

# Arquitectura de Software II

## **Entrega 2: desarrollo** **de microservicios**

---

Alumnos: Mauro Bailon, Federico Sandoval.

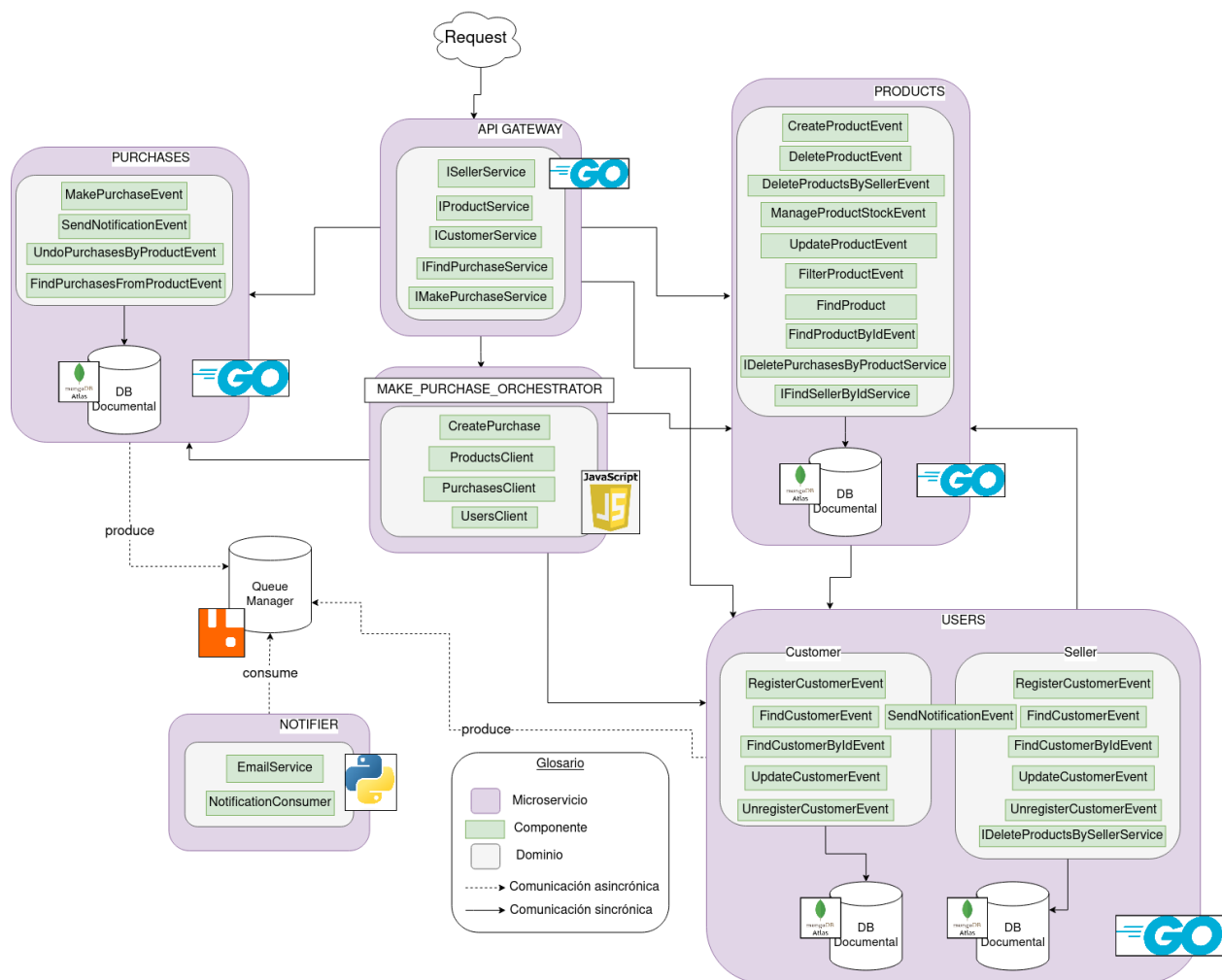
Profesores: Marcia Tejeda, Mariano Claveria.

Universidad Nacional de Quilmes, 2023s1.

