



**UNIVERSIDAD  
CATÓLICA  
DE CÓRDOBA**  
JESUITAS

**Ingeniería en Software 3  
TRABAJO PRÁCTICO N°4**

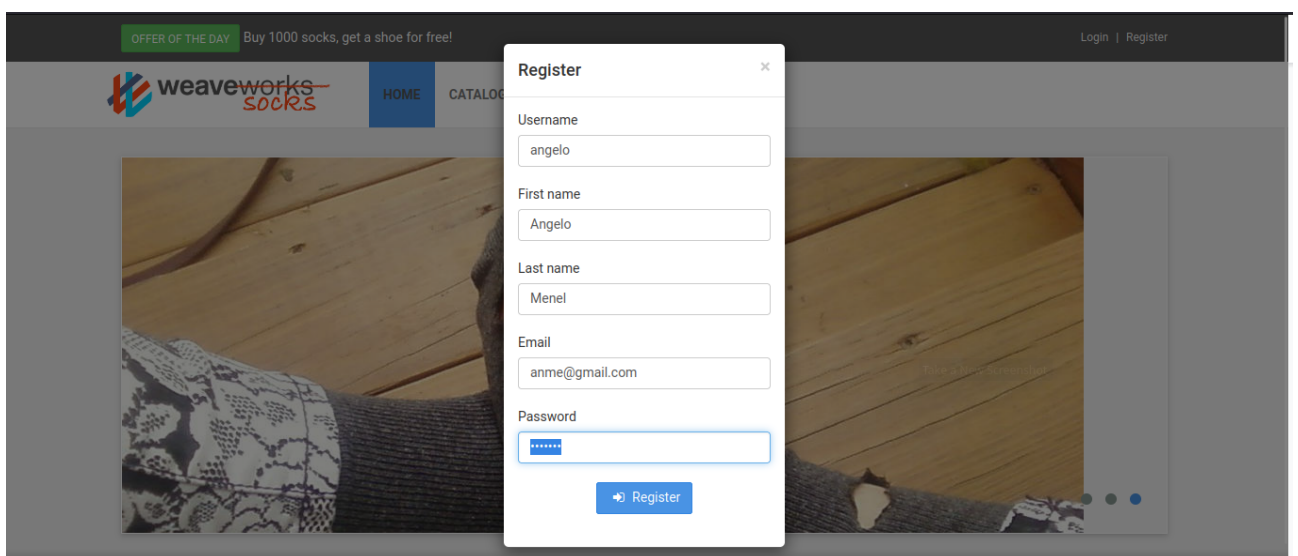
**Profesor:** Ing. Bono Fernando

**Alumno:** Menel Angelo (1804789)

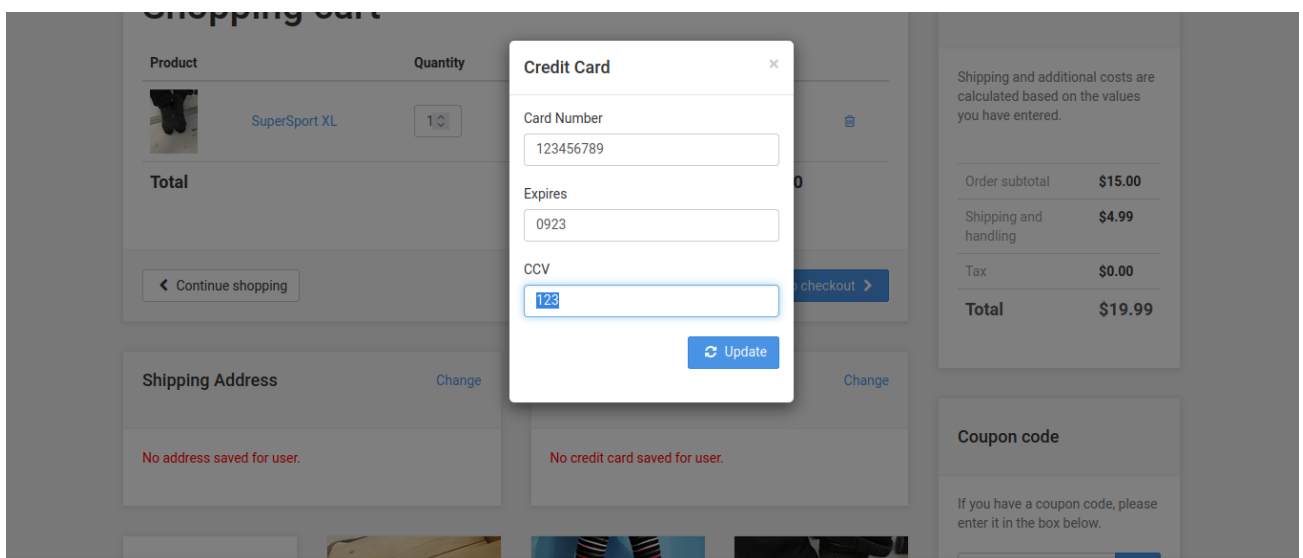
## Ejercicio 1

```
~$ mkdir -p socks-demo
~$ cd socks-demo
~/socks-demo$ git clone https://github.com/microservices-demo/microservices-demo.git
Cloning into 'microservices-demo'...
remote: Enumerating objects: 10197, done.
remote: Total 10197 (delta 0), reused 0 (delta 0), pack-reused 10197
Receiving objects: 100% (10197/10197), 52.95 MiB | 945.00 KiB/s, done.
Resolving deltas: 100% (6208/6208), done.
~/socks-demo$ cd microservices-demo
~/soc/microservices-demo$ docker-compose -f deploy/docker-compose/docker-compose.yml up -d
```

- Accedemos la localhost:80 y nos muestra la aplicación web, donde nos registramos:



- Desde la opción del carrito de compras, registramos la tarjeta de crédito:



## Ejercicio 2

El sistema tiene los siguientes contenedores:

- **front-end:** Aplicación web escrita en Node.js el cual pone todos los microservicios juntos.
- **edge-router:** Api gateway que mediante cap\_add redirecciona a catalogue, carts, orders, payment y user. Utiliza el puerto 80 (modo visualización) y el 8080 (modo monitoreo). Este es el punto de ingreso del sistema.
- **catalogue:** Servicio que provee la información del catálogo/producto.
- **catalogue-db:** Posee todos los datos del catálogo en la base de datos. El nombre de la base de datos es socksdb, la contraseña es la contraseña root y permite una contraseña null o vacía.
