Hanzehogeschool Groningen
Instituut voor Informatie
en Communicatietechnologie

Thema 4.2I Gedistribueerde applicaties
Practicumopgaven week 3

Opgave I– classic REST

Vorige week heb je een aantal opgaven gemaakt rondom webservices. Dit waren “klassieke” webservices, dat wil zeggen, WSDL wordt gebruikt om de beschikbare services te beschrijven, waarbij SOAP als message formaat wordt gebruikt. Nadelen van deze aanpak zijn:

- ✓ WSDL beschrijvingen en SOAP berichten zijn relatief groot in relatie tot de achterliggende functionaliteit en de “payload” van een bericht;
- ✓ WSDL moet de hele webservice correct beschrijven, met als gevolg dat een toevoeging aan de beschikbare webmethoden een volledige hergeneratie van het bijbehorende WSDL bestand nodig is.

Je ziet de laatste jaren de opkomst van een alternatieve webservice standaard: REST. Met deze standaard hebben client en server meer vrijheid in het afspreken van het formaat van de berichten.

a. Leg de algemene filosofie achter REST uit aan de hand van internetbronnen. Gebruik in je uitleg de term *resource* en beschrijf ook hoe een URL er uit kan zien bij het aanroepen van een resource.

b. Leg uit hoe je met REST de bovenstaande problemen oplost.

Om je een beeld te geven hoe een REST webservice binnen Netbeans geïmplementeerd is kun je StudentData.zip downloaden van Blackboard. Run het bijbehorende student.sql script en pas de inloggevens in `StudentDAOImpl` aan voor je eigen databaseinstelling.

c. Open het project, build, deploy het en kies vanuit het project context-menu “Test RESTful Web Services”. Test de verschillende methoden en paden. Leg ook het verband uit tussen de vorm van de URL’s en je antwoord op vraag a).

d. Wat is de rol van de twee binding klassen? Gebruik in je antwoord de term (un)marshalling.

Opgave 2 GBA – moderne REST webservice

Veel webservices die tegenwoordig RESTfull worden genoemd, zijn dit feitelijk niet. Je ziet namelijk dat HTTP verbs niet worden gebruikt. Uit de URL blijkt wat de actie (opvragen, wijzigen en verwijderen) is. Een voorbeeld is

StudentData/Student?action= get&stdnr=123456.

Hiermee wordt dus het idee verlaten dat de URI alleen de naam van de resource bevat en geen parameters;

In het vervolg van de opgave ga je een GBA REST webservice – in het project maken we gebruik van meer zuivere REST.

a. Download `GBA.zip` van Blackboard. Er is al een kleine dummy opzet gemaakt voor de “get” action. Je ziet een scheiding tussen code (`doGet` methode) en de view (get.jsp).

b. Vul het NatuurlijkPersoon domain object aan zodat de volgende velden aanwezig zijn:

```
bsn: type long
geslacht: type char
initialen: type String
achternaam: type String
straatnaam: type String
nummer: type int
postcode: type String
woonplaats: type String
```

c. Maak een bijbehorende database in MySQL en definieer een interface NatuurlijkPersoonDAO (met slechts één methode `getNatuurlijkPersoonbyBSN(long bsn)`) en implementeer de bijbehorende `NatuurlijkPersoonDAOImpl` klasse. Als er geen Natuurlijk Persoon bestaat bij een bsn dien je een `NatuurlijkPersoon` object terug te geven met bsn -1.

d. Implementeer de volledige actie en view die bij de “get” action hoort. Run het programma en demonstreer de werking aan je practicumdocent.

Opgave 3 – Beveiliging

Je hebt in de vorige vraag alleen een methode geïmplementeerd om gegevens op te halen. Het moet uiteraard mogelijk zijn om persoonsgegevens te plaatsen. Echter, we gaan er van uit dat niet iedereen die de GBA gegevens kan raadplegen deze ook mag plaatsen. Je kunt de REST webservice op verschillende manieren beveiligen, maar in deze opgave beperk je je tot encryptie en hashing. We gaan eerst wat algemener naar beveiligingsaspecten kijken.

In beveiliging kun je een aantal basisdiensten herkennen. Een mogelijke, niet uitputtende, lijst is:

- ✓ Vertrouwelijkheid
- ✓ Authenticatie
- ✓ Integriteit
- ✓ Onweerlegbaarheid

a. Geef een definitie van de vier bovenstaande diensten. Geef bij elke dienst een voorbeeld waarom je de dienst zou willen gebruiken.

