A feladatok általános követelményei

Architektúra

A feladatok tipikusan olyan egyszerű alkalmazás integrációk, amelyek a JBoss vagy Wildfly alkalmazás szerver, vagy docker segítségével megvalósíthatók.

Feladatok beadása

A feladatot a félév végén kell leadni, személyesen bemutatva. Lehet saját laptopon is vagy a labor gépein. Csak email-ben elküldött megoldásokat nem fogadunk el.

Beadási határidő

Az utolsó tanítási héttel bezárólag minden gyakorlaton. Ezután pótlás is lehetséges.

Feladatok

1.

Készítsen egy alkalmazást, amely 2 kliensből áll. Az első kliens a '/queue/colorQueue' üzenetsorra pont-pont csatlakozással véletlenszerűen RED, GREEN és BLUE paraméterrel ellátott üzeneteket küld 1 másodpercenként. Készítsen három MDB-t (üzenet vezérelt bean) amelyek filterrel a 'RED', 'GREEN' és a 'BLUE' paraméterrel ellátott üzeneteket kapják kizárólag. Minden 10 megkapott üzenet után az MDB-k a '/queue/colorStatistics' sorra küldenek egy üzenetet, ami azt jelzi, hogy 10 (adott színű) üzenetet feldolgoztak. Készítsen egy második klienst, ami a '/queue/colorStatistics' sorrol olvassa a statisztikát és a konzolba kiírja hogy pl. '10 'RED' messages has been processed'

2.

Készítsen egy alkalmazást, amely 3 kliensből áll. Az első kliens a '/queue/colorQueue' üzenetsorra pont-pont csatlakozással véletlenszerűen RED, GREEN és BLUE paraméterrel ellátott üzeneteket küld 1 másodpercenként. Készítsen három MDB-t amelyek filterrel a 'RED', 'GREEN' és a 'BLUE' paraméterrel ellátott üzeneteket kapják kizárólag. Az MDB-k véletlenszerűen átlagosan 10 ből 3 szor,

rollback-elik az üzenetet, ami így a halott levél csatornára kerül. Minden 10 sikeresen megkapott (nem rollback-elt) üzenet után az MDB-k a '/queue/colorStatistics' sorra küldenek egy üzenetet, ami azt jelzi, hogy 10 (adott színű) üzenetet feldolgoztak. Készítsen egy második klienst, ami a '/queue/colorStatistics' sorrol olvassa a statisztikát és a konzolba kiírja hogy pl. '10 'RED' messages has been processed'. A harmadik kliens a '/queue/DLQ' halott levél csatornáról a konzolon jelzi, ha egy üzenetet nem dolgoztak fel.

3.

Készítsen alkalmazást, amely az 1. feladat alapján működik, annyi különbséggel, hogy a kliens egy webszolgáltatás, amely SOAP-on keresztül küldi a véletlenszerű színeket a a Wildfly webszolgáltatásnak, és a webszolgáltatás küldi tovább '/queue/colorQueue' üzenetsorra pont-pont csatlakozással az üzeneteket. Az ez utáni teendők megegyeznek az 1.-es feladatban leírtakkal. A lényeges különbség az, hogy a kliens nem kapcsolódik közvetlenül az üzenetsorra, hanem a Wildfly-ban futó szolgáltatás küldi tovább az üzenetet a sorra.

4.

Készítsen alkalmazást, amely az 2. feladat alapján működik, annyi különbséggel, hogy a kliens egy webszolgáltatás, amely SOAP-on keresztül küldi a véletlenszerű színeket a a Wildfly webszolgáltatásnak, és a webszolgáltatás küldi tovább '/queue/colorQueue' üzenetsorra pont-pont csatlakozással az üzeneteket. Az ez utáni teendők megegyeznek az 2.-es feladatban leírtakkal. A lényeges különbség az, hogy a kliens nem kapcsolódik közvetlenül az üzenetsorra, hanem a Wildfly ban futó szolgáltatás küldi tovább az üzenetet a sorra.

5.

Készítsen alkalmazást amelynek egyik kliense a '/queue/colorQueue' üzenetsorra véletlen szöveges üzeneteket küld. ('RED', 'GREEN', 'BLUE') szöveggel 0.5 másodpercenként. Készítsen 3 további klienst amelyek folyamatosan 'hallgatják' a '/queue/colorQueue' üzenetsort, ha valamelyik üzenetet olvas a sorról, akkor véletlenszerűen (2000-9000) milliszekundum ideig várakozik, majd utána újra hallgatja az üzenetsort. (A véletlenszerű feldolgozási idő miatt -e közben egy másik üzenetet a következő kliens fogja feldolgozni). A 3 kliensnek ne legyen konzol outputja, azaz ne írjon ki semmit a konzolra. Minden üzenet feldolgozása után a '/queue/processedColors' sorra küldjenek egy üzenetet, ami azt jelzi, hogy feldolgozták a feladatot. Készítsen egy klienst, ami a '/queue/processedColors' sort olvassa és a megjelenő üzeneteket kiírja az outputra. pl: 'Client 2, processed a 'BLUE' message'. Ennél a feladatnál nem kell a Wildfly-ban komponenst létrehozni.

