Innlevering 2 - PG4100

av Eivind Vegsundvåg

1. Innledning

Løsningen leveres som en Maven parent/reactor med tre moduler:

- commons, som inneholder modeller og et enkelt Quiz annotations-API.
- client, som inneholder en DataFX-applikasjon med to kontrollere
- server, som inneholder en Spring Boot-applikasjon med:
    - Hibernate ORM
    - Spring Data JPA

Både klienten og tjeneren bruker Netty som Socket-rammeverk. Løsningen går derfor noe rundt noen av problemstillingene til innleveringen, og erstatter dem med helt andre. Eksempelvis har jeg ikke hatt noen behov for å selv sette opp tråder for klienttilkoplinger; dette følger med Netty sine Bootstraps. Databasetilkoplingen er også delvis abstrahert bort; dette ordnes i Spring Boot sine konfigurasjonsklasser og instansieres bare delvis manuelt. Hele resten av databaselaget er instansiert ved hjelp av Spring “proxies”. Disse er Aspects som sørger for korrekt transaksjonsbehandling ved å sette inn kode “før” og “etter” metodekall.

Denne overkompliseringen har resultert i en ganske reell applikasjon; det er problemfritt å slenge opp et Web-API for å legge inn ekstra bøker; dog vil dette medføre at applikasjonen ikke kunne blitt kjørt i IDE, men oppstart ved hjelp av Maven. Dette ville vært uønsket, men jeg påpeker at dette bokstavelig talt er innsetting av tre annotations. Se gjerne punkt for tenkte nye funksjoner.

2. Forutsetninger

Dokumentasjonen for dette prosjektet er mangelfull i forhold til hvor mye som kunne vært forklart. Dette skyldes de mange avhengighetene til prosjektet, og det forutsettes derfor at leser har en viss forståelse av komponentene som brukes. Det er vedlagt kilder som gir en enkel innføring i dem. Alle klasser er i tillegg kommentert i JavaDoc-format for å redusere effekten av dette.

Prosjektet brukes avhengigheter fra Maven i veldig stor grad. Det forutsettes derfor at bruker enten har Maven installert, og/eller bruker et utviklingsmiljø som integrerer Maven.

Serveren har ingen SQL-filer; tabellstruktur og startdata ordnes automatisk ved hjelp av Java-objekter.

3. Uløste problemstillinger

3.1 Netty

Dette rammeverket har, til tross for at det tillatter rask produksjon av kraftige tilkoplinger, noen svakheter:

3.1.1 java.io compliance

Netty støtter ikke ObjectOutputStream eller lignende. Med mindre man skriver all dataen ved hjelp av utstrømmer av like lavt nivå som Netty selv er bygget på, støtter Netty-servere kun Netty-klienter. Dette har også bidratt til at de to viktigste metodene(channelRead()) i applikasjonen bare er tested rundt, ikke på.

