

Trabajo Práctico 4

Materia: Ingeniería de Software 3

Alumno: Vietto Santiago

Docente: Fernando Bono

Institución: UCC

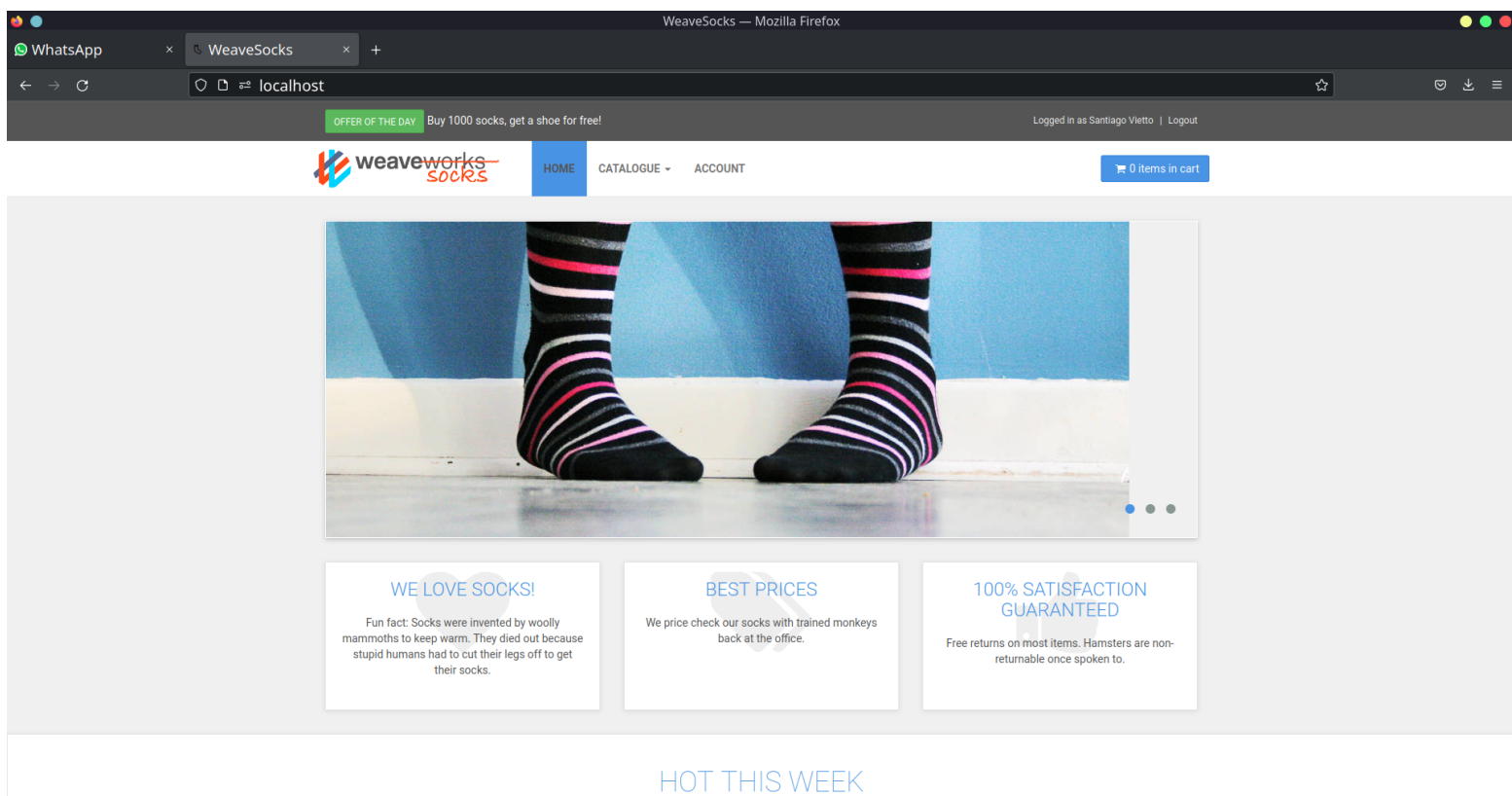
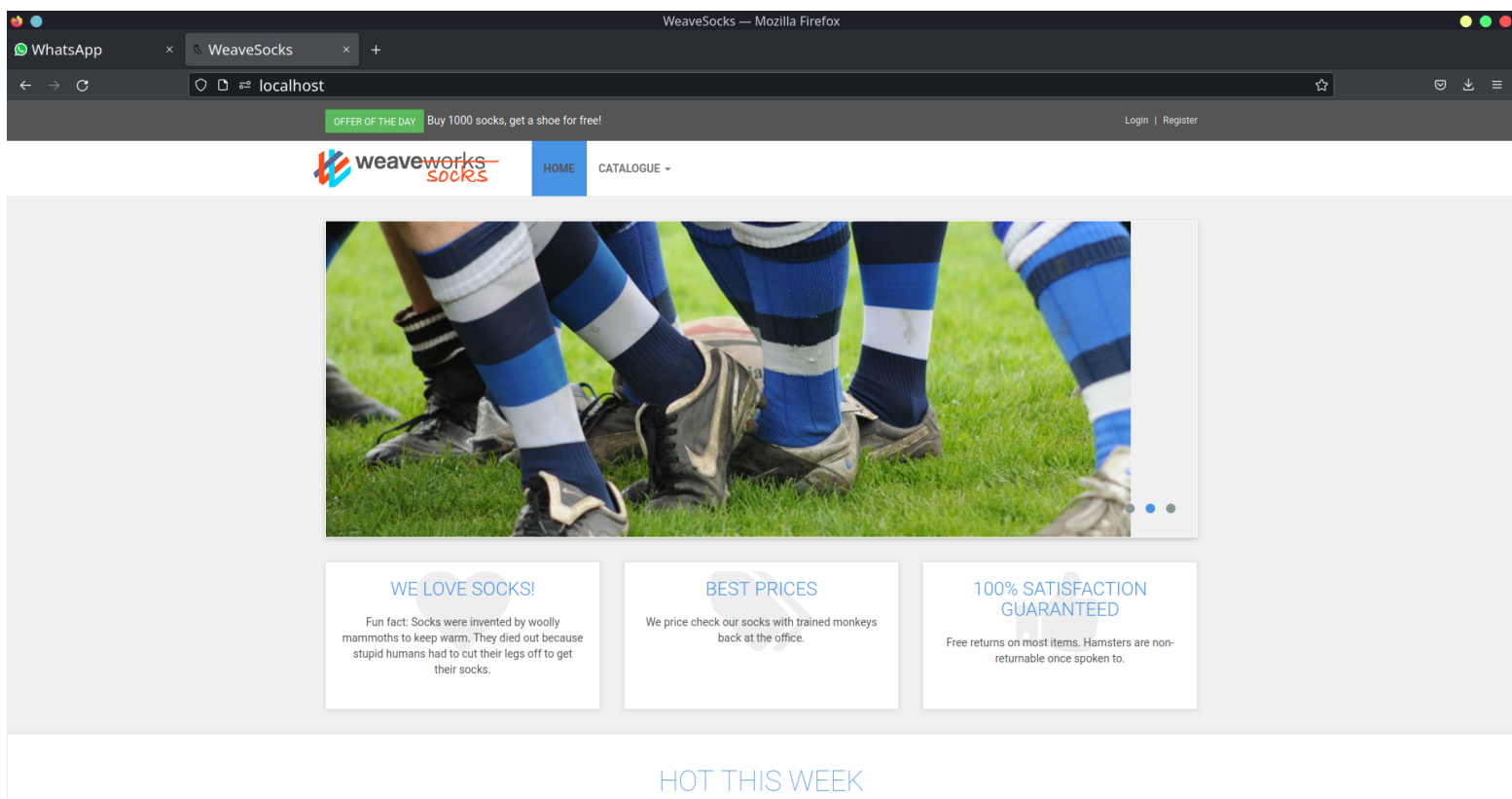
Año: 2022

