IT Arkitektur & Infrastruktur

Opgavesæt M7.02 - REST HTTPClient

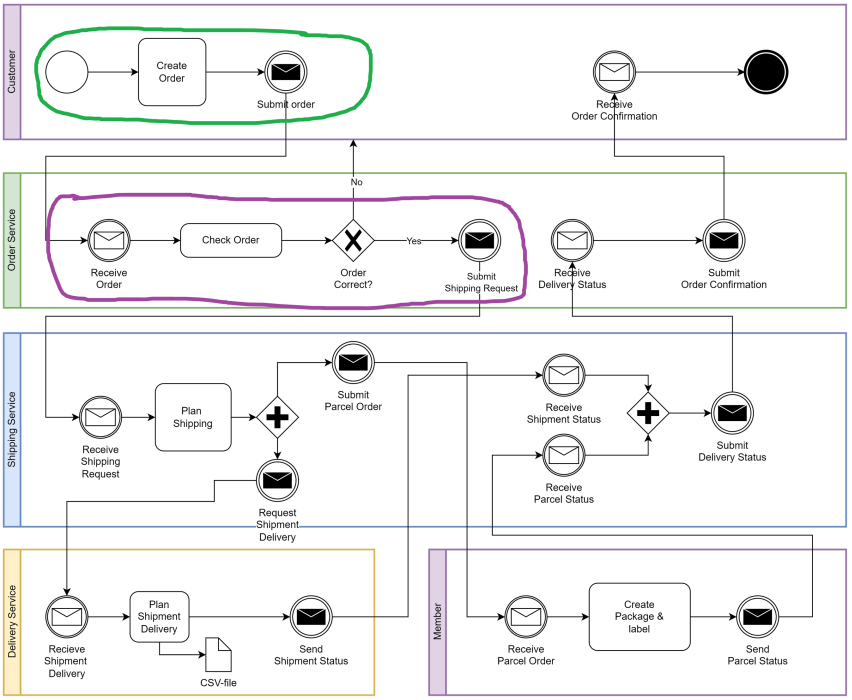
Formål

I denne opgave skal I lave 2 nye microservices til HaaV's Logistik Capability:

1. En `CustomerPortalService` som er en service med en frontend (MVC eller Blazor) til at oprette en ny ordre som vidresendes til `OrderService` (beskrevet herunder) vha. af en `HTTPClient`. I figur 1, længere nede, er denne service fremhævet med grønt.
2. En `OrderService` som modtager har *REST*-endepunkt til at modtage en indkøbsordre fra en kunde. Dette validere indholdet og opretter derefter en forsendelseanmodning (`ShippingRequest`) og sender denne videre til jeres `ShippingService` (fra **opgave M6.02**) vha. en `HTTPClient`. I figur 1, er denne service fremhævet med en lilla farve.

Forberedelse

I jeres GitHub organisation tilføj fordel de 2 services imellem jeres interne udviklingteams opret de nye opgaver på jeres kanban-board (se de specifikke krav længere nede for hver service).



Figur 1 - HaaV Ordrer-proces

> **IHttpClientFactory**
>
> Opgaverne herunder forventer at kommunikationen imellem servicene foregår vha. anvendelse af en C# `IHttpClientFactory`. Find detaljer omkring dens anvendelse her: IHttpClientFactory with .NET.

Krav til OrderService

1. OrderService skal leveres som et selvstændigt microservice i form af et Docker container-image.
2. Projektets kildekode skal versionsstyres og skal følge den fastlagte struktur som anvendes i HaaVs projekter (dvs. anvend processen fra **opgave M5.02**)
3. `OrderService` skal eksponere mindst ét REST API endepunkt som modtager en Kundes bestillingsordre i JSON-formatet. Bestillingsordre skal overholde HaaVs JSON schema for disse
4. Når servicen modtager en bestillingsordre skal den verificeres for korrekt data. Er der fejl i dataene skal der afleveres en passende HTTP-kode.
5. Modtages og valideres en en bestillingordre korrekt, skal der oprettes en forsendelseordre som skal sendes vha. HTTP til `ShippingService`
6. Adressen på `ShippingService` skal være konfigurerbar i det miljø som servicen kører i.

> **Mock af ShippingService**
>
> Da I muligvis ikke har `ShippingService` parat endnu, kan I bruge et standard `Alpine`-image og kommandoen `nc` (se opgave M5.01), til at lave en mock af ShippingService.
>
> Brug følgende `docker-compose.yaml` til at oprette mock'en med:

```yaml
services:
  http-mock-server:
    image: alpine:latest
    command: ["sh", "-c",
        "while true; do (echo -e 'HTTP/1.1 200 OK\r\n Content-Length: 0\r\nConn
        | nc -lp 8000 -k; echo -e '\r\n'; done"
      ]
    ports:
      - "8080:8000"
```

> Når containeren er oppe at køre kan I tilgå port `8080` på `localhost` med en C# app som bruger HTTPClient eller Postman. I kan også afprøve den med en `curl`-kommando fra en bash-terminal:
>
> ```
> $ curl -X POST http://localhost:8080/shipping -d "test af mock"
> ```

For at se outputet i containeren, find **Logs-fanebladet** i jeres Docker App.

Krav til CustomerPortalService

1. OrderService skal leveres som et selvstændigt microservice i form af et Docker container-image.
2. Projektets kildekode skal versionsstyres og skal følge den fastlagte struktur som anvendes i HaaVs projekter (dvs. anvend processen fra **opgave M5.02**)
3. `CustomerPortalService` skal eksponere en HTML side som indeholder en form hvori kunder kan indtaste en bestillingsordre
4. En bestillingsordre skal kunne indsendes til `OrdreService` vha. HTTP
5. Adressen på API endepunktet fra `OrderService` skal være konfigurerbar i det miljø som servicen kører i

---

Ekstra opgave

Lav en Docker compose yaml-fil som samler de 2 services i en fælles løsning. Tilføj evt. en Shipping-mock.

■