# Componentes

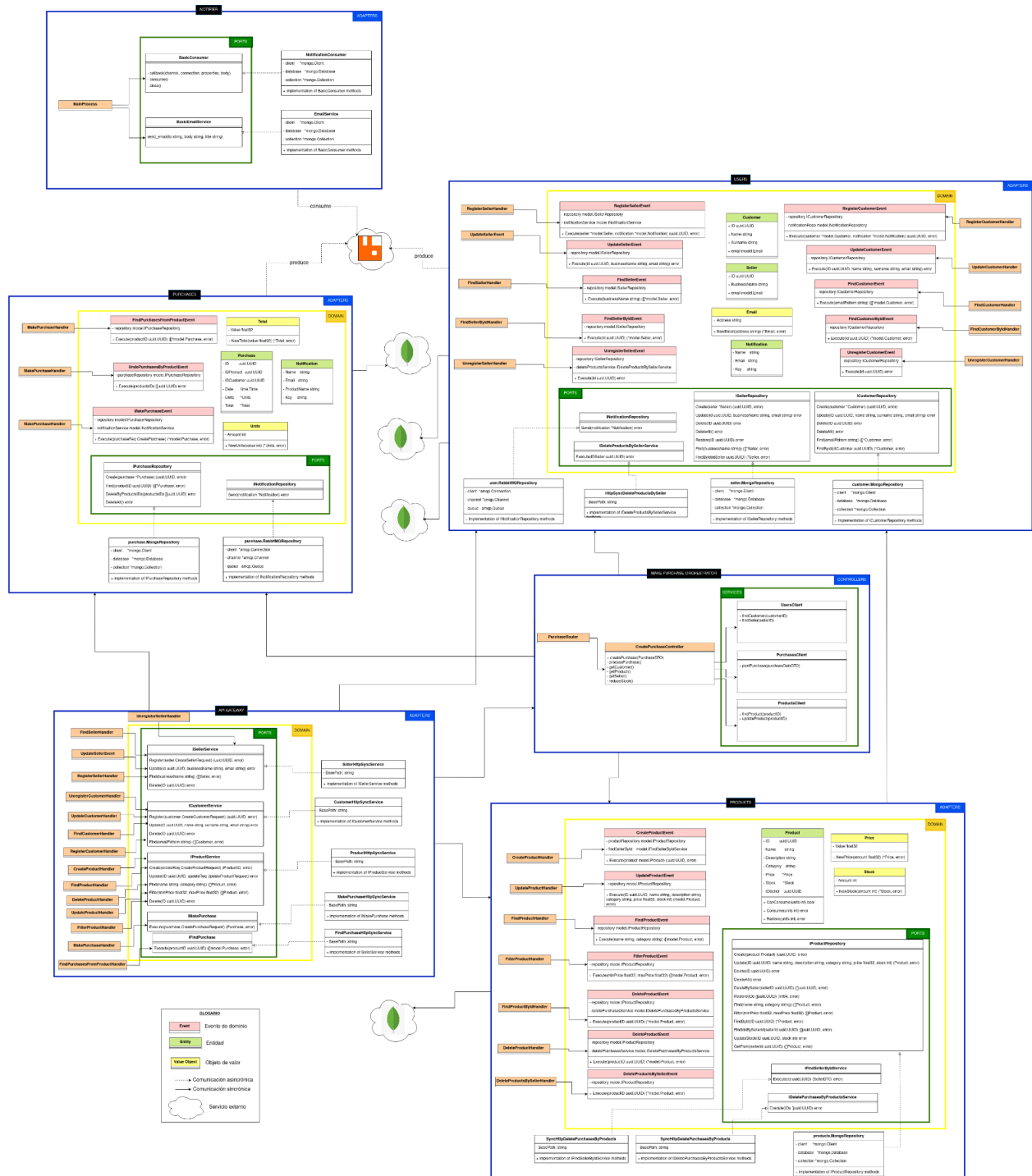
## Diagrama de componentes

Este diagrama fue creado con la idea de representar los componentes, tipos de comunicaciones entre componentes, lenguajes y tecnologías utilizadas, y los microservicios con sus principales componentes sin contar con los de acceso a bases de datos. Para una mejor visualización haga [click acá](#).



## Diagrama de arquitectura

Este diagrama se encuentra adjunto en el repositorio y a su vez puede accederlo con [click aqui](#).



