



powered by **{codemotion}**

Frontend vitaminized from the backend

MIGUEL GARCIA SANGUINO



Miguel García Sanguino

15 años como developer
Frontend 70% Backend 30%
Software engineer en ING



twitter.com/sanguinoide



github.com/sanguino



linkedin.com/in/sanguinoide

Frontend vitaminized from the backend

Un front en un master de cloud apps ¿De que hago el TFM?

Procesos complicados ...

Multi respuestas ...

Asincronia ...

Actualizaciones ...

Modelos de datos ...

Relaciones entre squads ...

Trabajo de investigación

Pains - modelos de datos

Cuentas

Modificar notificaciones

Vas a realizar cambios en las notificaciones de esta cuenta:

☒

 Cuenta NÓMINA

Elige cuándo y cómo quieres recibir la notificación.

Transferencia recibida superior a

0

€

Email

☐

Móvil

☒

Transferencia ejecutada superior a

0

€

Email

☐

Móvil

☒

Las recibirás en el momento que la transferencia llegue a su destino

Transferencia ordenada superior a

Email

☒

Móvil

☒

Por seguridad, siempre las recibirás cuando realices una transferencia

✕ Cancelar

Guardar cambios

id	account	type	min_amount	channel	active
1	C. Nomina	trans_rcv	0	email	FALSE
2	C. Nomina	trans_rcv	0	push	TRUE
3	C. Nomina	trans_exec	0	email	FALSE
4	C. Nomina	trans_exec	0	push	TRUE
5	C. Nomina	trans_order	0	email	TRUE
6	C. Nomina	trans_order	0	email	TRUE

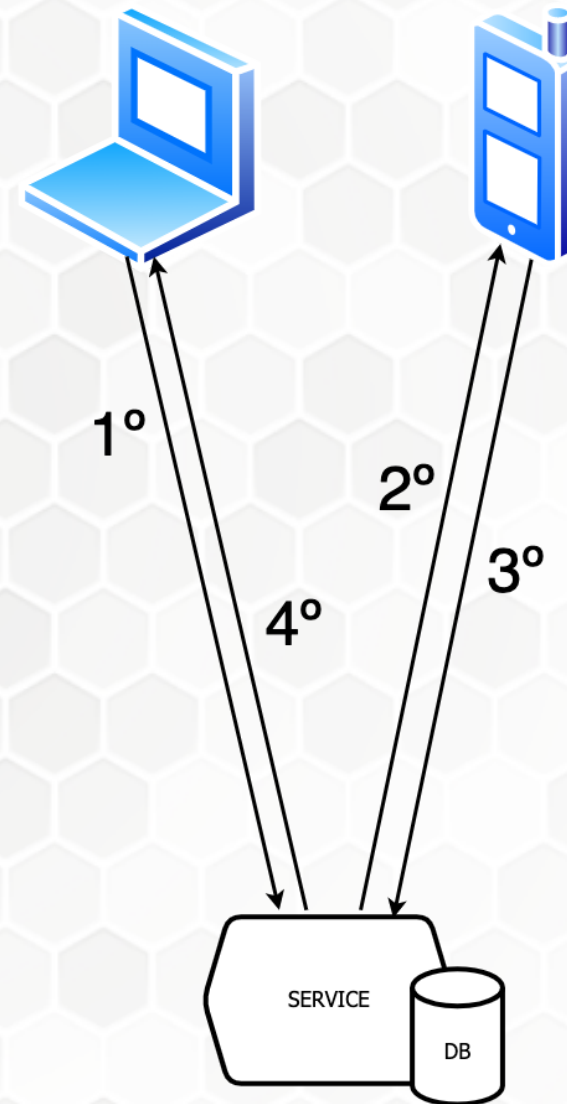
Frontend vitaminized from the backend - Miguel García Sanguino

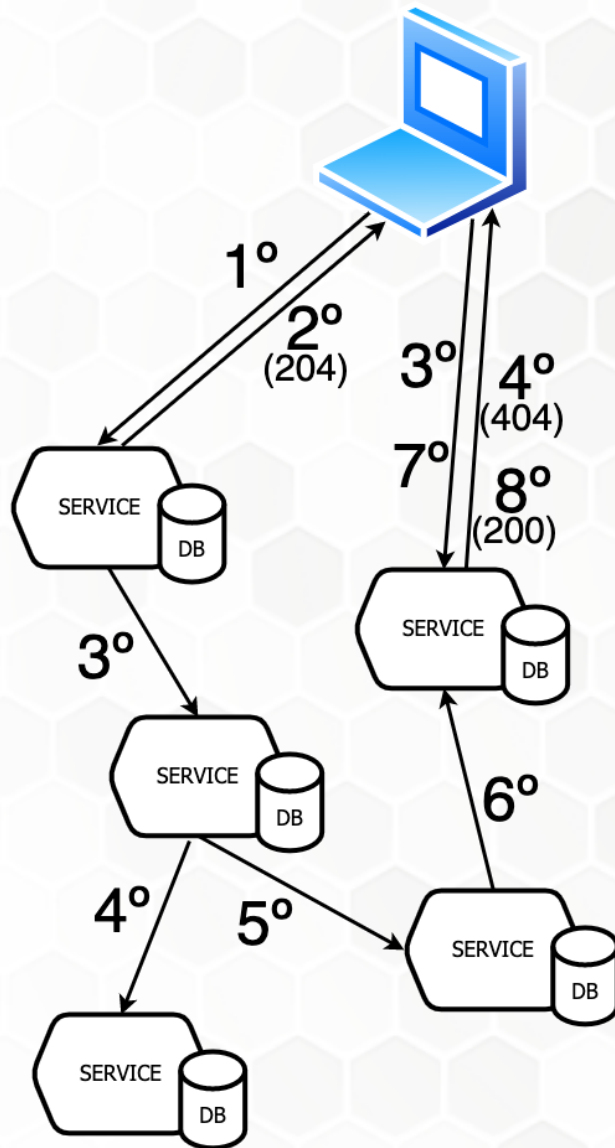
Pains - respuestas asincronas

Proceso depende de otra acción humana

Solución: comunicación servidor - cliente

Long pooling, WebComponents, o ServerSentEvents, servicio con complejidad extra !~ funcional





Pains - respuestas asincronas

Proceso depende de otros
asincronos

Solución: espera? 204 y preguntar?

De nuevo pooling, WC o SSE, pero
que servicio se queda con la
conexión?

