5. QA y PROD con misma imagen

El servicio backend-qa y backend-prod usan exactamente la misma imagen `sofiiooliveto/backend-app:dev`, garantizando así que el código fuente es idéntico en ambos entornos. La diferenciación se produce únicamente a través del uso de variables de entorno inyectadas por `docker-compose`:

- `ENVIRONMENT`: define si el contenedor está corriendo en modo qa o prod, útil para condicionar logs, modo de ejecución (debug vs release), o cargar configuraciones específicas.
- `DB_HOST`: apunta al servicio de base de datos correspondiente (`mysql-qa` o `mysql-prod`), lo cual reemplaza el uso de `localhost`.
- `DB_PORT`: en ambos casos se mantiene "3306" como puerto interno de MySQL.
- `DB_NAME`, `DB_USER`, `DB_PASSWORD`: valores personalizados por entorno, permitiendo a cada backend conectarse a su propia base.

## 6. Entorno reproducible con docker-compose

Para asegurar que el entorno de ejecución funcione de manera **idéntica en cualquier máquina**, se utilizó **Docker Compose**, una herramienta que permite definir y administrar múltiples contenedores desde un único archivo de configuración (`docker-compose.yml`).

### ¿Qué hace el `docker-compose.yml`?

El archivo actual levanta de forma automática:

- Dos bases de datos MySQL (mysql-qa y mysql-prod), cada una con su propio volumen persistente.
- Dos instancias del backend (backend-qa y backend-prod), usando la **misma imagen** pero distintas variables de entorno.
- Una instancia del frontend (frontend), que depende de ambos backends.

Para garantizar que los datos **no se pierdan al reiniciar contenedores**, se declararon volúmenes persistentes:

- mysqlqa\_data
- mysqlprod\_data

Se crearon redes qa y prod por separado, lo que permite aislar los entornos y evitar interferencias cruzadas, simulando un entorno real con distintas configuraciones de infraestructura.

## Evidencias de correcto funcionamiento

Se crea el contenedor con el docker-compose.yml

```
PS C:\Users\sofis\IngSoft3-Cervellini-Oliveto\TP2-Docker> docker-compose up -d
[+] Running 9/9
✓ Network tp2-docker_qa          Created
✓ Network tp2-docker_prod        Created
✓ Volume "tp2-docker_mysqlqa_data" Created
✓ Volume "tp2-docker_mysqlprod_data" Created
✓ Container mysql-prod           Healthy
✓ Container mysql-qa             Healthy
✓ Container backend-qa           Started
✓ Container backend-prod         Started
✓ Container frontend             Started
```

Verificamos que se crean los contenedores correspondientes

```
PS C:\Users\sofis\IngSoft3-Cervellini-Oliveto\TP2-Docker> docker-compose ps
NAME                IMAGE                COMMAND                SERVICE    CREATED        STATUS
backend-prod        sofioliveto/backend-app:dev  "./main"              backend-prod  About a minute ago  Up About a minute
0.0.0.0:8082->8080/tcp, [::]:8082->8080/tcp
backend-qa          sofioliveto/backend-app:dev  "./main"              backend-qa    About a minute ago  Up About a minute
0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
frontend            sofioliveto/frontend-app:dev  "/docker-entrypoint..." frontend      About a minute ago  Up About a minute
0.0.0.0:5173->80/tcp, [::]:5173->80/tcp
mysql-prod          mysql:8.4            "docker-entrypoint.s..." mysql-prod    About a minute ago  Up About a minute (health
0.0.0.0:3308->3306/tcp, [::]:3308->3306/tcp
mysql-qa            mysql:8.4            "docker-entrypoint.s..." mysql-qa      About a minute ago  Up About a minute (health
0.0.0.0:3307->3306/tcp, [::]:3307->3306/tcp
```