- **carts:** Servicio que provee el carrito de compras para los usuarios.
- **carts-db:** Posee todos los datos del carrito de compras en la base de datos. Utiliza una imagen de una base de datos no relacional mongo.
- **orders:** Servicio que provee las capacidades de pedidos.
- **orders-db:** Posee todos los datos de las órdenes en la base de datos. Utiliza una imagen de una base de datos no relacional mongo.
- **Shipping:** Servicio que provee las capacidades de envío.
- **queue-master:** Servicio que procesa las colas. Implementa la cola de envíos y simula el proceso de envío.
- **rabbitmq:** Es un sistema de manejo de colas. El sistema a través de este carga un mensaje a través de la cola de envíos.
- **Payment:** Servicio que provee los métodos de pago.
- **User:** Servicio que contiene el almacenamiento de las cuentas de usuario e incluye tarjetas y direcciones asociadas al usuario.
- **user-db:** Posee todos los datos de los usuarios en la base de datos.
- **user-sim:** Posee los datos de un usuario test que es usado por el desarrollador para probar o testear funcionalidades.

## Ejercicio 3

Se están utilizando repositorios separados para el código y/o la configuración del sistema ya que cada uno es un servicio diferente y realizan funciones distintas, cada uno es un código base independiente y estos servicios se comunican entre sí mediante APIs bien definidas. Los detalles de la implementación interna de cada servicio se ocultan frente a los otros.