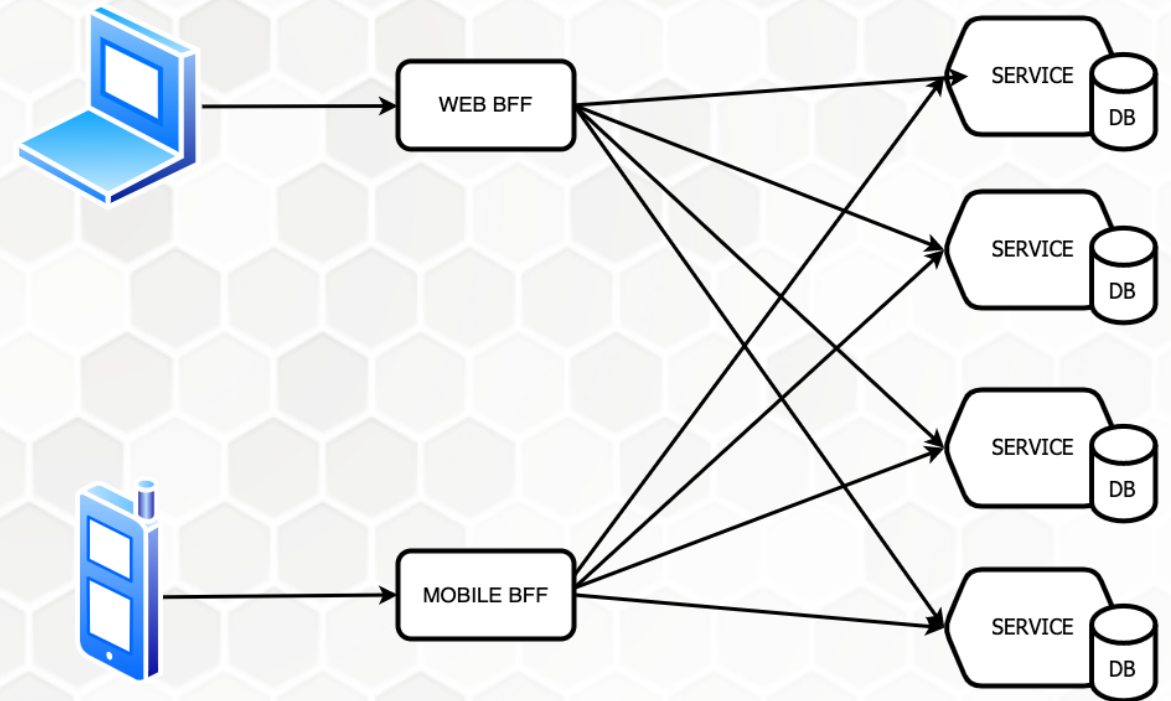
Patron Backend for Frontend

Uno por cada tipo de cliente

Adapta el api a cada consumidor

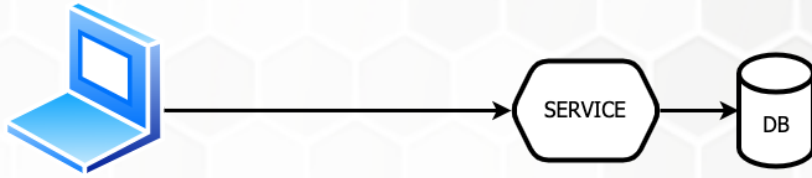
Simplifica clientes

Elimina la sobrecarga de servicios



BFF : añadir un servicio solo para el front

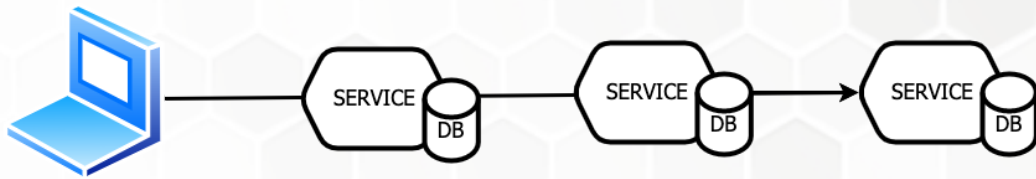
Front recibe los datos que necesita,
cuando los necesita, desde el BFF,
en el formato que los necesita,
sin entorpecer a middle,
ni en el modelo,
ni con desarrollos extra

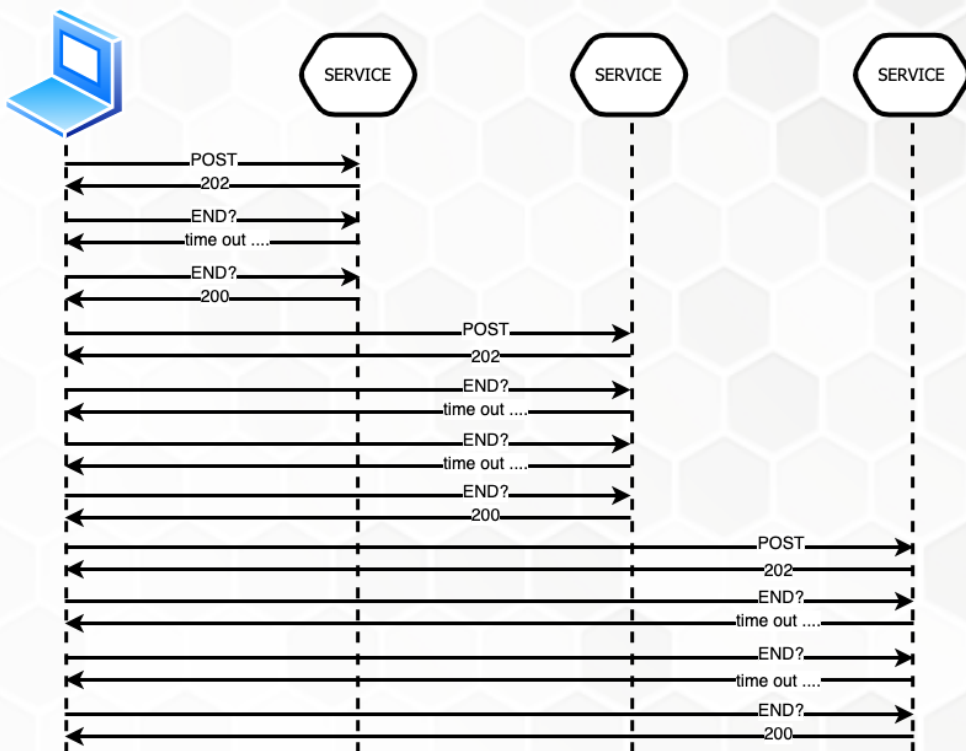
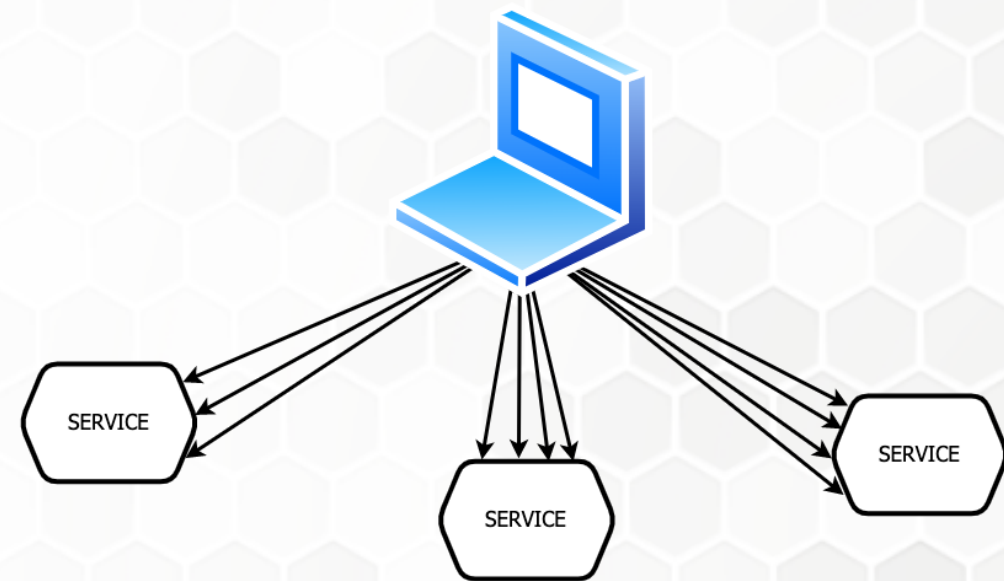


Pains - transacciones

Antes teníamos transacciones ACID "simples"

Si tenemos múltiples bases de datos? que hacemos?





