2017-2018 Software engineering 3

Examenproject – 2e zittijd

Order picking

Inleiding

Dit document beschrijft het examenproject van het vak Software engineering 3 voor de **tweede zittijd** academiejaar 2017-2018. Dit project wordt **individueel** uitgewerkt (beide delen, zowel generator als order picker) . Het wordt afgegeven, gedemonstreerd en toegelicht op de dag van het examen (zie Verloop van het examen).

Beoordelingscriteria

- Een goed begrip van de tijdens de les aangebrachte concepten (zie slides)
- Een correcte toepassing hiervan in het project
- Cleane en onderhoudbare code (met inbegrip van logging en error handling) die voldoet aan de standaard Java code conventies en zinvol gedocumenteerd is.
- De beheersing van het project: het ontwerp, de code, de te verwachten uitbreidingen en hoe de code hierop voorzien is.

Opdracht

Ontwerp en implementeer een systeem voor de samenstelling (*picking*) van bestellingen (*orders*) in de backoffice van *WarmRed*, een grote online winkel. Dit omvat een aantal aspecten die in wat volgt worden omschreven.
Met *instelbaar* wordt hierbij steeds bedoeld: instelbaar vanuit de test code, daar waar alle objecten worden geassembleerd en de toepassing wordt gestart (zie de slides rond factory/wiring).

A. Generator

De *generator* is een standalone Java applicatie die wordt gebruikt om test orders aan te maken. Hiermee kan de logica en schaalbaarheid van de Order picker worden uitgetest zonder dat deze hoeft

gekoppeld te zijn aan de web applicatie waarop de orders in realiteit worden aangemaakt. De generator moet gestart en gestopt kunnen worden. Na het starten wacht hij telkens een instelbaar aantal milliseconden alvorens hij een nieuwe order genereert.

Elke order bevat volgende informatie en wordt op de aangegeven manier gegenereerd:

- **OrderID** – integer (7 cijfers) – oplopend vanaf 1000000 met elke nieuwe run van de generator
- **CustomerID** – integer (7 cijfers) – random geselecteerd uit een instelbare collectie van klantnummers
- **Price** – integer – random tussen twee instelbare grenzen.
- **Items** – collectie – aantal items is random tussen twee instelbare grenzen
    - **ProductID** – integer (7 cijfers) – random geselecteerd uit een instelbare collectie van productnummers.
    - **Amount** – integer – random tussen twee instelbare grenzen

Het gebeurt dat orders door klanten geannuleerd worden. Om dit te simuleren produceert de generator af en toe een annulatie bericht. Dit bevat volgende gegevens:

- **OrderID** – Integer (7 cijfers) – id van de te annuleren order.

Het is instelbaar via 2 integers hoe vaak een annulatie bericht moet uitgestuurd worden. Het eerste getal geeft aan om de hoeveel orders dit moet gebeuren (bv. elke 300 orders). Het tweede getal geeft aan hoeveel seconden er moet zitten tussen de generatie van de eigenlijke order en de annulatie (bv. na 360s).

B. Message broker

Om hoge frequenties van orders te kunnen bufferen voor verwerking in piekmomenten (kerstperiode,…), wordt gebruik gemaakt van een **message broker** tussen de webshop/generator en de orderpicker. Elke order en order annulatie wordt als een aparte XML message op de queue gezet door de order generator en uitgelezen door de orderpicker.

C. Order picker

De **orderpicker** is net zoals de generator opgezet als een standalone Java applicatie. De applicatie heeft als taak om binnenkomende orders van de queue te halen, elke order te voorzien van locatie info (zie verder), picking optimalisatie toe te passen (zie verder) en vervolgens per order een met locatie info verrijkt order bericht op een uitgaande queue te plaatsen. Deze queue is bedoeld om uitgelezen te worden door een mobiele applicatie voor de order pickers. Dit zijn magazijniers die orders samenstellen door de items uit de rekken te halen en de order in te pakken voor verzending. De mobiele applicatie zelf hoeft niet gebouwd te worden. Een afdruk (System.out) van wat er op de uitgaande queue komt volstaat. De precieze structuur en het formaat van de berichten op de uitgaande queue mag vrij worden bepaald als onderdeel van de opdracht.

Met *voorzien van locatie info* wordt bedoeld dat voor elk item in de order in het uitgaande bericht is aangeven waar het betreffende product zich bevindt in de magazijnen van WarmRed. De locatie (magazijn_nr, gang_nr, rek_nr) van een product kan worden opgevraagd via de (synchrone) proxy

van een *LocationService* (zie zip file op BB). Opgevraagde locaties dienen in een in memory cache bijgehouden te worden om onnodige netwerk trafiek te vermijden (de locatie van een product in een magazijn wijzigt immers zelden). Het moet *instelbaar* zijn na welke tijd deze cache automatisch wordt leeggemaakt.