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Actions
<input type="checkbox"/>	tp2-docker	-	-	-	22.67%	
<input type="checkbox"/>	mysql-qa	75a2a4f21adf	mysql:8.4	3307:3306	11.37%	
<input type="checkbox"/>	mysql-prod	ad3fb5e59a9f	mysql:8.4	3308:3306	11.3%	
<input type="checkbox"/>	backend-qa	a0e6876dc529	sofioliveto	8081:8080	0%	
<input type="checkbox"/>	backend-prod	53007477b99e	sofioliveto	8082:8080	0%	
<input type="checkbox"/>	frontend	4f5ad50bd5c4	sofioliveto	5173:80	0%	

Probamos un endpoint en QA:8081 (no hay cursos cargados todavía)

GET
http://localhost:8081/search

Params
Authorization
Headers (6)
Body
Scripts
Settings

Body
Cookies
Headers (3)
Test Results

Pretty
Raw
Preview
Visualize
JSON

```

1 null

```

Probamos crear un curso en QA:8081

POST
http://localhost:8081/createCourse

Params
Authorization
Headers (8)
Body
Scripts
Settings

none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON

```

1 {
2   "course_id": 2,
3   "nombre": "Bartender ricos tragos",
4   "profesor_id": 2,
5   "categoria": "Cocteleria",
6   "descripcion": "Aprende a hacer los mejores tragos",

```

Body
Cookies
Headers (3)
Test Results

201 Created

Pretty
Raw
Preview
Visualize
JSON

```

1 {
2   "course_id": 2,
3   "nombre": "Bartender ricos tragos",
4   "profesor_id": 2,
5   "categoria": "Cocteleria",
6   "descripcion": "Aprende a hacer los mejores tragos",
7   "valoracion": 3.7,
8   "duracion": 45,
9   "requisitos": "Bajo",
10  "url_image": "",
11  "fecha_inicio": "2024-06-03T00:00:00Z"
12 }

```

Comprobamos que el curso se creó correctamente con el endpoint de search que antes nos devolvió null



```
GET http://localhost:8081/search

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (3) Test Results 200 OK

Pretty Raw Preview Visualize JSON

1 [
2   {
3     "course_id": 2,
4     "nombre": "Bartender ricos tragos",
5     "profesor_id": 2,
6     "categoria": "Cocteleria",
7     "descripcion": "Aprende a hacer los mejores tragos",
8     "valoracion": 4,
9     "duracion": 45,
10    "requisitos": "Bajo",
11    "url_image": "",
12    "fecha_inicio": "2024-06-03T00:00:00Z"
13  }
14 ]
```

Probamos el mismo endpoint en PROD:8080 (no hay cursos cargados todavía)

```
GET http://localhost:8082/search

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (3) Test Results 200 OK

Pretty Raw Preview Visualize JSON

1 null
```