Pains - respuesta al front

Has creado el pedido ya?

Has creado el pedido ya?

Has creado el pedido ya?

Si

Has pedido la comida ya?

Has pedido la comida ya?

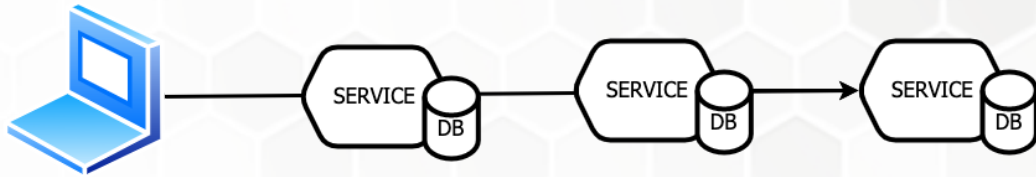
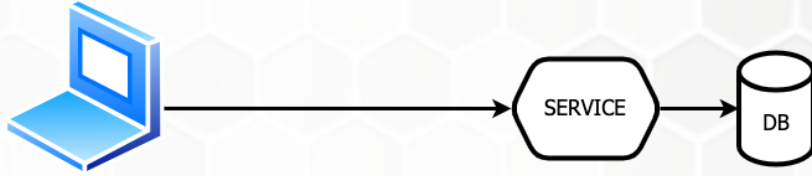
Has pedido la comida ya?

Si

Has reservado un rider ya?

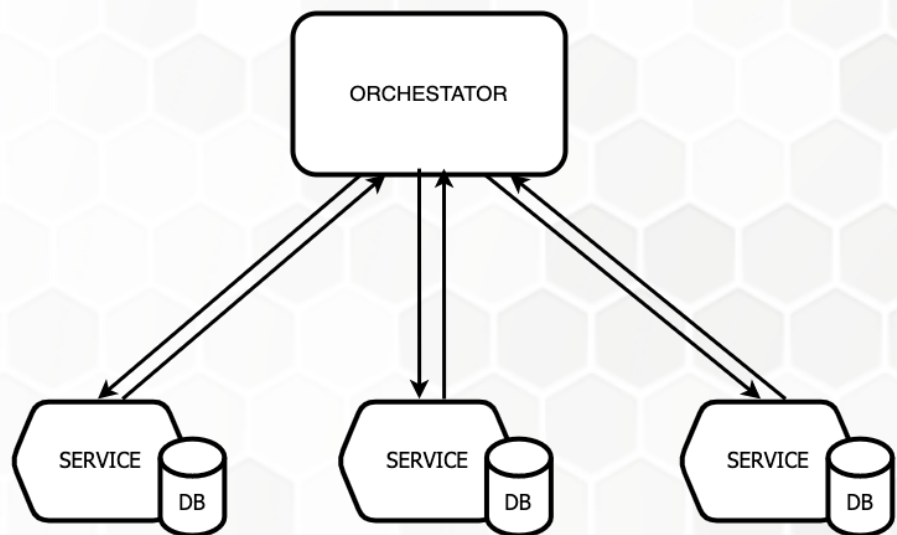
Has reservado un rider ya?

...



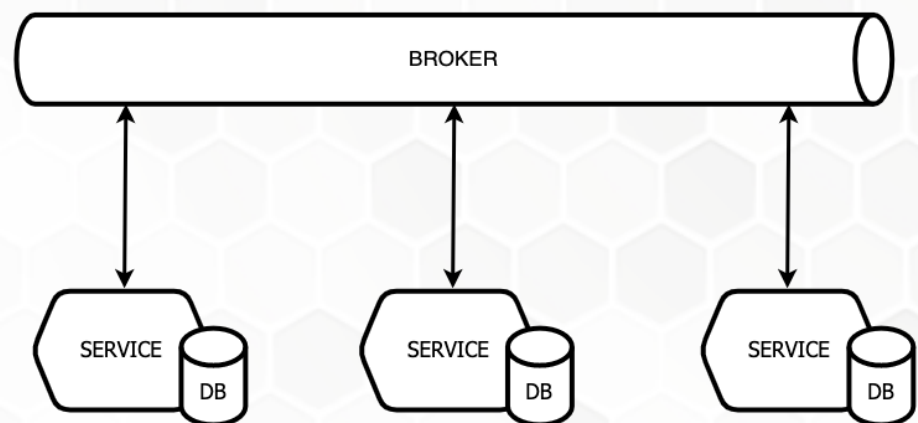
Patron Saga

- Transacción en microservicios
- 1 servicio - 1 transacción
- Si algo va mal rollback de todo
- Asegura Consistencia
- Orquestadas / Coreografiadas



Saga Orquestada

Orquestador llama y espera
Sencilla en procesos sincronos
Acoplamiento de servicios
+ difícil resiliencia y escalabilidad



Saga Coreografiada

Servicios reciben y envían eventos
Desacoplamiento de servicios
+ fácil resiliencia y escalabilidad