Ejercicio 1:

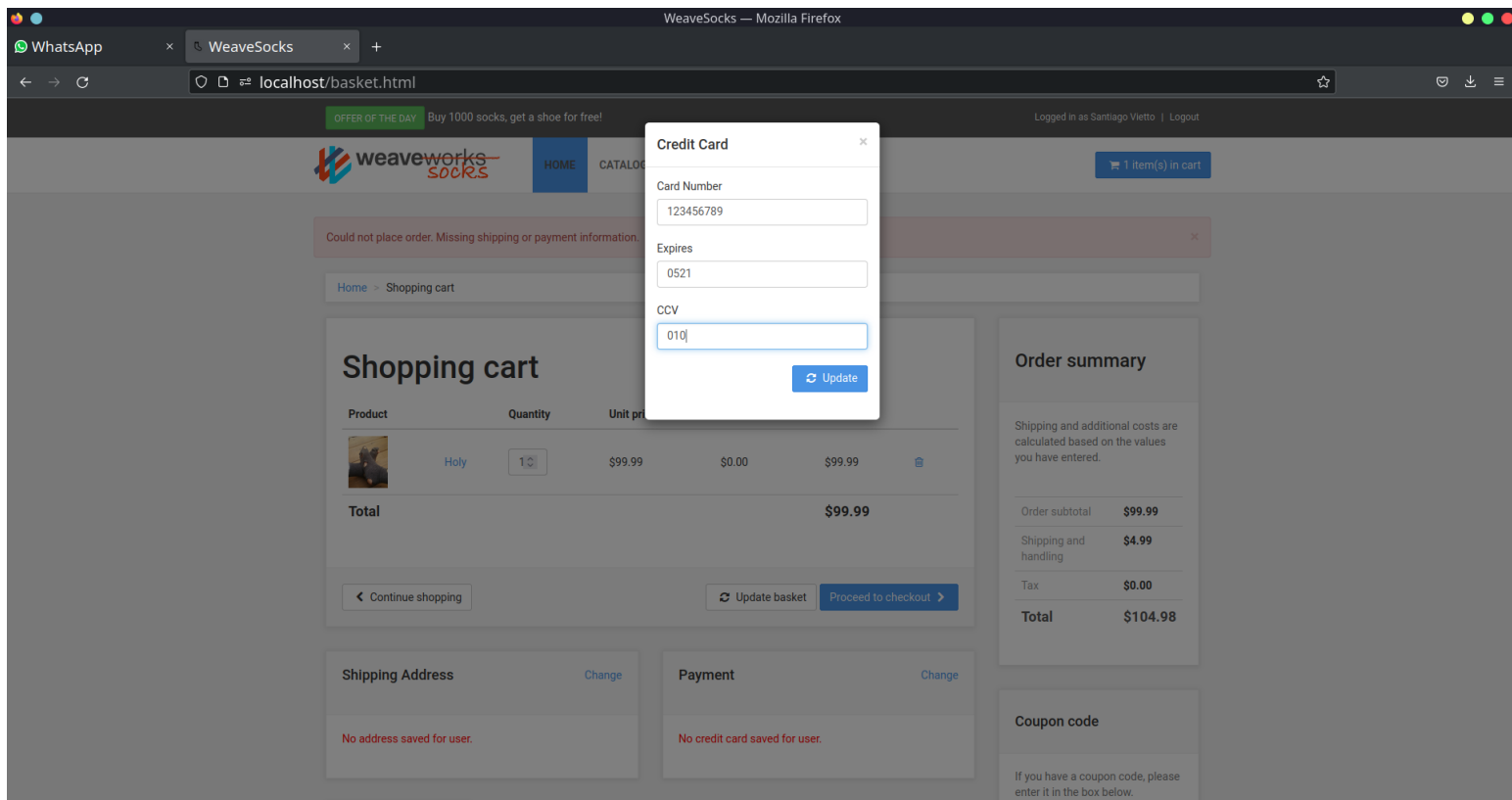
```
~$ mkdir -p socks-demo
~$ cd socks-demo
~/socks-demo$ git config --global user.name "santi2019"
~/socks-demo$ git config --global user.email "santiagovietto5@hotmail.com"
~/socks-demo$ git clone https://github.com/microservices-demo/microservices-demo
Cloning into 'microservices-demo'...
remote: Enumerating objects: 10197, done.
remote: Total 10197 (delta 0), reused 0 (delta 0), pack-reused 10197
Receiving objects: 100% (10197/10197), 52.95 MiB | 1.03 MiB/s, done.
Resolving deltas: 100% (6208/6208), done.
~/socks-demo$ cd microservices-demo
```

```
~/socks-demo/microservices-demo$ sudo systemctl start docker.service
~/socks-demo/microservices-demo$ sudo docker-compose -f deploy/docker-compose/docker-compose.yml up -d
WARNING: The MYSQL_ROOT_PASSWORD variable is not set. Defaulting to a blank string.
Creating network "docker-compose_default" with the default driver
Pulling front-end (weaveworksdemos/front-end:0.3.12)...
0.3.12: Pulling from weaveworksdemos/front-end
709515475419: Pull complete
8a7a0a7b6b80: Pull complete
79dadb976eeb: Pull complete
f48430971a7c: Pull complete
69ee1f19577a: Pull complete
89bb0a032ada: Pull complete
01d99281a40a: Pull complete
07a3ae088a4f: Pull complete
a9efb8659a27: Pull complete
Digest: sha256:26a2d9b6b291dee2dca32fca3f5bff6c2fa07bb5954359afcbcb8001cc70eac71
Status: Downloaded newer image for weaveworksdemos/front-end:0.3.12
Pulling edge-router (weaveworksdemos/edge-router:0.1.1)...
0.1.1: Pulling from weaveworksdemos/edge-router
b7f33cc0b48e: Already exists
6c1bb5764098: Pull complete
3b141d7c0818: Pull complete
820d487bb3eb: Pull complete
2261519a3626: Pull complete
Digest: sha256:72855b1ab6ec5b8b7e03429af9fa04a2a87f97a8c139724afc75a47970fa6438
Status: Downloaded newer image for weaveworksdemos/edge-router:0.1.1
Pulling catalogue (weaveworksdemos/catalogue:0.3.5)...
0.3.5: Pulling from weaveworksdemos/catalogue
709515475419: Already exists
a1a52e31c764: Pull complete
0c10f86b9838: Pull complete
449f4beba5c2: Pull complete
eba920f195c8: Pull complete
Digest: sha256:0147a65b7116569439eefb1a6dbed455fe022464ef70e0c3cab75bc4a226b39b
Status: Downloaded newer image for weaveworksdemos/catalogue:0.3.5
Pulling catalogue-db (weaveworksdemos/catalogue-db:0.3.0)...
0.3.0: Pulling from weaveworksdemos/catalogue-db
```