## Decisiones a la hora de desintegrar el monolito

En primera instancia se decidió desintegrar en el microservicio de *purchases* la lógica relacionada a las compras basándonos en el desintegrador de escalabilidad y rendimiento porque creemos que este servicio será el más consumido de la aplicación por el caso de uso principal del domini, realizar una compra.

Por otro lado, se desintegró en el microservicio de *products* la lógica relacionada a los productos por su alcance y funcionalidad, esto es debido a todo lo que integra a este servicio en su gran mayoría depende de sí mismo y todo lo que integra este servicio forma parte de la lógica de gestión de un producto. A su vez, como nuestro dominio se basa en compra de productos, podemos suponer que también será un servicio bastante solicitado, por lo que el desintegrador de escalabilidad y rendimiento se suma a la necesidad de desacoplar esta funcionalidad en este microservicio.

Por último, hemos decidido crear el microservicio *users* para colocar toda la lógica que tiene que ver con los usuarios de la plataforma, tanto clientes como vendedores. Creemos que comparten lógica similar (casos de uso sobre las entidades) y que también representan a lo mismo en el mundo real, personas. Sumado a esto, creemos que serán los servicios menos utilizados en comparación a los otros dos. Por último, podemos deducir que el desintegrador de seguridad es el más adecuado para mantener separado este servicio de los otros, debido a que se manejaría información sensible sobre personas y, a la vez, poseen código compartido. Por todo lo mencionado, creemos que es una buena idea mantener estos servicios en conjunto.

## Data ownerships

Se ha trabajado para que cada servicio sea dueño de sus datos y que si un servicio externo desea acceder a los mismos lo haga a través de la API del servicio, por lo que la relación de datos y servicios quedó diseñada de la siguiente manera:

- *products*: es el dueño de la información relacionada a los productos del sistema.
- *purchases*: es el dueño de la información relacionada a las compras del sistema.
- *users*: en este caso, cabe recalcar que el acoplamiento del servicio de vendedor y el de cliente, solo se da debido a que pertenecen a la misma unidad de deploy, pero a nivel datos, cada servicio cuenta con su propia DB para manipulación y acceso de los mismos. Por lo que:
  - *sellers* es el servicio, dentro de *users*, dueño de la información de los vendedores.
  - *customers* es el servicio, dentro de *users*, dueño de la información de los clientes.

## Otros microservicios que integran el sistema


Si bien los tres servicios mencionados son los que poseen la manipulación de las entidades de dominio del sistema, nuestra aplicación posee microservicios extra que la complementan. Los cuales son:

- *notifier*: es un servicio que se encarga de consumir mensajes que los demás servicios generen y enviar notificaciones a los usuarios que les corresponda dicho mensaje. Trabaja de forma asíncrona a través de una cola de mensajes y actualmente, envía notificaciones a los nuevos usuarios (tanto vendedores como clientes) de registro exitoso; y a la vez, cuando se produce una compra, notifica al dueño del producto (vendedor) y al comprador del mismo (cliente). Actualmente su único método de notificación es vía correo, pero esto podría cambiar o inclusive tener más de una forma de notificación ante más eventos, por lo que el desintegrador de extensibilidad es un gran justificativo para mantener a este microservicio desacoplado de los demás.
- *api gateway*: este microservicio fue creado con el fin de seguir el patrón con su mismo nombre que es bastante común a la hora de modelar esta arquitectura. Su responsabilidad básica es hacer para el cliente una comunicación transparente con los microservicios de la aplicación, ahorrando tiempo de configuración y a la vez creando una caja negra para que el mundo exterior no sepa, y a la vez le sea indistinto, con quien está hablando, sino que simplemente funciona como un proxy entre el mundo real y la aplicación.
- *make purchase orchestrator*: la finalidad de este servicio es bastante clara, servir como orquestador del caso de uso más importante que envuelve a todos los servicios dueños de información, realizar una compra. El orquestador se encarga de controlar que esta transacción que se encuentra distribuida entre los servicios *customer*, *product* y *seller*, se lleve a cabo y se realicen los rollbacks correspondientes para tener consistencia de la información que ronda en dichos servicios.

## Servicios externos

Nuestro sistema utiliza dos servicios externos a la hora de ponerse en marcha.