Objetivos

Servicios desacoplados

Respuestas multiples asincronas

El middleware no se tiene que preocupar del front

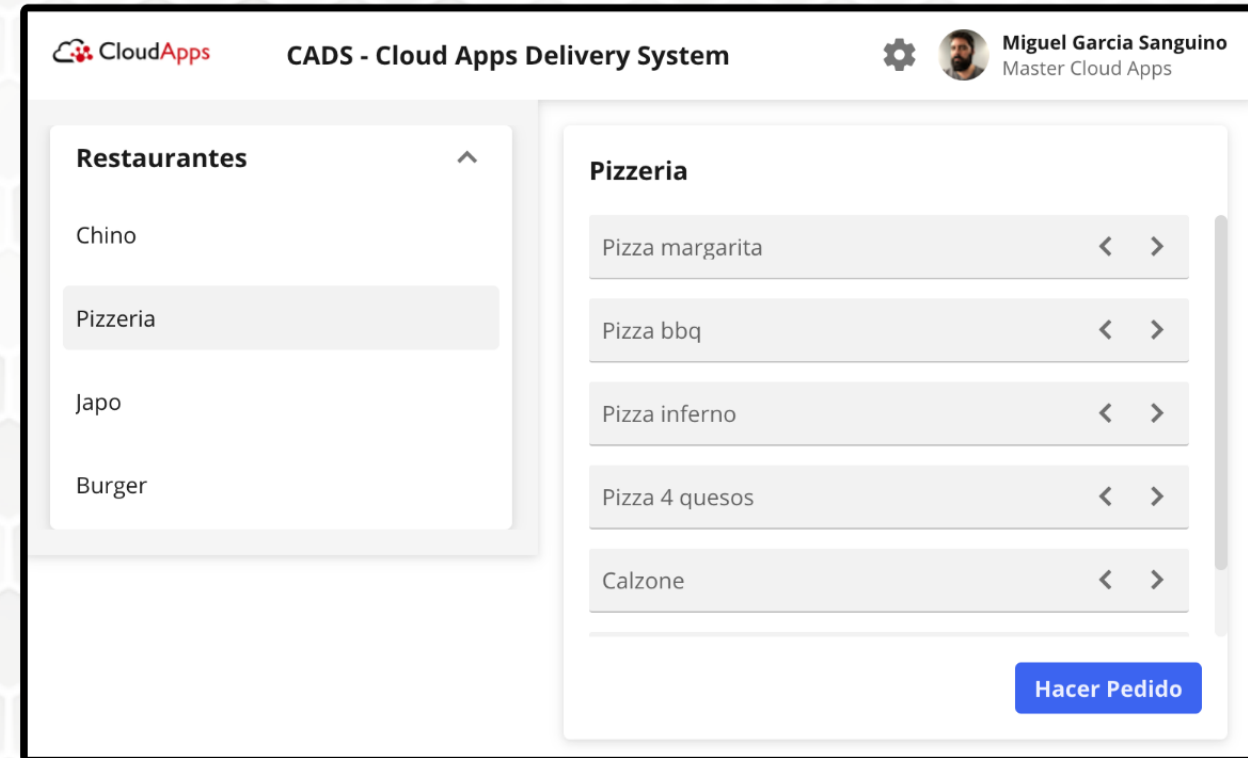
Escalables y resilientes

Desacoplamientos de squads, no solo técnico

Caso de uso

Pedido de comida online

- Asincrono
- Depende de servicios externos
- Actualizaciones multiples
- Transacciones
- Modelos diferentes



Caso de uso

Pedido en curso



Pedido

Pedido creado



Restaurante

pendiente...



Rider

pendiente...

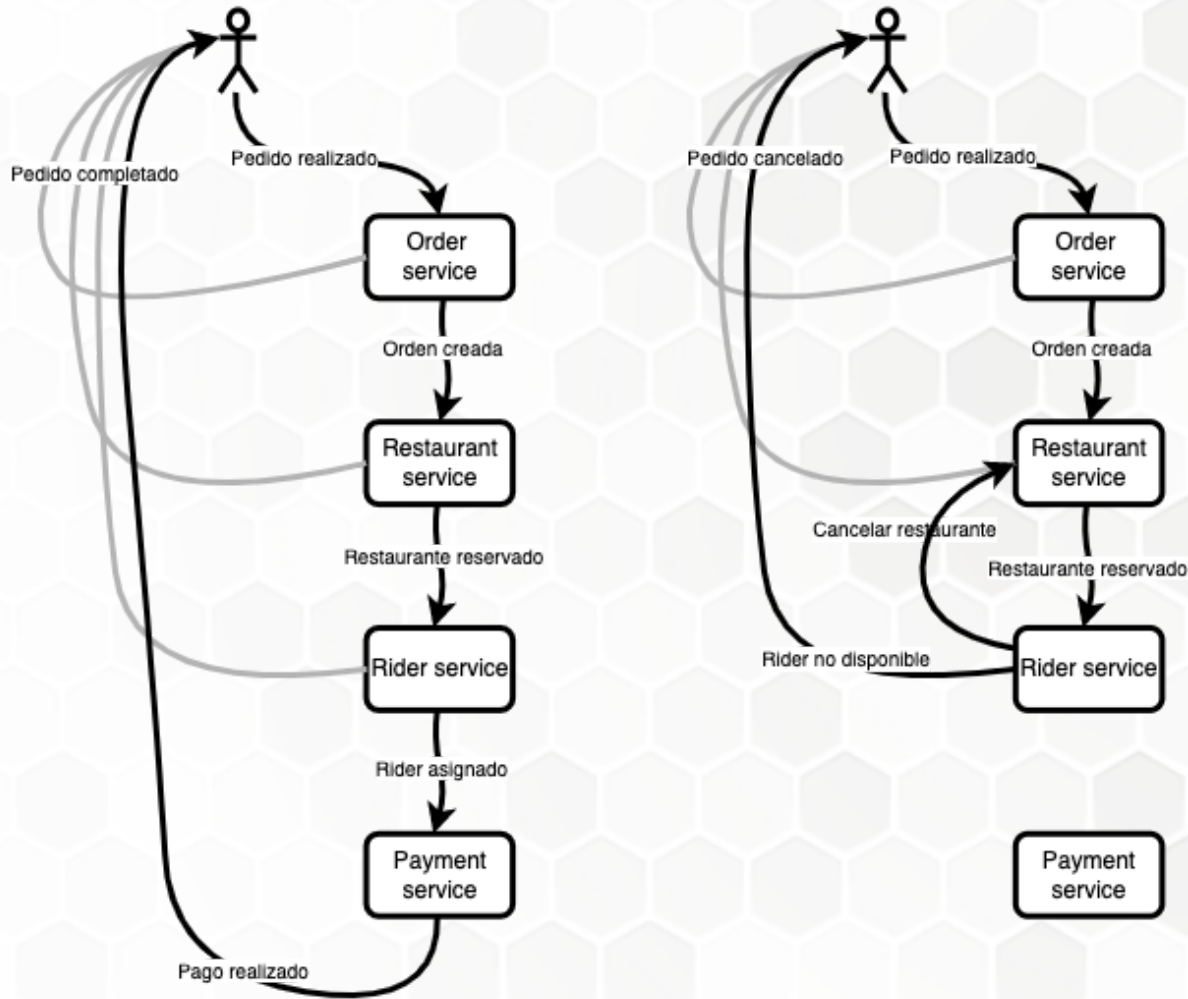


Pago

pendiente...