Daarnaast bestaan (onder andere) de volgende basistechnieken:

- ✓ Digitale handtekening
- ✓ Hashing
- ✓ Symmetrische encryptie
- ✓ Asymmetrische encryptie

b. Beschrijf kort de werking van bovenstaande technieken.

c. Welke technieken uit de vorige vraag horen bij het bovenstaande lijstje?

d. Waarom wordt asymmetrische encryptie niet vaak gebruikt om grote hoeveelheden data te versleutelen?

Download `AES128Encryption.zip`. AES (Advanced Encryption Standard) is een officiële encryptie standaard en geïmplementeerd door het Rijndael algoritme. Zie wikipedia voor meer informatie. Run `AESCodecTester` om te zien hoe AESCodec gebruikt kan worden voor encryptie en hashing.

e. Leg het encryptiedeel van de encode methode in AESCodec uit aan de hand van de volgende subvragen.

   I. In deze methode wordt blijkbaar de ECB methode van AESE gebruikt. Wat is de ECB modus?

   II. Als je goed naar de implementatie van `encode` kijkt zie je dat er geen ECB modus wordt toegepast maar een andere encryptiemodus. Welke modus is dat? Bestudeer de wiki-pagina over Block Cipher Modes voor je antwoord. Uiteraard heeft de `decode` methode een “gespiegelde” implementatie.

f. Leg uit hoe in de methoden `encode` en `decode` van AESCodec SHA hashing wordt toegepast.

g. Implementeer de “add” action. Vul ook de `NatuurlijkPersoonDAO` interface en de implementatie `NatuurlijkPersoonDAOImpl` aan.

h. Implementeer de `doPost` methode van de GBA servlet en ga daarbij uit dat de meegestuurde JSON gegevens in de body van het POST verzoek gecodeerd zijn met AESCodec.

i. Stuur een gecodeerd JSON bericht terug waarin de status wordt vermeld: OK of NOT OK. Een status is niet OK als:

   ✓ De sleutel of hashing verkeerd was;

   ✓ Het BSN nummer al bestaat.

Opgave 4 - BBS

In de komende twee opgaven ga je het BBS en OLA systeem maken. Daarna komt de ontsluiting van die systemen via messaging aan bod.

In deze opgave ga je allereerst aan de slag met het maken van een boetesysteem in MySQL en de ontsluiting daarvan. De businessrules achter een boete zul je met behulp van een stateless session bean implementeren. Lees de relevante hoofdstukken in The Java EE 6 Tutorial (die je op Blackboard kunt vinden) en probeer een aantal voorbeelden. Lees daar ook de opmerking over de updatetool waarmee je de tutorial voorbeelden kunt downloaden.

Het boetebedrag dat bij een snelheidsovertreding hoort, wordt aan de hand van de volgende gegevens bepaald:

Snelheidsovertreding (km/u)	Boetebedrag (€)	Boetezwaartepunt
1-4	25	1
5-9	50	1
10-14	75	2
15-19	100	2
29-29	150	3
30-39	250	4
40-49	400	5

Daarnaast gelden de extra regels:

✓ Een boete gegeven in een gebied waar niet harder dan 50 km/u (bijvoorbeeld in de bebouwde

kom) mag worden gereden krijgt een opslag van 100%.

- ✓ De opgetelde hoeveelheid boetezwaartepunten binnen één kalenderjaar, terugrekenend vanaf de overtreding die tot deze boete leidt, mag niet meer dan vijf zijn. Elk extra boetezwaartepunt boven de vijf levert een opslag van 50% op.

De regels worden in deze volgorde toegepast. Boetes zijn uiteraard gekoppeld aan natuurlijke personen en niet aan het voertuig.

Een chronologisch voorbeeld om deze regels te illustreren (tip: je kunt van deze voorbeelden natuurlijk makkelijk unit-tests maken):

- ✓ 28 februari: Michael rijdt succesvol af en koopt dezelfde dag een scheurijzer.
- ✓ 1 maart: Michael rijdt 54 km/u waar 50 km/u is toegestaan: een overschrijding van 4km/u die een boete van 25 +100% = 50 euro en totaal 1 boetezwaartepunt oplevert.
- ✓ 19 oktober: Michael rijdt 152 km/u waar 120 km/u is toegestaan. De overschrijding is dus 32 km/u en levert een boete van 250 euro en totaal 1+4=5 boetezwaartepunten op.