Por un lado, utiliza cuatro bases de datos distintas hospedadas por Mongo Atlas. Cada base de dato manipula y es conectada por los servicios que son dueños de información del dominio: *products*, *sellers*, *customers* y *purchases*.



Y por el otro lado, cuenta con el servicio de AMPQ para encolar y consumir los mensajes que se producen dentro de la app.

## Patrón Saga

Se ha utilizado en cuatro oportunidades este patrón para el desarrollo de distintos casos de uso. Contando con un orquestador y tres coreografías.

- orquestador de comprar un producto: consiste en confirmar la existencia del producto y el cliente que desea adquirirlo, consumir el stock del producto y grabar la compra en el sistema. En caso de no existir el producto o cliente, no hay rollback; en caso de no falla al consumir stock, no hay rollback; y en el caso de falla al grabar una compra, se realiza el rollback para retornar el stock consumido por la compra que falló al grabarse.
- coreografía de crear un producto: consiste en, a la hora de crear un producto, chequear la existencia del vendedor del mismo y luego guardar el producto en el sistema. Esta coreografía no posee rollbacks debido a que se realiza una única escritura.
- coreografía de borra un vendedor: consiste en eliminar un vendedor, luego los productos asociados al mismo y por último las compras asociadas a los productos borrados. Si no existe el vendedor no se realiza rollback; si se falla a la hora de eliminar los productos del vendedor, se realiza un rollback para que el vendedor vuelva a estar en el sistema; si hay una falla a la hora de eliminar las compras asociadas a los productos eliminados, se realiza el rollback para que se restaure la información eliminada de los productos y su vendedor.
- coreografía de borrar un producto: consiste en eliminar un producto y las compras asociadas al mismo. Si se falla al borrar el producto no hay rollback; si hay una falla a la hora de eliminar las compras asociadas al producto, se realiza el rollback para que se restaure la información del producto.

Debido a las transacciones distribuidas que se dan en las coreografías de borrado, se optó por cambiar la implementación de borrado físico a borrado lógico (soft delete), que consiste en marcar una entidad como eliminada para el sistema pero que siga existiendo en la base de datos, para que, si se debe realizar un rollback (restaurar la información), simplemente se elimina la marca puesta a la entidad y la misma vuelve a existir para el sistema y el mundo que lo consume..

## Casos de uso

<b>Caso de uso</b>	CU-1
<b>Título</b>	Registrar cliente
<b>Actor</b>	Cliente
<b>Descripción</b>	Como cliente quiero poder registrarme en el sistema con mi nombre, apellido y correo electrónico.

<b>Caso de uso</b>	CU-2
<b>Título</b>	Edición de cliente
<b>Actor</b>	Cliente
<b>Descripción</b>	Como cliente quiero poder modificar mi nombre, apellido y correo electrónico.

<b>Caso de uso</b>	CU-3
<b>Título</b>	Búsqueda de clientes
<b>Actor</b>	Cliente
<b>Descripción</b>	Como cliente quiero poder buscar otros clientes por correo electrónico.

<b>Caso de uso</b>	CU-4
<b>Título</b>	Dar de baja cliente
<b>Actor</b>	Cliente

<b>Descripción</b>	Como cliente quiero poder darme de baja del sistema.
--------------------	--

<b>Caso de uso</b>	CU-5
<b>Título</b>	Registrar vendedor
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder registrarme en el sistema con mi nombre de negocio y mi correo.

<b>Caso de uso</b>	CU-6
<b>Título</b>	Edición de vendedor
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder modificar mi nombre de negocio y correo electrónico.

<b>Caso de uso</b>	CU-7
<b>Título</b>	Búsqueda de vendedores
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como cliente quiero buscar a otros vendedores por nombre de negocio.

<b>Caso de uso</b>	CU-8
<b>Título</b>	Dar de baja vendedor
<b>Actor</b>	Vendedor



<b>Descripción</b>	Como vendedor quiero poder darme de baja del sistema junto a mis productos.
--------------------	---

<b>Caso de uso</b>	CU-9
<b>Título</b>	Crear producto
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder crear un producto con un nombre, precio y stock disponible.

<b>Caso de uso</b>	CU-10
<b>Título</b>	Edición de producto
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder modificar el nombre, precio o stock disponible de un producto.