Accedemos al sitio web y procedemos a registrarnos:



Buscamos un artículo, lo añadimos al carrito de compras y procedemos a comprarlo añadiendo datos de la tarjeta de débito/crédito (saldrá un cartel de que los datos de la misma son falsos):



Ejercicio 2:

1)_ A continuación docker-compose.yml:

- front-end: contenedor que carga el front end de la página usando la imagen weaveworksdemos/front-end:0.3.12 y nos permite visualizar la misma.
- 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: contenedor que carga el front end y los servicios de la opción catalogue en la página.
- catalogue-db: contenedor que 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: contenedor que carga el front end y los servicios de la opción del carrito de compras para añadir más productos, en la página.
- carts-db: contenedor que 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: contenedor que carga el front end y los servicios de la opción órdenes en la página, y que permite visualizar una vez registrado, todas las órdenes de compras.
- orders-db: contenedor que posee todos los datos de las órdenes en la base de datos. Utiliza una imagen de una base de datos no relacional mongo.

- shipping: contenedor que carga el front end y los servicios de la opción de envíos. No lo podemos visualizar ya que no podemos realizar ninguna compra.
- queue-master: es la lógica de las colas. Implementa una cola de envíos y simula el proceso de envío.
- rabbitmq: es como un redis que maneja colas, es decir, es un sistema de manejo de colas. El sistema a través de este carga un mensaje a través de la cola de envíos.
- payment: contenedor que carga el front end y los servicios de la opción payment en la página, y que permite añadir los datos de la tarjeta de débito/crédito para realizar el pago.
- user: contenedor que carga el front end y los servicios de la opción login en la página, y que permite añadir los datos del usuario para iniciar sesión y acceder a la página.
- user-db: contenedor que posee todos los datos de los usuarios en la base de datos.
- user-sim: contenedor que posee los datos de un usuario test que es usado por el desarrollador para probar o testear funcionalidades.

2)_ Clonamos los repositorios:

```

❏ ~/socks-demo ➤ git clone https://github.com/microservices-demo/front-end.git
Cloning into 'front-end'...
remote: Enumerating objects: 1232, done.
remote: Total 1232 (delta 0), reused 0 (delta 0), pack-reused 1232
Receiving objects: 100% (1232/1232), 47.89 MiB | 1.10 MiB/s, done.
Resolving deltas: 100% (684/684), done.
❏ ~/socks-demo ➤ git clone https://github.com/microservices-demo/user.git
Cloning into 'user'...
remote: Enumerating objects: 1063, done.
remote: Total 1063 (delta 0), reused 0 (delta 0), pack-reused 1063
Receiving objects: 100% (1063/1063), 172.83 KiB | 867.00 KiB/s, done.
Resolving deltas: 100% (601/601), done.
❏ ~/socks-demo ➤ git clone https://github.com/microservices-demo/edge-router.git
Cloning into 'edge-router'...
remote: Enumerating objects: 50, done.
remote: Total 50 (delta 0), reused 0 (delta 0), pack-reused 50
Receiving objects: 100% (50/50), 14.51 KiB | 297.00 KiB/s, done.
Resolving deltas: 100% (12/12), done.