✓26 mei: Michael rijdt 75 km/u waar 30 km/u is toegestaan. De overschrijding is 45 km/u. Regel 1 wordt eerst toegepast en leidt tot een boete van 400 + 100% = 800 euro. Vervolgens wordt met regel 2 bepaald dat Michael 4+5=9 opgetelde boetezwaartepunten heeft, immers de eerste overtreding van 1 maart is vervallen in de telling. Dat is vier meer dan is toegestaan, dus de totale boete is voor deze overtreding is 800 + 4 * 50% = 800 + 200% = 2400 euro. De negen boetezwaartepunten blijven staan.

a. Maak een database boeteregistratie dat gebruikt kan worden door BBS.

b. Implementeer bovenstaande code in een stateless sessionbean. Gebruik JPA voor de database OR mapping.

Let op: je dient niets weg te schrijven in de database; daar is immers het BOS systeem voor bedoeld. Dat is wellicht vreemd, maar deze scheiding is kunstmatig aangebracht om je te laten oefenen met messaging en het verschil tussen topics en queues (zie de voorlaatste opgave) te laten zien.

Opgave 5 – messaging met FTP

Tot nu toe heb je bijna alleen maar gewerkt met standaardservices. Een andere, ietwat klassiekere, manier is het gebruik van FTP om een bericht te plaatsen in een inbox directory. Een apart systeem scant deze directory op berichten, verwerkt ze en plaatst de uitkomst in een outbox directory. Het systeem dat het oorspronkelijke bericht heeft geplaatst kan dan in de outbox directory het antwoord ophalen, wederom met FTP.

Je gaat in deze opgave via FTP het Optisch Leesbare Acceptgiro (OLA) systeem benaderen. Dit systeem produceert op basis van een tekstbestand in een verderop gespecificeerd formaat een pdf bestand van een acceptgiro.

Download het iText jar bestand en OLA.zip van Blackboard en pak het uit. Plaats acceptgiroeuro.jpg in een aparte directory en open het project. Je mist nog de verwijzing naar de iText library. Klik rechts op het OLA project, kies “Add Library”, “Create” en maak een nieuwe library entry voor iText aan. Voeg de library toe aan het project. Vervang de twee directories in de eerste regel van de `main` methode in `Sampler` door je eigen directories. Run Sampler.

De implementatie van AcceptGiro is bijna compleet, slechts het renderen van de onderste regel ontbreekt.

a. Implementeer de methode `printOLALine8`. Roep deze methode aan in de `paint` methode.

b. Maak een `AcceptGiroFileReader` klasse die een bestand in de inbox kan inlezen. Het formaat van het in te lezen bestand staat in box 1 hiernaast.

c. Maak een `AcceptGiroValidator` klasse die de validaties die in box 2 staan uitvoert. Het veld REFERENTIE komt weer terug als parameter in `createAcceptGiroPdf` en wordt de naam van het te genereren pdf bestand.

d. Maak een `AcceptGiroTransformer` die de volgende transformaties doet:

✓ Euro en Cent naar ints omzetten;

✓Betalingskenmerk groeperen in vier groepen van vier en gescheiden door een spatie;

✓Een naam opbouwen uit het geslacht, initialen en achternaam;

✓Een adres + pc opbouwen uit straatnaam, straatnummer en plaats.

```
REFERENCE=<<REFERENCE>>
BEDRAG=<<EURO>>,<<CENT>>
BETALINGSKENMERK=<<BETALINGKENMERK>>
REKENINGNUMMER=<<REKENINGNUMMER>>
GESLACHT=M | V
INIT=<<INITIALEN>>
ACHTERNAAM=<<NAAM>>
STRAATNAAM=<<NAAM>>
STRAATNUMMER=<<NUMMER>>
POSTCODE=<<POSTCODE>>
PLAATSNAAM=<<NAAM>>
REKENINGNUMMERNAAR=<<REKENINGNUMMERNAAR>
```

Box 1: het formaat van het in te lezen bestand.

e.Maak een `AcceptGiroFileRemover` klasse die een bestand in de inbox wist.