<b>Caso de uso</b>	CU-11
<b>Título</b>	Eliminar un producto
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder eliminar un producto del sistema.

<b>Caso de uso</b>	CU-12
<b>Título</b>	Buscar producto

<b>Actor</b>	Cliente y Vendedor
<b>Descripción</b>	Como cliente y vendedor quiero buscar productos del sistema por nombre o categoría.

<b>Caso de uso</b>	CU-13
<b>Título</b>	Filtrar productos
<b>Actor</b>	Cliente y Vendedor
<b>Descripción</b>	Como cliente y vendedor quiero filtrar los productos del sistema por un precio mínimo, un precio máximo o un rango de precio.

<b>Caso de uso</b>	CU-14
<b>Título</b>	Realizar una compra
<b>Actor</b>	Cliente
<b>Descripción</b>	Como cliente quiero poder realizar la compra de una cantidad determinada de un producto.

<b>Caso de uso</b>	CU-15
<b>Título</b>	Obtener compras de producto
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero poder buscar las compras realizadas sobre un producto determinado.

<b>Caso de uso</b>	CU-16
--------------------	-------

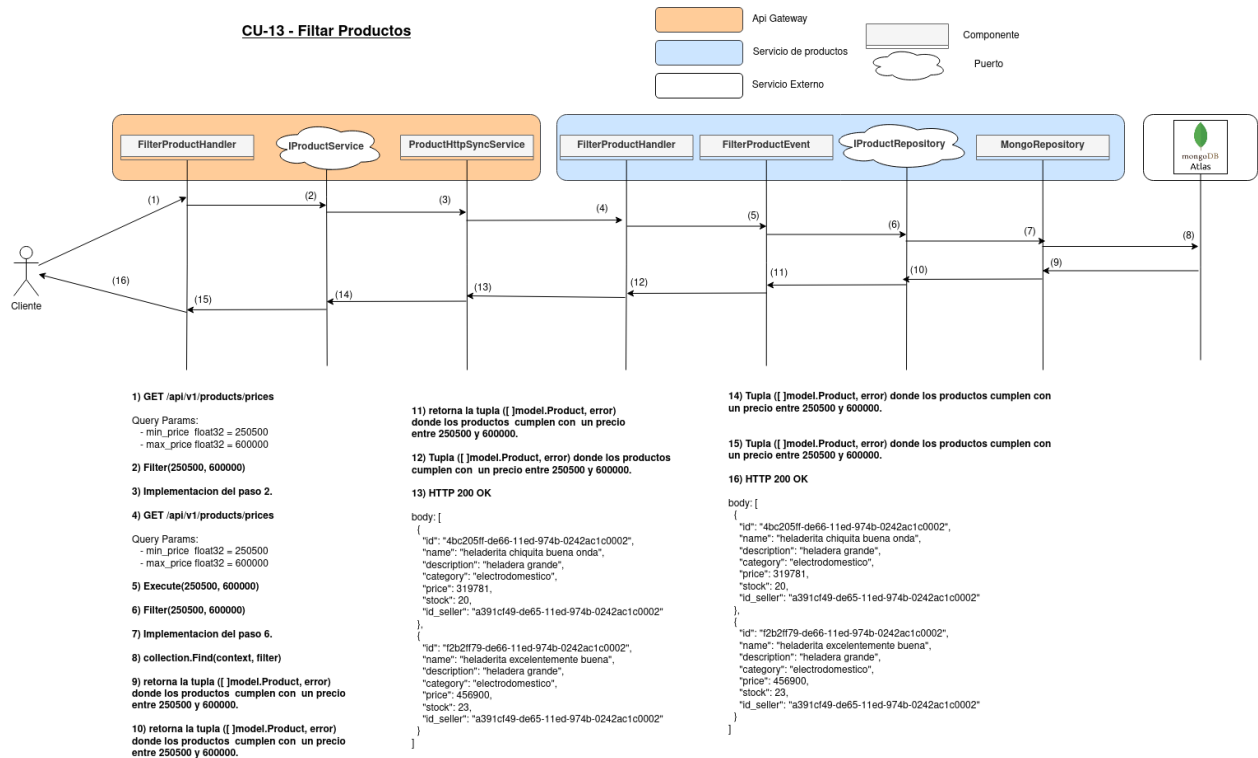
<b>Título</b>	Notificación de compra a vendedor
<b>Actor</b>	Vendedor
<b>Descripción</b>	Como vendedor quiero recibir notificaciones cada vez que se realice la compra de alguno de mis productos.