Om ervoor te zorgen dat de pickers zo weinig mogelijk afstand afleggen bij het samenstellen van een order, wordt aan *picking optimalisatie* gedaan. Dit dient als volgt ingesteld te kunnen worden:

- **Single**. Optimalisatie gebeurt op elke order apart. Het optimalisatie algoritme zet de order items (met bijhorende locatie info) in een optimale volgorde[1].
- **Group**. Orders worden gebufferd in memory tot een instelbaar aantal. Op deze groep van orders wordt vervolgens een algoritme uitgevoerd dat de orders in een optimale volgorde zet[2]. Op elke order in de groep wordt ook 'Single' optimalisatie toegepast.

De optimalisatie algoritmes zelf hoeven niet geïmplementeerd te worden. Je voorziet voor beide gevallen wel een test implementatie die de items/orders gewoon in 'een' andere volgorde zet (bv. reverse).

Het kan voorkomen dat een order item een productID bevat dat niet bekend is bij de LocationService of dat deze niet bereikbaar is en de gevraagde locatie nog niet in de cache zit. In deze gevallen wordt de order in een database van onverwerkte orders gestopt. Met een instelbare frequentie (bv. elke 120s) worden de orders uit deze database opgehaald om ze opnieuw te verwerken. Er hoeft geen echte database gebruikt te worden, een in memory Java map volstaat.

Wanneer de annulatie van een order binnenkomt en de bijhorende order is nog niet verwerkt (bv. omdat hij nog in de 'Group' buffer zit of in de map met gefaalde orders) wordt deze order verwijderd en niet verder behandeld.

D. Toekomstig verwachte uitbreidingen

Volgende wijzigingen/uitbreidingen kunnen verwacht worden in de toekomst. Deze hoeven niet geïmplementeerd te worden, maar de code moet er op voorzien zijn dat dit makkelijk kan gebeuren.

- Wijzigingen aan de API van de LocationService
- Gebruik van een ander type broker
- Andere mechanismen (dan random) voor het genereren van testberichten (bv. laden uit een file)
- Andere manieren en algoritmes om aan picking optimalisatie te doen
- Gebruik van een echte database voor gefaalde orders
- Melding (via queue) van orders die niet geannuleerd kunnen worden.

[1] bv. eerst de 2 koptelefoons die vlak bij elkaar in de rekken liggen, dan de ipad die in een naburig rek ligt, enz…
[2] bv. orders met enkel multimedia items kunnen best direct na elkaar gepicked worden en daarna pas een kleding order (zeker indien kleding zich in een ander magazijn of aan de andere kant van het magazijn bevindt)

E. Niet-functionele vereisten

- Er wordt kwaliteitsvolle **code** opgeleverd voor zowel de generator als de order picker. UML kan hierbij een hulpmiddel zijn.
- De 2 applicaties zijn onafhankelijk van elkaar, ze delen geen code en communiceren enkel via de message broker. Alle verwerking van gegevens gebeurt verder volledig in memory, er wordt geen externe databank gebruikt. Als broker wordt een open source messaging systeem zoals RabbitMQ of ActiveMQ gebruikt (cloud of lokaal geïnstalleerd naar keuze). Het berichtenformaat op de broker is XML.
- De code wordt geschreven in Java - in het Engels[3]. Je gebruikt enkel de standaard JDK[4]. Je gebruikt geen externe code/library's/… behalve voor het benaderen van de message broker, formaat conversie tussen JSON/XML/Java en voor logging (zie verder). Voor JSON en XML conversie mag gewerkt worden met een library naar keuze (org.json, javax.json, castor,…).
- Logging mag gebeuren
    - ofwel op System.out en dit in een static method van een zelf te schrijven Logger class waaraan info (message, level, exception) kan doorgegeven worden vanuit code die de log wil schrijven.
    - ofwel via slf4j-log4j, log4J of een andere Java logging library
- Het gebruik van gradle of maven is toegelaten maar niet verplicht.
- Het uitwerken van unit testen is optioneel.

Oplevering

Het project wordt individueel uitgewerkt en afgegeven door dit voor de aanvang van het examen door te sturen naar het e-mail adres be.kdg.inf.se3@gmail.com. Dit e-mail adres mag enkel gebruikt worden voor het doorgeven van de oplevering. Je stuurt een (drive, dropbox,…) **link** door naar een **zip** (geen rar, niet als attachment). De naam van de zip is als volgt: achternaam_voornaam.zip.

Verloop van het examen

Het examen duurt ongeveer 40 minuten (waarvan 20' voorbereiding)

- Voor binnenkomst zet je op je laptop de 2 projecten open (in IntelliJ of andere IDE).
- Je komt binnen zodra een vorige student het lokaal verlaat, tekent de lijst en trekt 2 vragen die je uitschrijft op papier, gesloten boek. De vragen handelen over concepten uit de cursus. Zie BB voor een overzicht van de mogelijke vragen.
- Wanneer de vorige student klaar is met zijn verdediging, koppel je je laptop aan de beamer. Het project wordt toegelicht, bevraagd en beoordeeld (70%). De 2 vragen worden overlopen en besproken met eventuele bijvragen (30%).

Veel succes!

[3] Je wordt in dit vak niet beoordeeld op de kwaliteit van het Engels op zich, wel op de kwaliteit van de code/commentaar
[4] Spring (boot) mag indien gewenst gebruikt worden