f.Maak een klasse `AcceptGiroProcessor` die in een configureerbare inbox directory kijkt voor nieuwe bestanden. Elk bestand wordt ingelezen (zie b)), gevalideerd (zie c)), getransformeerd (zie d)), gewist (zie e)) en via `AcceptGiro` als pdf in een outbox directory geplaatst.

g. Maak een `AcceptGiroRunner` klasse die via een `java.util.Timer` de `AcceptGiroProcessor` de handeling van vraag f) - kijken, verwerken en verwijderen – laat uitvoeren. De intervalperiode van de timer dient aanpasbaar te zijn.

h. Configureer een FTP server die de in- en outbox beschikbaar stelt.

i. Ga naar http://commons.apache.org/net/ voor een FTP client in Java. Schrijf in een nieuw project OLAClient een klasse die op basis van de gegeven velden in c) een bestand aanmaakt en in de inbox plaatst. Je hoeft niets op te halen met deze client Als een validatie mislukt dien je de verdere afhandeling af te breken.

j. Demonstreer de werking aan je practicumdocent.

```
<<EURO>> IN {1,...,99999}
<<CENT>> IN {0,...,99}
<<BETALINGSKENMERK>> IN CHAR(16)
<<REKENINGNUMMER>> IN CHAR(1, 10) of leeg
<<INITIALEN>> IN VARCHAR (5)
<<NAAM> IN VARCHAR(20)
<<NUMMER>> IN {1,9999}
<<POSTCODE> IN NUMBER(4) CHAR(2)
<<REKENINGNUMMERNAAR>> IN CHAR(1, 10)
```

Box 2: de uit te voeren validaties.

Opgave 6 – JMS Topics en Queues

De bedoeling van deze opgave is dat je leert werken met asynchrone communicatie via messaging. Dat doe je met JMS queues en topics. Lees de relevante hoofdstukken in The Java EE 6 Tutorial en probeer een aantal voorbeelden.

a. Wat is het essentiële verschil tussen een queue en een topic?

b. Maak via de admin console van Glassfish een JMS Queue (naam: jms/boetebepaling) en een JMS Topic (naam: jms/boeteafhandeling) aan.

c. Maak een MessageBean `BBSMessage` die luistert naar de boetebepaling queue. Deze bean gebruikt de session bean van opgave 4.

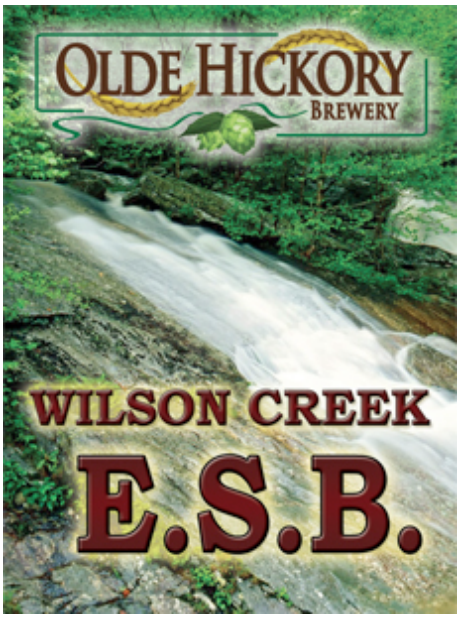
d. Maak een MessageBean `BOSMessage` die luistert naar de boeteafhandeling topic. Deze bean schrijft de boete weg in de boeteregistratiedatabase.

e. Maak een Messagebean `OLAMessage` die luistert naar de boeteafhandeling topic. Deze bean gebruikt de OLAClient van de vorige opgave.

Opgave 7 – Integratie en alternatieven

a. Integreer het CJIB proces tot één proces volgens het schema dat wordt genoemd aan het begin van week 2. Dit proces moet als webservice beschikbaar worden gesteld. Gebruik een JSON library om de GBA service aan te spreken

b. Bestudeer de dia's op http://tinyurl.com/cb8vj57 of bekijk de presentatie van Mark Richards op QCon 2006 (te vinden op http://tinyurl.com/4rvxgr). Leg aan de hand hiervan uit welke functies en rollen een ESB kan hebben.



c. Had de integratie van het CJIB proces die je in de vorige opgave hebt gemaakt ook met een ESB gedaan kunnen worden? Onderbouw je antwoord.

d. Leg uit wat BPEL is en wat de relatie is met ESB.