<b>Caso de uso</b>	CU-17
<b>Título</b>	Notificación de compra a cliente
<b>Actor</b>	Cliente
<b>Descripción</b>	Como cliente quiero poder recibir notificaciones cada vez que realice la compra un producto.

## Diagramas de secuencia

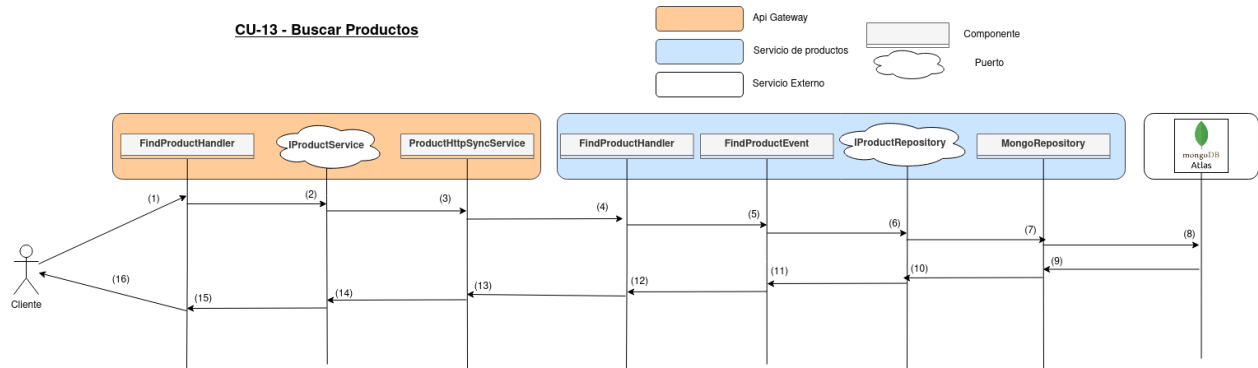
Todos los diagramas se encuentran adjuntos en el repositorio para mejor visualización.

## Filtrar productos por precio



## Buscar productos por nombre y categoria

### CU-13 - Buscar Productos



#### 1) GET /api/v1/products

Query Params:  
 - name string = "BueNa"  
 - category string = "DomestiCo"

#### 2) Find('BueNa', 'DomestiCo')

#### 3) Implementacion del paso 2.

#### 4) GET /api/v1/products

Query Params:  
 - name string = "BueNa"  
 - category string = "eleC"

#### 5) Execute('BueNa', 'DomestiCo')

#### 6) Find('BueNa', 'DomestiCo')

#### 7) Implementacion del paso 6.

#### 8) collection.Find(context, filterIgnoringCase)

9) retorna la tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

10) retorna la tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

11) retorna la tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

12) Tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

#### 13) HTTP 200 OK

```
body: [
  {
    "id": "4bc205ff-de66-11ed-974b-0242ac1c0002",
    "name": "heladerita chiquita buena onda",
    "description": "heladera grande",
    "category": "electrodomestico",
    "price": 319781,
    "stock": 20,
    "id_seller": "a391cd49-de65-11ed-974b-0242ac1c0002"
  },
  {
    "id": "f2b2f79-de66-11ed-974b-0242ac1c0002",
    "name": "heladerita excelentemente buena",
    "description": "heladera grande",
    "category": "electrodomestico",
    "price": 456900,
    "stock": 23,
    "id_seller": "a391cd49-de65-11ed-974b-0242ac1c0002"
  }
]
```