Probamos crear un curso en PROD:8082

```
POST http://localhost:8082/createCourse

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

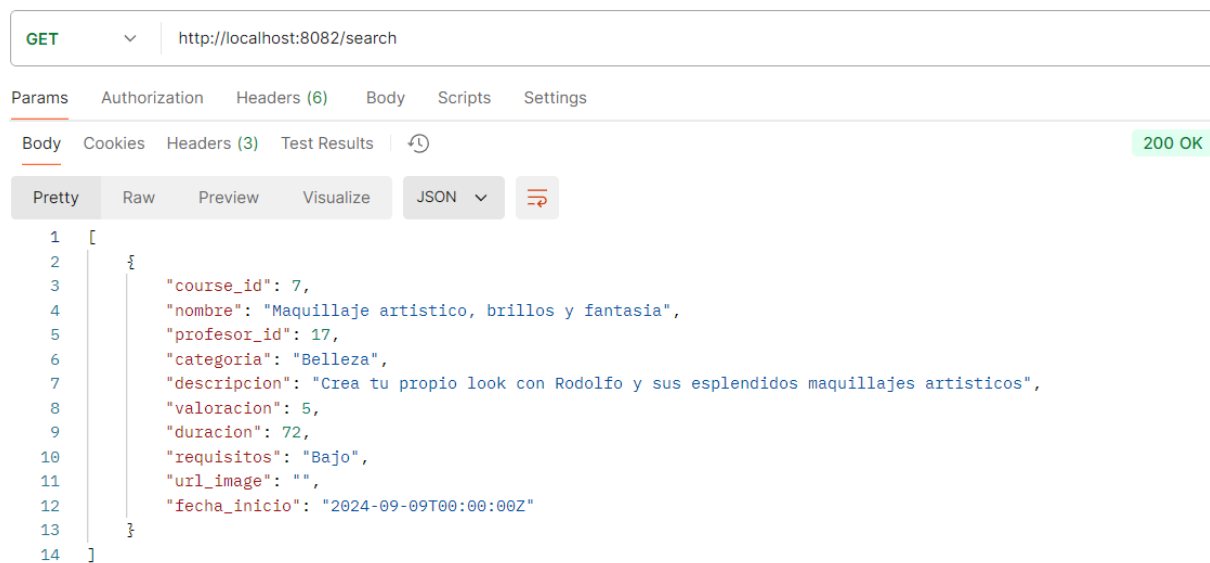
1 {
2   "course_id": 7,
3   "nombre": "Maquillaje artistico, brillos y fantasia",
4   "profesor_id": 17,
5   "categoria": "Belleza",
6   "descripcion": "Crea tu propio look con Rodolfo y sus esplendidos maquillajes artisticos",
7   "valoracion": 4.8,
8   "duracion": 72,
9   "requisitos": "Bajo",
10  "url_image": "",
11  "fecha_inicio": "2024-09-09T00:00:00Z"
12 }
```

```
Body Cookies Headers (3) Test Results 201 Created

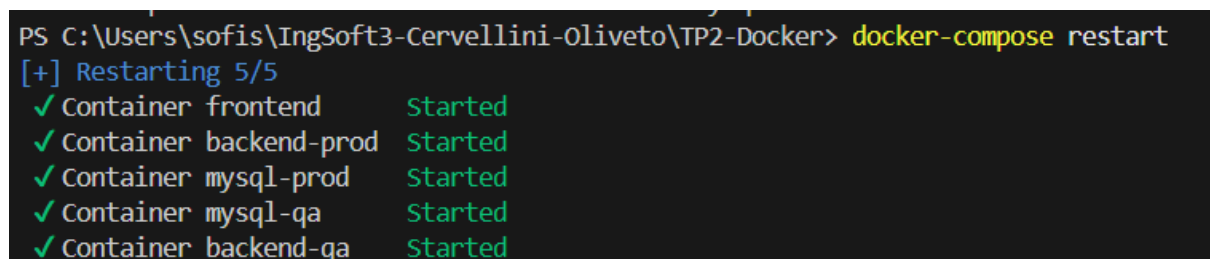
Pretty Raw Preview Visualize JSON

1 {
2   "course_id": 7,
3   "nombre": "Maquillaje artistico, brillos y fantasia",
4   "profesor_id": 17,
5   "categoria": "Belleza",
6   "descripcion": "Crea tu propio look con Rodolfo y sus esplendidos maquillajes artisticos",
7   "valoracion": 4.8,
8   "duracion": 72,
9   "requisitos": "Bajo",
10  "url_image": "",
11  "fecha_inicio": "2024-09-09T00:00:00Z"
12 }
```

