**Opdracht 2 (10pt).**

Geef van de verschillende componenten die je in de flow gebruikt hebt aan of ze synchroon of asynchroon zijn. Bij leertaak 3 is de choice naar een asynchroon blok gehaald. Waarom is dit een goede keuze?

De communicatie met de website/webservices gebeurt asynchroon. Dit is een goede keuze omdat dit voorkomt dat een probleem met de website of requests die er lang over doen voorkomen of vertragen dat er berichten naar de infoborden worden gestuurd.

**Opdracht 3 (15pt).**

a. Als je een webservice beschikbaar maakt is het van belang dat op de één of andere manier bekend wordt hoe service benaderd kan worden en waar de service beschikbaar is. We hebben bij leertaak 2 gezien dat bij een SOAP service een contract gepubliceerd wordt in de vorm van een WSDL. Bij Rest API's kan gebruik gemaakt worden van bijvoorbeeld RAML of Swagger (Open API). Maak een vergelijking tussen SOAP en RAML, beschrijf hierin de overeenkomsten en verschillen met betrekking tot de architectuur, gebruikte technieken, noodzakelijke onderdelen van de implementatie, etc.

SOAP more resource intensive

SOAP alleen XML

REST is altijd stateless

RAML is een aparte markup language voor endpoint definitie

b. Wat betekent de term *contract-first* met betrekking tot webservices? In de aangeleverde code genereert de Enpoint.publish het contract, welke tool kun je gebruiken om *contract-first*in Java te gebruiken bij SOAP servers?

Contract-first betekent dat je eerst de openbare definitie van beschikbare methodes (het contract) opstelt, en hier vervolgens je onderliggende applicatie onder bouwt. Je programmeert hier als het ware naar een interface - Java interfaces worden ook wel contracten genoemd.

Spring-WS is hiervoor schijnbaar de tool bij uitstek.

In deze opdracht is het de bedoeling om de logging en de synchronisatie van de tijd tussen de infoborden en de bussimulator te realiseren. De java en Python programmatuur die hiervoor benodigd is zit al in de startcode. De verbinding tussen de applicaties en een deel van de logging moet in Anypoint worden gerealiseerd. Kijk voor je aan de opdracht begint welke informatie in het bijgevoegde document wordt aangeleverd.

**Opdracht 1 (10pt.)**  
Het eerste onderdeel van de opdracht is het realiseren van tijd-synchronisatie tussen de simulator en het infobord. In de bij opdracht 1 aangeleverde code is de benodigde functionaliteit voor het aanroepen en verwerken van de tijdsynchronisatie al aangeleverd als comments in de code. Je kunt deze regels vinden in de Runner klasse van de Simulator,  en JSONBericht en Infobord klasses in het InfobordSysteem. Als je deze regels hebt toegevoegd aan het project zijn de applicaties in principe klaar voor synchronisatie.  
De applicaties zullen nu een GET uitvoeren naar de url http://localhost:8081/TijdServer?responseType=xml|json. De simulator vraagt om een XML bericht en het infobord om een json bericht.  
Het python programma TimeServer.py dat ook in de startcode aanwezig is, bevat een applicatie die een server opstart die via de url http://localhost:5002/TimeServer de tijd (sinds het opstarten van de server) retourneert. Dit gebeurt als json-bericht met de attributen uur, minuut, seconde.   
Maak in Anypoint een flow die de GET requests afhandelt door de tijd op te halen van de timeserver en op basis van het gevraagde type een XML of een json bericht terug stuurt. Als het goed is heb je aan het einde van deze opdracht een extra flow in je Anypoint project die er ongeveer uitziet zoals het onderstaande plaatje. Je kunt gebruik maken van de bij deze opdracht aangeleverde voorbeeld formaten om de metadata van de request en de output van de transform message te definiëren.

**Opdracht 2 (15pt.)**  
Het tweede deel van de opdracht is het realiseren van de logging van de berichten. De drie bedrijven die bussen laten rijden (ARRIVA, QBUZZ en FLIXBUS) hebben alle drie een eigen methode van loggen. ARRIVA verwerkt de berichten en slaat ze via een java-applicatie op in een database (in de startcode file zit een mock-applicatie die deze verwerking simuleert). Deze java applicatie luistert naar de queue ARRIVALOGGER. Om de belasting van de (fictieve) database te reguleren draait deze applicatie niet continu, maar verwerkt deze de berichten in een batch. Deze logger kan via het Dashboard worden gestart.

De QBUZZ berichten worden via de in opdracht 2 en leertaak 3 gerealiseerde SOAP services verwerkt.

De berichten van FLIXBUS ten slotte moeten rechtstreeks vanuit Anypoint in een database worden weggeschreven. In deel 2 vaan leertaak 3 wordt uitgelegd hoe je een database-connector gebruikt in AnyPoint.

De gegevens uit het bericht moeten worden samengevoegd tot een tekstveld dat er als volgt uitziet:

*BusId*|*eindpunt*|*aankomsttijd*|[*ETA-halte*|*ETA-richting*|*ETA-aankomsttijd*].

Waarbij het stuk tussen de vierkante haken een herhalende groep is. Een record in de database ziet er dus bijvoorbeeld zo uit:

4LIJN7-1|A|17588|X|-1|17594|Y|-1|17599|G|-1|17606|Z|-1|17618|N|-1|17635|O|-1|17640|M|1|17649|L|1|17660|Q|-1|17666|R|-1|17673|A|1|17683|

Breidt de flow die je in de eerste twee opdrachten hebt gemaakt uit zodat deze drie verschillende manieren van logging worden gerealiseerd.

**Opdracht 3 (15pt.)**  
Maak een schema van het uiteindelijke applicatie landschap. Geef een overzicht van de verschillende informatie stromen en bespreek aan de hand hiervan de volgende punten:

* Wat voor technieken worden er gebruikt voor de verbindingen.
* Zijn er eventueel alternatieve technieken die beter geschikt zijn.
* Welke patterns worden er gebruikt.
* Beschrijf welke rol AnyPoint heeft in het proces (mediation of orchestration of beide).
* Wat voor technieken worden er gebruikt voor de verbindingen.
* Zijn er eventueel alternatieve technieken die beter geschikt zijn.
* Welke patterns worden er gebruikt.
* Beschrijf welke rol AnyPoint heeft in het proces (mediation of orchestration of beide).