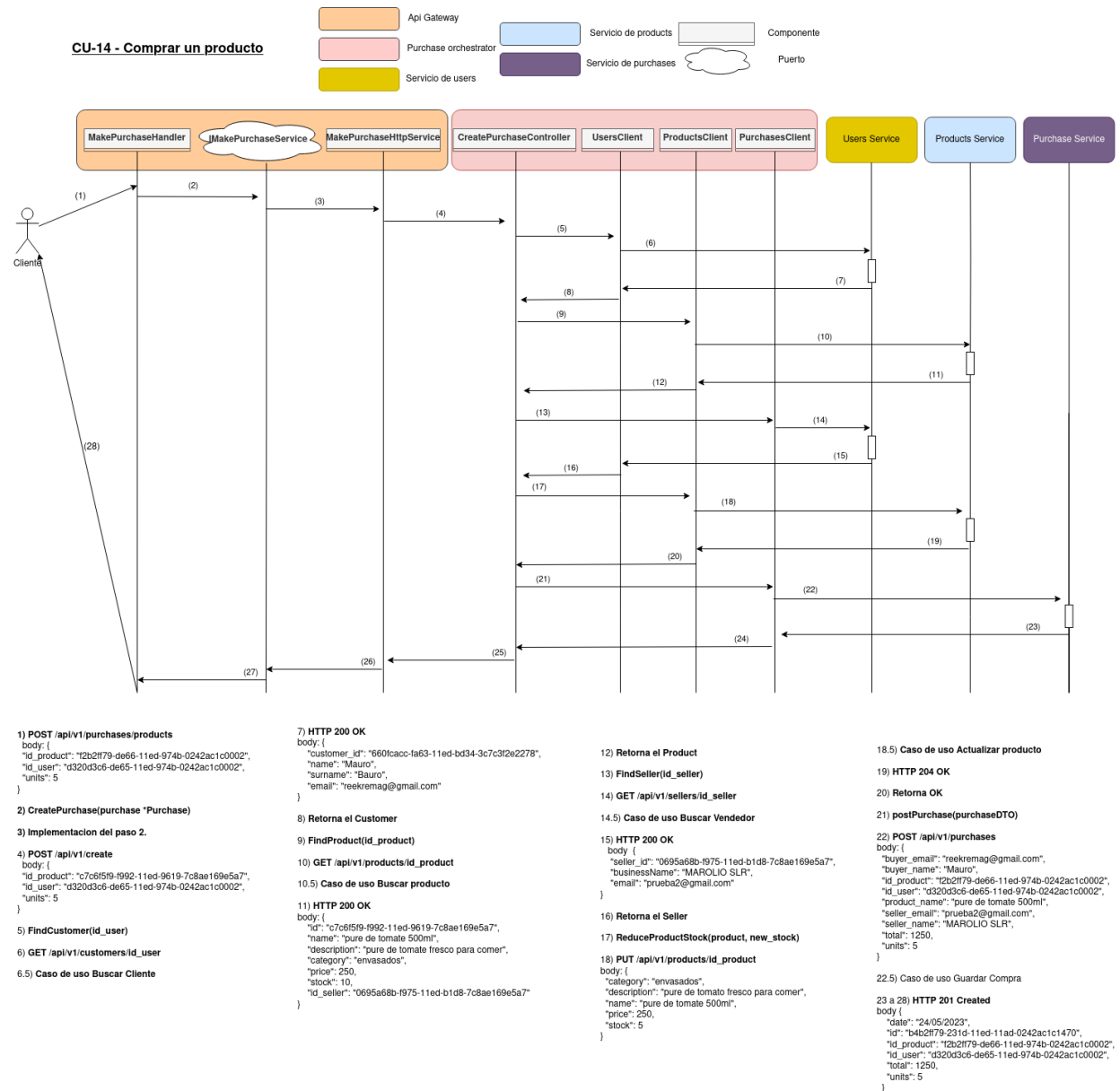
14) Tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

15) Tupla ([IModel.Product, error]) donde los productos cumplen con contener en su nombre 'buena' y en su categoria 'domestico'.

#### 16) HTTP 200 OK

```
body: [
  {
    "id": "4bc205ff-de66-11ed-974b-0242ac1c0002",
    "name": "heladerita chiquita buena onda",
    "description": "heladera grande",
    "category": "electrodomestico",
    "price": 319781,
    "stock": 20,
    "id_seller": "a391cd49-de65-11ed-974b-0242ac1c0002"
  },
  {
    "id": "f2b2f79-de66-11ed-974b-0242ac1c0002",
    "name": "heladerita excelentemente buena",
    "description": "heladera grande",
    "category": "electrodomestico",
    "price": 456900,
    "stock": 23,
    "id_seller": "a391cd49-de65-11ed-974b-0242ac1c0002"
  }
]
```

## Comprar un producto.



## Dar de baja un vendedor.