Detalles del pedido #628a62f52f619df1d53a8906



Completa

Cancelada

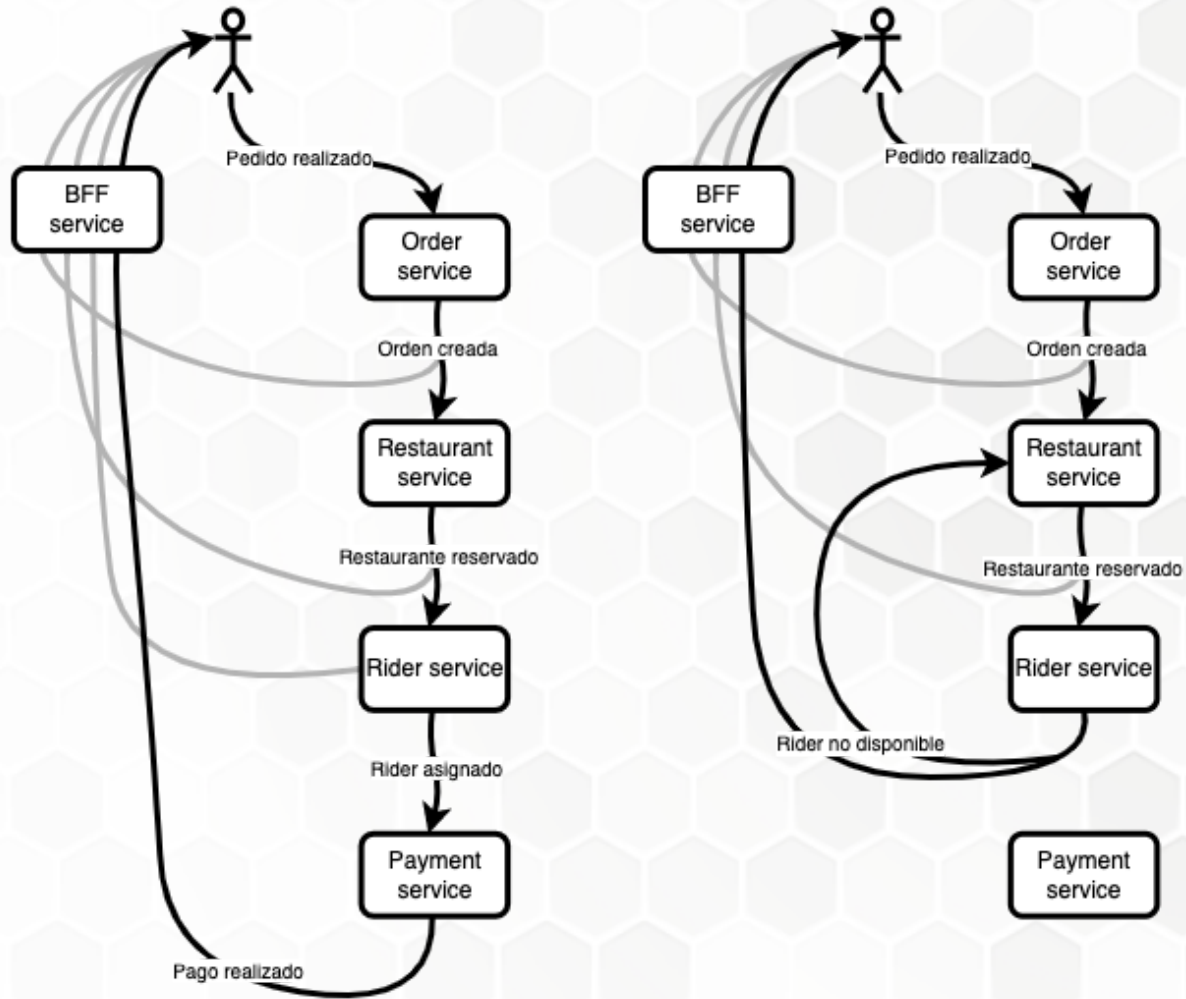
Caso de uso

Saga middleware

Cada paso un servicio

Rollback en caso de fallo

Informar al usuario en cada paso



Completa

Cancelada

Caso de uso

BFF consume los mismo eventos

Cada paso el BFF informa al front

Middleware no hace nada especial

Objetivos:

Servicios desacoplados

Respuestas multiples asincronas

El middleware no se tiene que preocupar del front

Escalables y resilientes

Squads Desacoplados, no solo técnico

Middleware

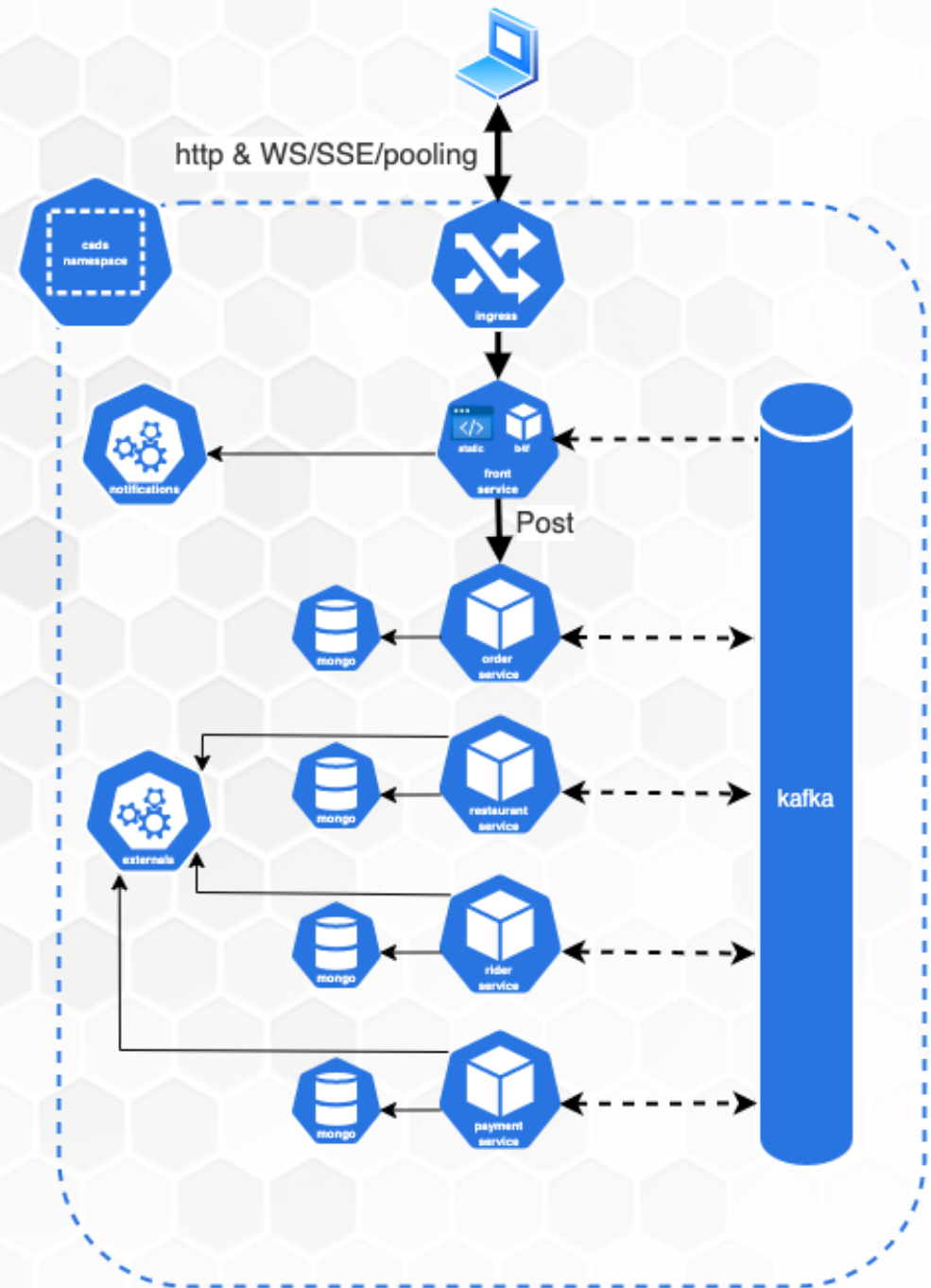
- servicios desacoplados
- saga coreografiada, sin estados
- comunicacion por eventos
- escalables y resilientes

Frontend

- pervertir patrón BFF
- consume de los eventos
- independiente y asíncrono
- notificaciones online / offline