6.

Készítsen egy képzeletbeli prepaid mobilfeltöltő alkalmazást amely egy tcp alapú socket szerver lesz egy adott porton fogadja a kliensen kéréseit. A kliensek atm automaták, amelyek a következő protokoll szerint működnek: 1.) a kliens egy tesztüzenetben elküldi a feltölteni kívánt telefonszámot, 2.) ha a szám feltölthető, akkor OK egyébként ERROR üzenettel megszakítja a tranzakciót. OK üzenet esetén egy tranzakció azonosítót kapunk, amely a következő 3. lépésben azonosítja a folyamatot. 3.) a kliens a tényleges feltöltést kezdeményezi a korábban kapott tranzakció azonosítóval, de újra el kell küldeni a telefonszámot és a feltöltési összeget.

A lehetséges feltöltési összegek: 3000,5000,10000,15000 Készítsen egy lekérdező klienst, amely egy adott telefonszámhoz tartozó korábbi tranzakciókat listázza. Használjon adatbázist a tranzakciók tárolására a szerver alkalmazásban.

Indítson 3 klienst, amelyek egy előre adott telefonszám halmazt használnak és véletlenszerűen teszt és feltöltés tranzakciókat indítanak.

7.

Készítsen egy kliens alkalmazást, ami http POST kérésekkel szöveges adatokat küld egy üzenetsor vezérelt mintarendszerbe. Úgy tervezze meg az alaprendszert, hogy később nemcsak szöveges, hanem más, pl. bináris állományokat is tudjon kezelni. A mintarendszer alapfeladata, hogy megszámolja a szöveges állományokban a szavak számát és eredményként visszadja a kliensnek. Készítsen egy futtató környezetet amiben több kliens párhuzamosan, véletlenszerű állományokkal használja a szolgáltatást. Készítsen legalább 3 mintaszöveget a teszteléshez. Tételezzük fel, hogy a működő rendszert ki kell egészíteni egy új funkcióval: mostmár bmp formátumú képeket is lehet küldeni, amelyeknél a rendszer a kép méretét adja vissza ereményként. Készítse el úgy a rendszert, hogy menet közben lehessen kicserélni a kliens és szerver komponenseket, azaz nem lehet leállítani a rendszert. A megoldáshoz használja az itt bemutatott elveket: http://ait2.iit.uni-miskolc.hu/oktatas/doku.php?id=tanszek:oktatas:informacios_rendszerek_integralasa:oesszetett_pelda_1

8.

Adott egy A és egy B szerverkomponens. A B komponens hozzáfér egy SQL adatbázishoz, amiben 1 db tábla van, ami személyek adatait tartalmazza (név, szül idő, stb..). Az 'A' komponensnek nincs hozzáférése az adatbázishoz. Az 'A' komponens rendelkezik viszont egy browser-ben megjelenő felülettel, ahol személyek adatait lehet lekérdezni (összes személyt egyszerre) és új személyt lehet felvenni. A 'B' komponens két funkciója: egy új személy felvétele és a személyek lekérdezése az adatbázisból. Készítse el az A és B komponenseket és hozzon létre kapcsolatot közöttük SOAP felhasználásával. Az A és B komponens és az adatbázis tetszőleges technológia lehet.

9.

Adott egy A és egy B szerverkomponens. A B komponens hozzáfér egy NO-SQL adatbázishoz, amiben 1 db tábla van, ami könyvek adatait tartalmazza (cím, szerzők, kiadó, év, stb..). Az 'A' komponensnek nincs hozzáférése az adatbázishoz. Az 'A' komponens rendelkezik viszont egy browser-ben megjelenő felülettel, ahol könyvek adatait lehet lekérdezni (összes könyvet egyszerre) és új könyvet is fel lehet venni. A 'B' komponens két funkciója: egy új könyv felvétele és a könyvek lekérdezése az adatbázisból. Készítse el az A és B komponenseket és hozzon létre kapcsolatot közöttük JAX/RS felhasználásával. Az A és B komponens tetszőleges technológia lehet.