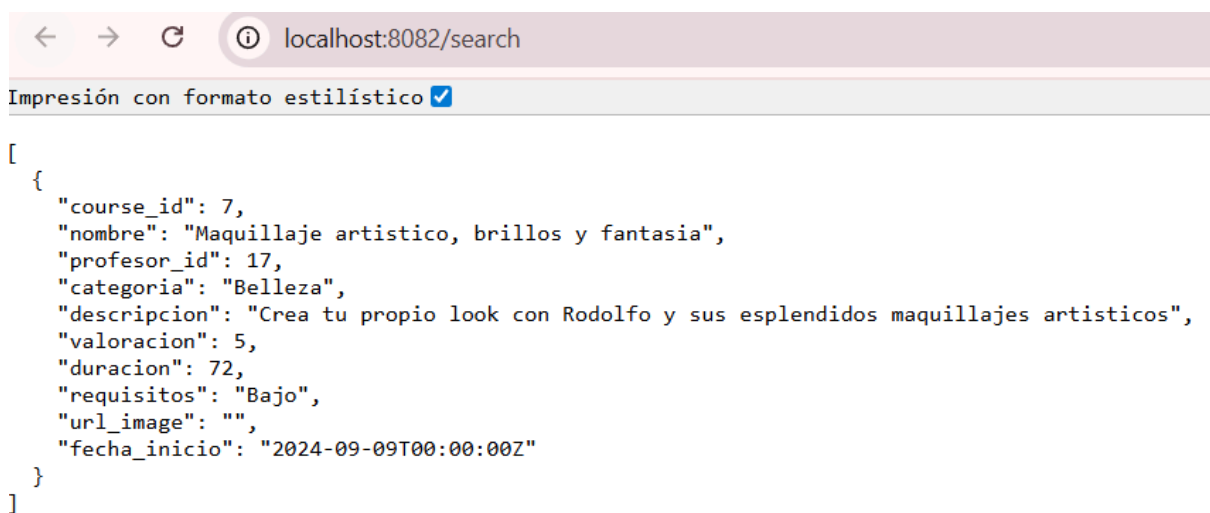
Comprobamos que el curso se creó correctamente con el endpoint de search que antes nos devolvió null



Comprobamos la persistencia de datos gracias a los volúmenes al reiniciar los contenedores



Se puede ver que el curso creado anteriormente persiste y además el curso de PROD no se ve en QA



Lo mismo para PROD

```
localhost:8081/search

Impresión con formato estilístico ☒

[
  {
    "course_id": 2,
    "nombre": "Bartender ricos tragos",
    "profesor_id": 2,
    "categoria": "Cocteleria",
    "descripcion": "Aprende a hacer los mejores tragos",
    "valoracion": 4,
    "duracion": 45,
    "requisitos": "Bajo",
    "url_image": "",
    "fecha_inicio": "2024-06-03T00:00:00Z"
  }
]
```

## 7. Versionado de imágenes

Una vez probada localmente la imagen `:dev`, se creó una versión estable y etiquetada como `v1.0`. Para ello, se ejecutó:

- Desde el backend:  
`docker tag sofiioliveto/backend-app:dev sofiioliveto/backend-app:v1.0`  
`docker push sofiioliveto/backend-app:v1.0`
- Desde el frontend:  
`docker tag sofiioliveto/frontend-app:dev sofiioliveto/frontend-app:v1.0`  
`docker push sofiioliveto/frontend-app:v1.0`











Local

My Hub

sofioliveto

Search

[View Docker Scout](#)

	Tags	OS	Vulnerabilities	Last pushed	Size
 sofioliveto/frontend-app	v1.0		 Inactive	6 seconds ago	21.32 MB
	dev		 Inactive	17 hours ago	21.32 MB
 sofioliveto/backend-app	v1.0		 Inactive	2 minutes ago	14.05 MB
	dev		 Inactive	17 hours ago	14.05 MB

Luego de que queden las imágenes subidas en docker hub con los nuevos tags actualizamos el `docker-compose.yml` para que se utilicen.

## Estrategia de versionado

Se utiliza la convención **SemVer (Semantic Versioning)** para etiquetar las imágenes Docker, que sigue el formato MAJOR.MINOR.PATCH, por ejemplo:

- v1.0.0: primera versión estable
- v1.1.0: nueva funcionalidad compatible hacia atrás
- v2.0.0: cambio incompatible o reestructuración mayor

En este trabajo se utiliza v1.0 como primer release estable, luego de haber probado con la etiqueta :dev en entorno local y en QA/PROD por separado.