Arquitectura

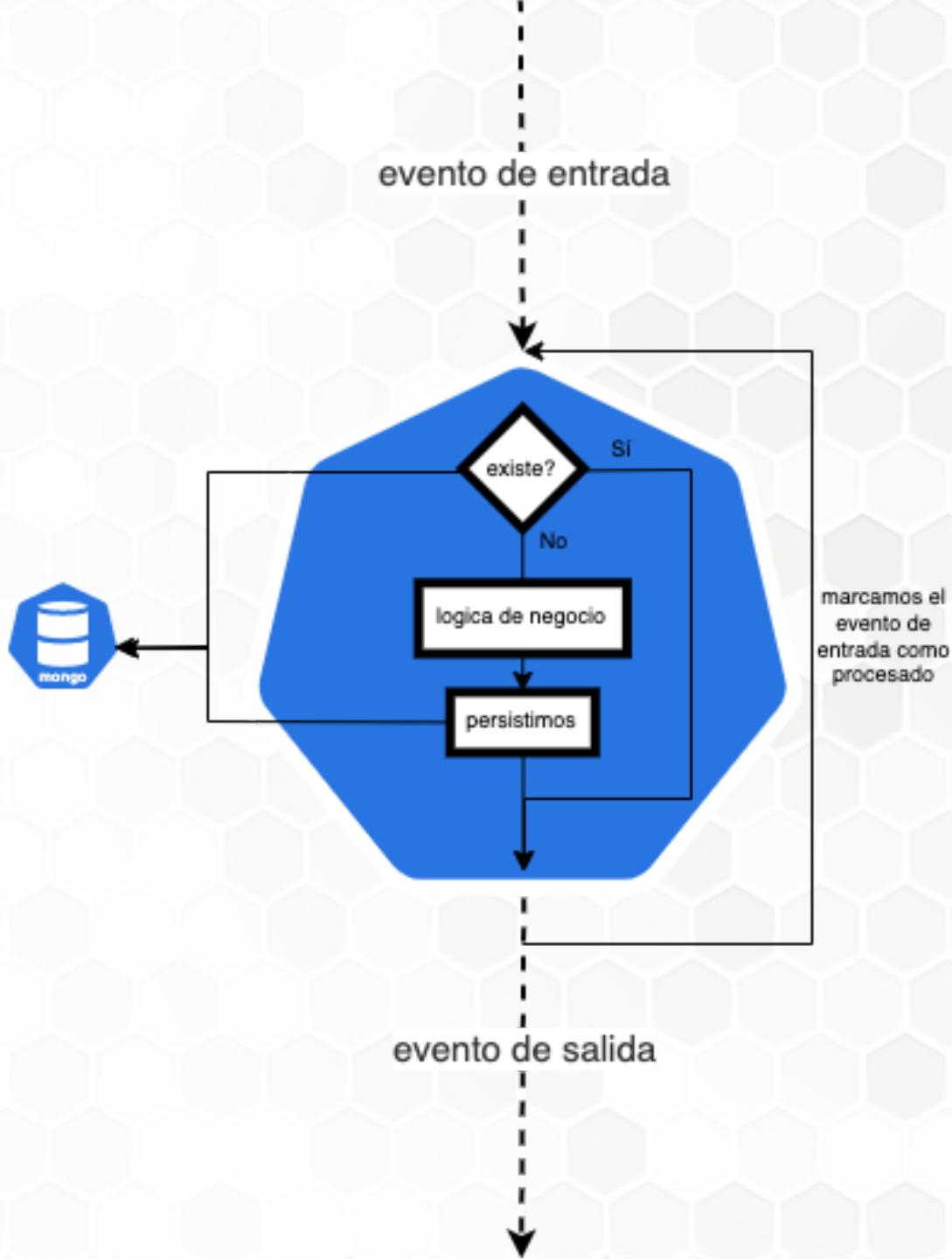
- ingress
- front
- servicios + bases de datos
- externals ~ Mocks
- notificaciones ~ Mocks
- Zookeeper y kafka





Stack

- Kubernetes
- Kafka
- Mongo DB
- Nodejs
- kafkajs
- express
- mongoose
- Rollup como builder
- Lit
- Kor-ui



Idempotencia

Marcamos el offset después de enviar la salida.

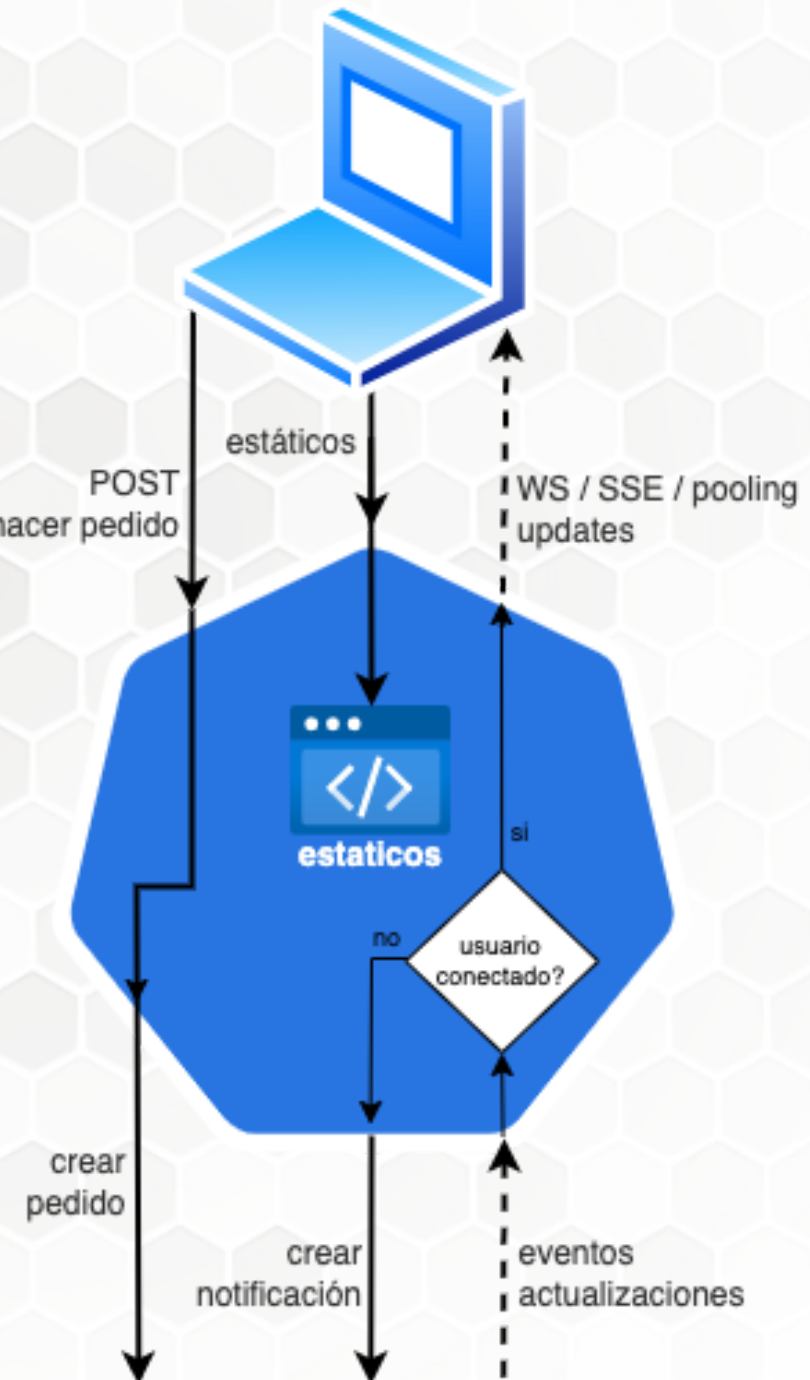
Deben ser idempotentes:

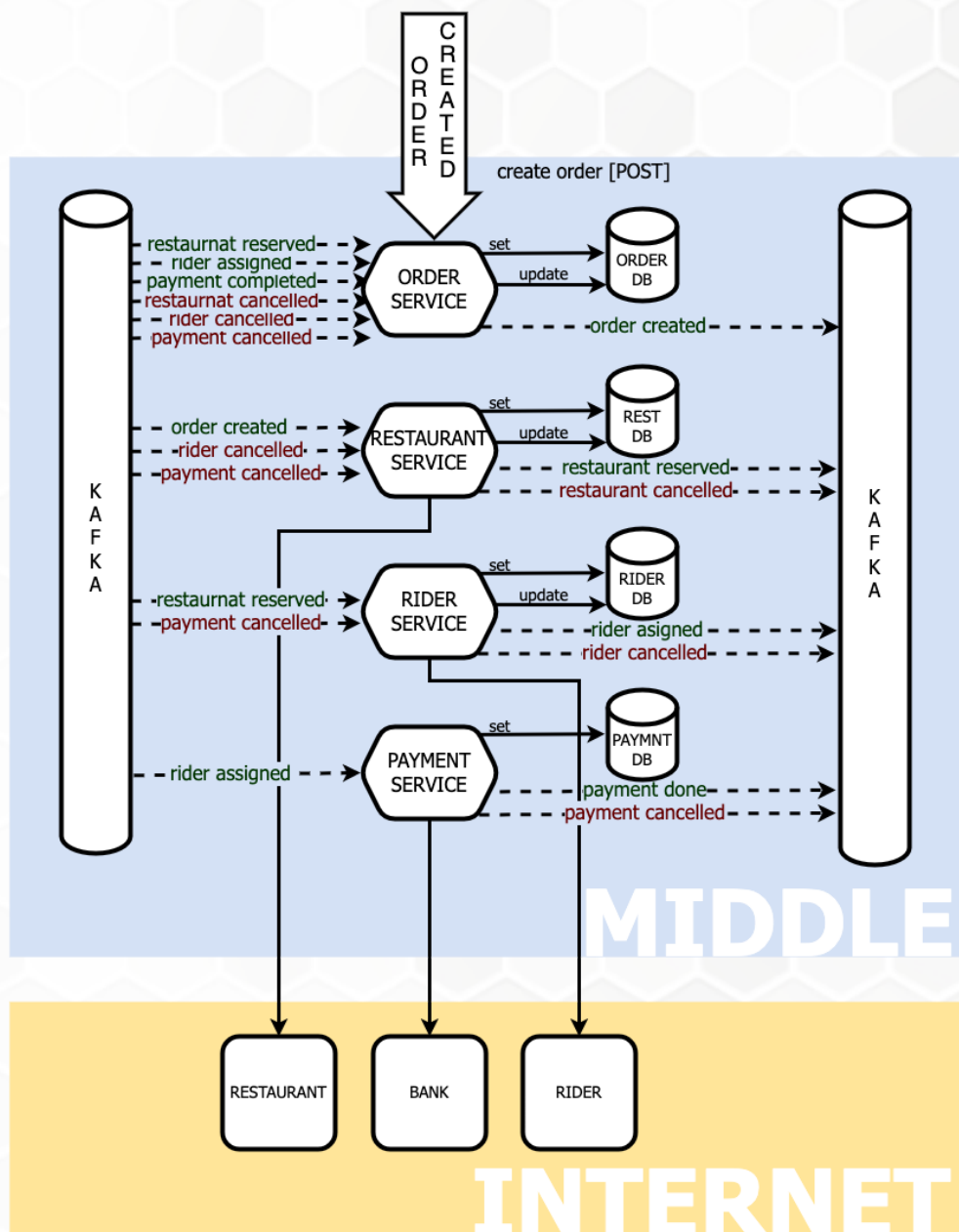
- todos los servicios de la saga
- los servicios externos
- los rollback de la saga

Estáticos y BFF juntos

El contenedor front lleva dentro el servicio BFF y los estáticos

- Los desarrolla el equipo front a sus necesidades
- Agiliza el CI/CD y el testing
- Decide si envía online / offline





Flujo middleware

- único punto de entrada, independiente del consumidor
- servicios no tienen conexiones entre ellos
- Order genera el orderId y audita
- OrderId como correlation id
- Variables de entorno cambiar orden de la saga
- resiliencia y escalabilidad



INTERNET

/api/order [POST]

/api/user/{userid}-{orderid}
[WS / SSE]



- restaurant reserved -
- rider assigned -
- payment completed -
- restaurant cancelled -
- rider cancelled -
- payment cancelled -

ORDER
BFF

send notification [POST]

NOTIFICATIONS
SERVICE

create order [POST]

ORDER
SERVICE

MIDDLE

Flujo Frontend

- backend for frontend
- sin bbdd
- reenvía eventos de middle a front
- convierte eventos en notificaciones
- adapta modelos

Web Sockets vs Server Sent Events vs pooling

- En los 3 casos se queda una conexión abierta, tiempos muy similares
- Pooling descartado por dejar 1 hilo y porque a los 30 seg se repite la petición
- Server Sent Events es REST
- Web Sockets permite bidireccionalidad y datos complejos

DEMO TIME!



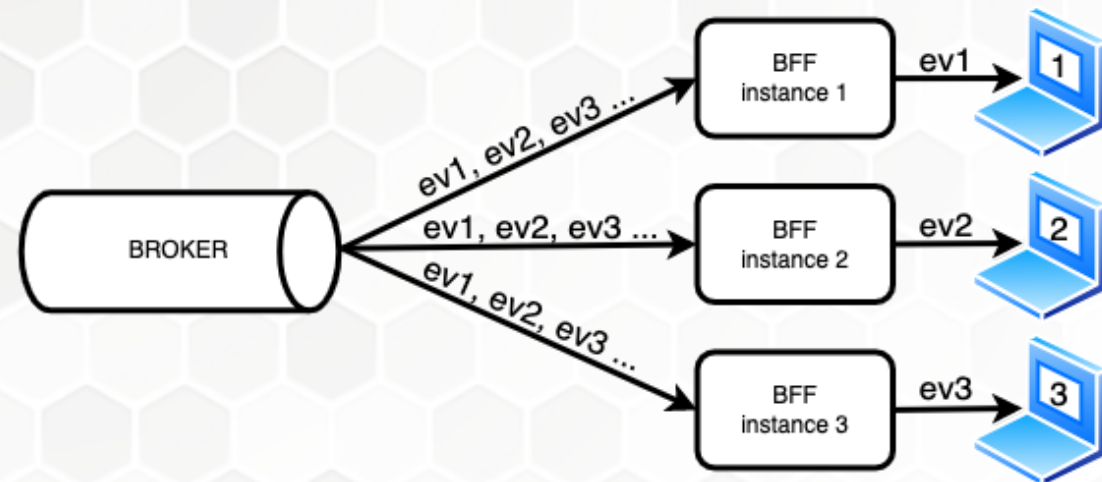
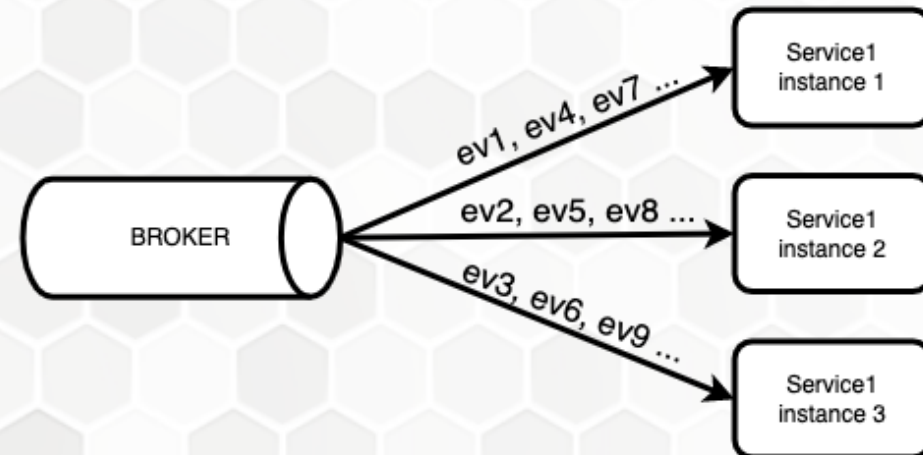
Escalabilidad y resiliencia