### Ventajas:

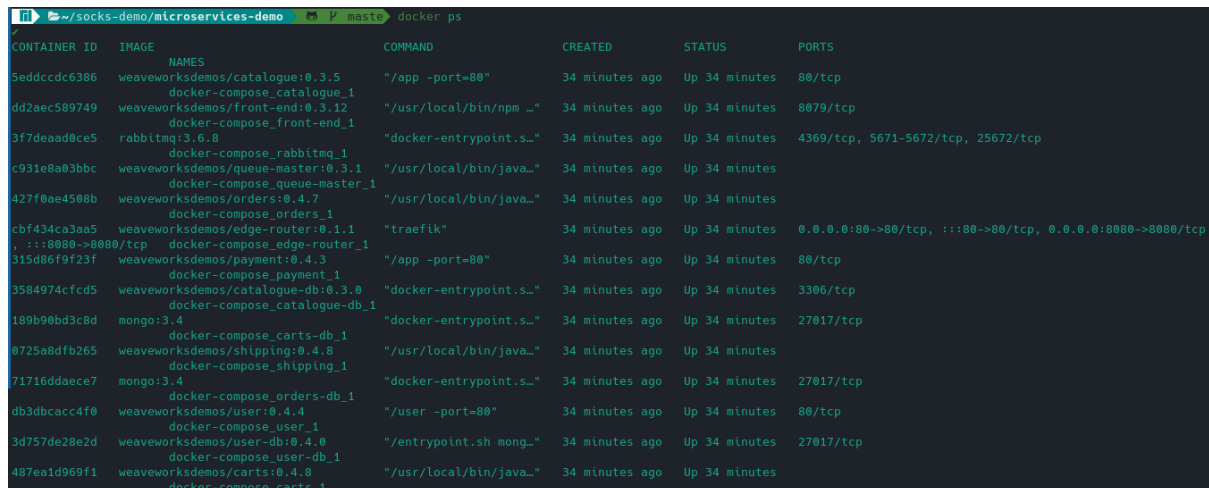
- Equipo de trabajo mínimo
- Escalabilidad
- Funcionalidad modular, módulos independientes.
- Libertad del desarrollador de desarrollar y desplegar servicios de forma independiente.
- Uso de contenedores permitiendo el despliegue y el desarrollo de la aplicación rápidamente.
- Independizamos la lógica, también en algunos casos lenguajes.

## Desventajas:

- Mayor consumo de recursos. Puesto que cada microservicio tiene su propio Sistema Operativo y dependencias, al final sale más caro a nivel de recursos usar microservicios que un monolito; que es un Sistema Operativo, y sus dependencias. Para solucionar esto usamos docker.
- La complejidad del despliegue en sí. Es muy difícil gestionar microservicios.
- Hay que lidiar con la complejidad adicional de los sistemas distribuidos. Implementar comunicación interna entre los servicios, implementar dependencias de un servicio hacia el otro, solicitudes que pueden extenderse a varios servicios, etc. Básicamente la configuración inicial.

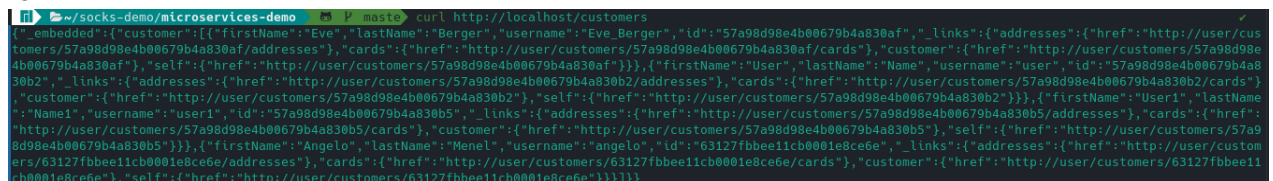
## Ejercicio 4

En este caso, el contenedor que hace las veces de API Gateway es edge-router. Traefik es la herramienta de API Gateway el cual se ingresa mediante localhost:8080. Para saber esto hice un docker ps y el único puerto el cual estaba escuchando en el 80 y 8080 es el contenedor edge-router.



CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
5eddccdc6386	weaveworksdemos/catalogue:0.3.5	docker-compose_catalogue_1	"/app -port=80"	34 minutes ago	Up 34 minutes	80/tcp
dd2aec589749	weaveworksdemos/front-end:0.3.12	docker-compose_front-end_1	"/usr/local/bin/npm ..."	34 minutes ago	Up 34 minutes	8079/tcp
3f7deaad0ce5	rabbitmq:3.6.8	docker-compose_rabbitmq_1	"docker-entrypoint.s..."	34 minutes ago	Up 34 minutes	4369/tcp, 5671-5672/tcp, 25672/tcp
c931e8a03bbc	weaveworksdemos/queue-master:0.3.1	docker-compose_queue-master_1	"/usr/local/bin/java..."	34 minutes ago	Up 34 minutes	
427f0ae4508b	weaveworksdemos/orders:0.4.7	docker-compose_orders_1	"/usr/local/bin/java..."	34 minutes ago	Up 34 minutes	
cbf434ca3aa5	weaveworksdemos/edge-router:0.1.1	traefik	"traefik"	34 minutes ago	Up 34 minutes	0.0.0.0:80->80/tcp, ::80->80/tcp, 0.0.0.0:8080->8080/tcp
315d86f9f23f	weaveworksdemos/payment:0.4.3	docker-compose_payment_1	"/app -port=80"	34 minutes ago	Up 34 minutes	80/tcp
3584974cfcd5	weaveworksdemos/catalogue-db:0.3.0	docker-compose_catalogue-db_1	"docker-entrypoint.s..."	34 minutes ago	Up 34 minutes	3306/tcp
189b90bd3c8d	mongo:3.4	docker-compose_carts-db_1	"docker-entrypoint.s..."	34 minutes ago	Up 34 minutes	27017/tcp
0725a8dfb265	weaveworksdemos/shipping:0.4.8	docker-compose_shipping_1	"/usr/local/bin/java..."	34 minutes ago	Up 34 minutes	
71716ddaec7	mongo:3.4	docker-compose_orders-db_1	"docker-entrypoint.s..."	34 minutes ago	Up 34 minutes	27017/tcp
db3dbacc4f0	weaveworksdemos/user:0.4.4	docker-compose_user_1	"/user -port=80"	34 minutes ago	Up 34 minutes	80/tcp
3d757de282d	weaveworksdemos/user-db:0.4.0	docker-compose_user-db_1	"/entrypoint.sh mong..."	34 minutes ago	Up 34 minutes	27017/tcp
487ea1d969f1	weaveworksdemos/carts:0.4.8	docker-compose_carts_1	"/usr/local/bin/java..."	34 minutes ago	Up 34 minutes	

## Ejercicio 5



```
curl http://localhost/customers
{"_embedded":{"customer":[{"first_name":"Eve","last_name":"Berger","username":"Eve_Berger","id":"57a98d98e4b00679b4a830af","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830af/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830af/cards"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830af"}}}],"first_name":"User","last_name":"Name","username":"user","id":"57a98d98e4b00679b4a830b2","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830b2/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830b2/cards"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830b2"},"first_name":"User1","last_name":"Name1","username":"user1","id":"57a98d98e4b00679b4a830b5","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830b5/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830b5/cards"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830b5"},"first_name":"Angelo","last_name":"Menel","username":"angelo","id":"63127fbbee11cb0001e8ce6e","_links":{"addresses":{"href":"http://user/customers/63127fbbee11cb0001e8ce6e/addresses"},"cards":{"href":"http://user/customers/63127fbbee11cb0001e8ce6e/cards"},"self":{"href":"http://user/customers/63127fbbee11cb0001e8ce6e"}}}]}}
```

- Nos devuelve un json de todos los clientes que fueron registrados en la aplicación, con todos los datos de cada usuario.

## Ejercicio 6

- El endpoint de customers se encuentra dentro del servicio de user.

## Ejercicio 7

- El endpoint catalogue se encuentra dentro del servicio catalogue.
- El endpoint tags se encuentra dentro del servicio catalogue.

**Ejercicio 8**

- Los datos se persisten en diferentes bases de datos, cada servicio tiene su propia base de datos. Por lo que si borramos un servicio y lo clonamos de vuelta, tienen los datos originales.

**Ejercicio 9**

- El componente encargado del procesamiento de la cola de mensajes es rabbitmq.

**Ejercicio 10**

- El tipo de interfaz que utilizan estos microservicios para comunicarse es del tipo API REST