```

```

❏ ~/socks-demo ➤ git clone https://github.com/microservices-demo/queue-master.git
Cloning into 'queue-master'...
remote: Enumerating objects: 299, done.
remote: Total 299 (delta 0), reused 0 (delta 0), pack-reused 299
Receiving objects: 100% (299/299), 51.22 KiB | 481.00 KiB/s, done.
Resolving deltas: 100% (89/89), done.

```

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.

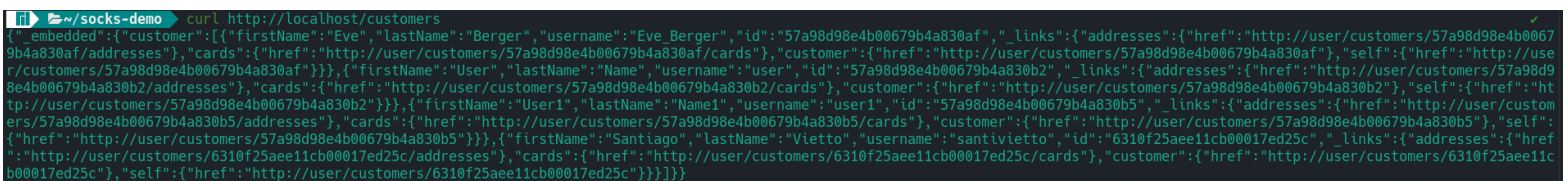
- Libertad del desarrollador de desarrollar y desplegar servicios de forma independiente.
- Uso de contenedores permitiendo el despliegue y el desarrollo de la aplicación rápidamente.
- Independizamos la lógica, también en algunos casos lenguajes.

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.

4)_ En este caso, el contenedor que hace las veces de API Gateway es edge-router. Traefik es la herramienta de API Gateway.

5)_



```
curl http://localhost/customers
{"_embedded":{"customer":[{"firstName":"Eve","lastName":"Berger","id":"57a98d98e4b00679b4a830af","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830af/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830af/cards"},"customer":{"href":"http://user/customers/57a98d98e4b00679b4a830af"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830af"}}}],{"firstName":"User","lastName":"Name","id":"57a98d98e4b00679b4a830b2","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830b2/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830b2/cards"},"customer":{"href":"http://user/customers/57a98d98e4b00679b4a830b2"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830b2"}}}],{"firstName":"User1","lastName":"Name1","id":"57a98d98e4b00679b4a830b5","_links":{"addresses":{"href":"http://user/customers/57a98d98e4b00679b4a830b5/addresses"},"cards":{"href":"http://user/customers/57a98d98e4b00679b4a830b5/cards"},"customer":{"href":"http://user/customers/57a98d98e4b00679b4a830b5"},"self":{"href":"http://user/customers/57a98d98e4b00679b4a830b5"}}}],{"firstName":"Santiago","lastName":"Vietto","id":"6310f25aee11cb00017ed25c","_links":{"addresses":{"href":"http://user/customers/6310f25aee11cb00017ed25c/addresses"},"cards":{"href":"http://user/customers/6310f25aee11cb00017ed25c/cards"},"customer":{"href":"http://user/customers/6310f25aee11cb00017ed25c"},"self":{"href":"http://user/customers/6310f25aee11cb00017ed25c"}}}]}}
```

6)_ Analizando la estructura de los contenedores a través del IDE VSCodium podemos determinar que la operación está siendo procesada por el servicio “user”. (El endpoint de customers se encuentra dentro del servicio de user).

7)_ Para los casos de los endpoints de catalogue y tags, ambos se encuentran y son utilizados dentro del servicio “catalogue”.

8)_ En este caso vemos que los datos van a persistir debido a que tenemos múltiples y diferentes bases de datos, correspondientes a cada servicio. Las bases de datos mongo almacenan por ejemplo toda la información del usuario (nombre, correo, nombre de usuario e información de la tarjeta), también todo lo que se guarda en el carrito de compras, luego tenemos una base de datos mysql que se utiliza para guardar por ejemplo la información del catálogo de medias de la página, y así sucesivamente con cada servicio. De esta manera si borramos alguno de los servicios y lo clonamos de vuelta, este va a seguir teniendo los datos originales.

9)_ El componente encargado del procesamiento de la cola de mensajes es rabbitmq.

10)_ El tipo de interfaz que utilizan estos microservicios para comunicarse es del tipo API REST.