Servicios idempotentes, 1 evento 1 instancia.

BFF 1 usuario 1 instancia, 1 evento a todas las instancias.

Si instancia cae, front reconecta

Fallback con notificaciones



Scenario: Payment service rejects as notification

6

- ✓ **Given** access to CADS page
- ✓ **And** externals configured to return 402 when 'post' to 'payment' service
- ✓ **When** create an order
- ✓ **When** user closes before get a response
- ✓ **Then** after 5 seconds, should receive a 'payment rejected' notification
- ✓ **And** mongo status should be restaurant: 5, rider: 5, payment: 4

E2E test

Test E2E en cypress con gherkin.

Cada test configura el api
externals: tiempo y response code
(banco, restaurante, rider).

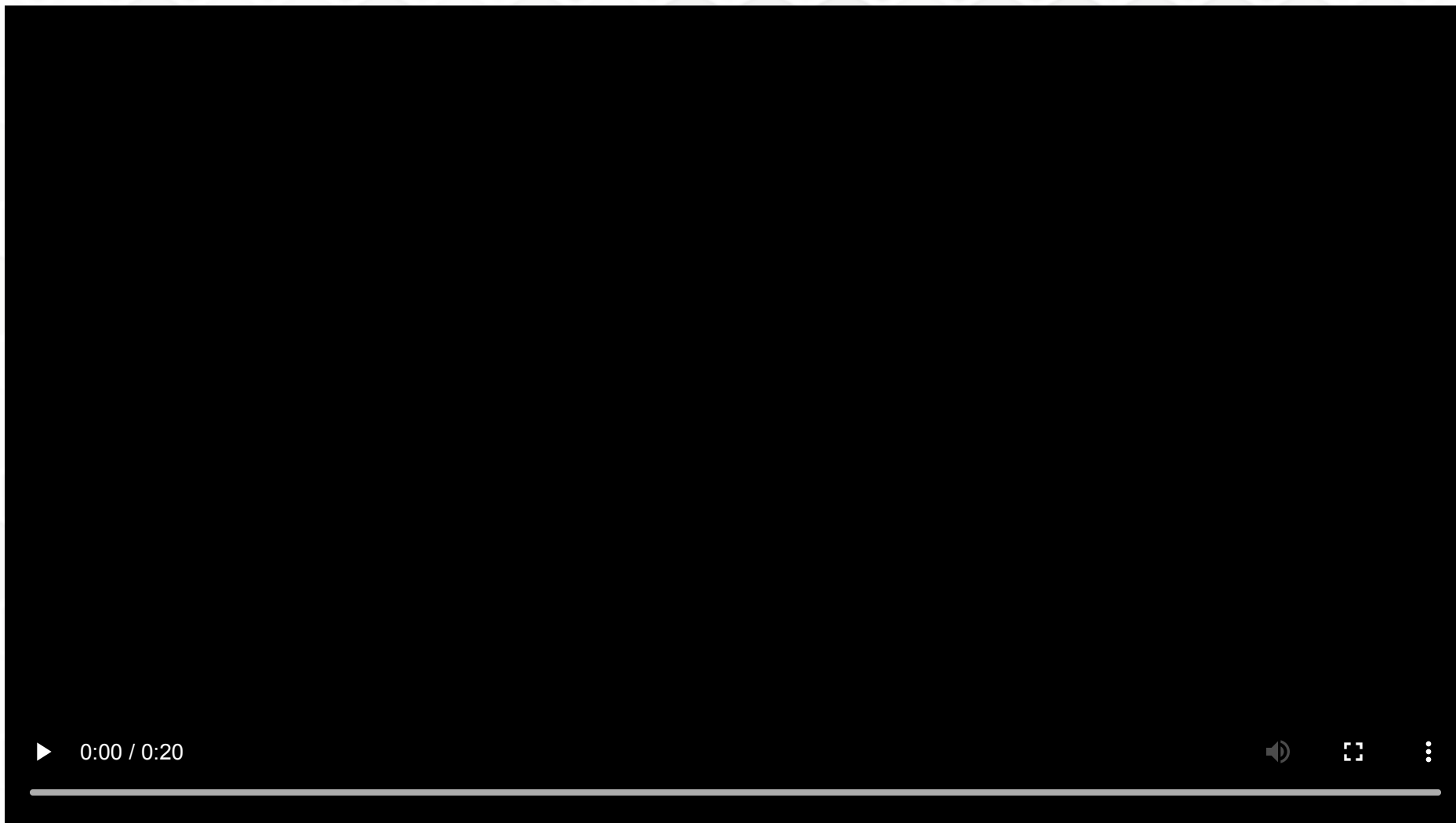
Tests con el usuario online y
offline, check de notificaciones.

Tests comprueban el rollback en
las bbdd.



reportes mocha y cucumber

E2E test video



Conclusiones

Objetivos conseguidos:

- Saga coreografiada con eventos en kafka
- Servicios idempotentes, resilientes, escalables e independientes.
- El frontend consume actualizaciones sin afectar a los servicios
- Las piezas y la arquitectura son muy simples, y muy mantenible
- El patron BFF esta pervertido pero es fundamental
- Los squads apenas tienen dependencias entre ellos más haya del contrato de los eventos
- SSE gana sobre WC y Pooling, aunque por poco

Otros casos de uso para el BFF

- Actualizar datos cuando llegan, tanto eventos como en bbdd
 - Usuario actualiza sus datos en el momento
 - Evita la sobrecarga del middleware
 - Evita recargas o gestion de cache
 - Desacoplamiento middle - frontend



GRACIAS!

Kafka Mongo connect

Pros

envía eventos al persistir en bbdd
simplifica idempotencia

Cons

un servicio más

obliga a tener mongo en replica set de al menos 3 instancias

Finalmente se descarta, no merece la pena