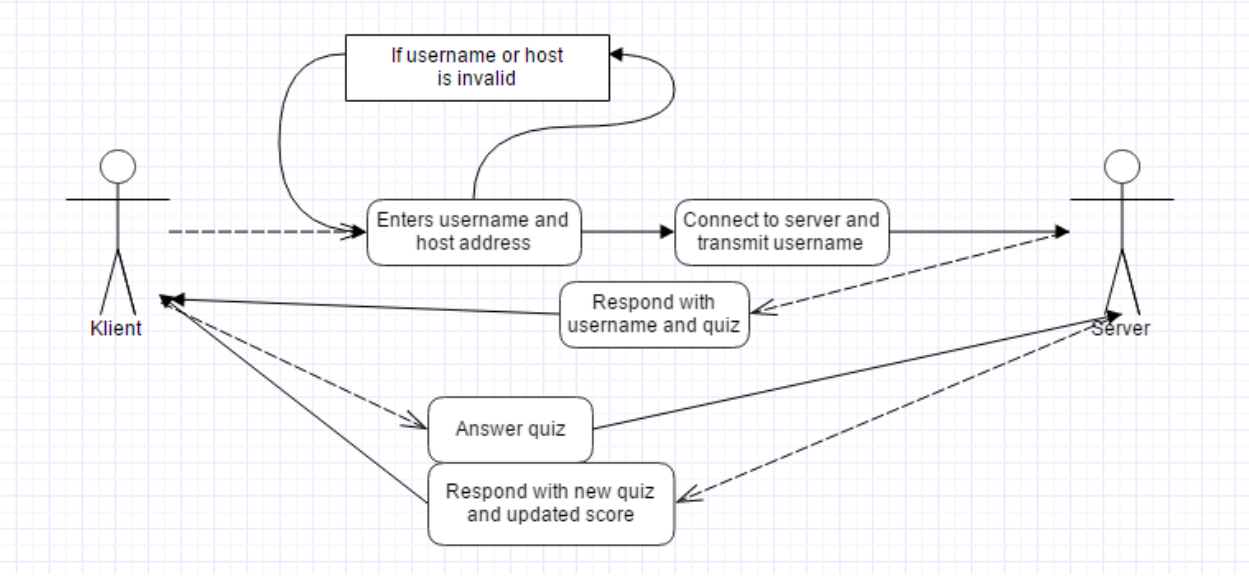
3.1.2 Inversion of Control

Spring Framework er basert på Inversion of Control; konseptet om at komponenter ikke trenger å vite hvordan man iverksetter en annen komponent de er avhengig av, bare hvordan man bruker dem. Forskjellige Netty-komponenter er dog vanskelig å starte opp på en slik måte, så dette gjøres ved å hente ut alle komponentene fra Spring for å iverksette dem i egne handlers.

3.2 JavaFX

Dette grensesnittrammeverket er veldig strengt når det gjelder hvilke tråder som aksesserer sine komponenter. Dette har resultert i at testene ofte må legge inn pauser etter å spesifikt ha bedt JavaFX om å ha utført operasjoner på sin egen tråd for å sikre at operasjonen er ferdig.

4. Programflyt



Jeg så for meg følgende flyt når jeg utviklet applikasjonen. For å unngå bruk av søkemotorer, ble det i senere tid lagt inn at klienten kun har 20 sekunder på å svare, før innholdet i tekstfeltet automatisk blir sendt.

5. Modulene

5.1 Klienten

Klienten bruker Netty IO og DataFX til å sette opp en enkel og konsis MVC-applikasjon. Testingen er især interessant; den testes i en hybrid av integrasjonstester og enhetstester, ved hjelp av TestFX, JUnit og Mockito.

5.1.1 TestFX

TestFX tilbyr muligheten for å kunne "klikke" komponenter, eller hente dem frem som ved et DOM-hierarki. Dette tillatter ellers vanskelig testing å utføres på en veldig enkel måte, og lar en simulere oppførelsen. Dessverre kunne ikke hele løsningen testes. Se punkt for problemstillinger rundt Netty.

5.2 Tjeneren

Tjeneren bruker Spring Boot til å omslutte resten av komponentene i et "Context Aware" miljø som tillatter enkel oppkobling av store mengder komponenter, ved å registrere dem i en SpringContext. Dette tillatter spennende og enkel integrasjonstesting ved hjelp av profiler.

Denne modulen bruker også Spring Data JPA for å generere et semantisk miljø for databasen, hvor metodenavn i Repository-klasser avgjør oppførselen til grensesnittet.

5.2.1 Profiler

Profilene "dev" og "test" starter applikasjonen med en H2-database i minnet, mens "prod" forsøker å koble til en database konfigurert i src/main/resources/application.properties. Profil kan settes ved å starte applikasjonen med ønsket profil som argument.

5.2.2 Tester

Løsningen har en selvvaliderende integrasjonstest; Spring kaster Exceptions hvis ikke alle modulene fungerer som de skal, og dette avslutter testen som feilende ved oppsett. Det er ikke fokusert på enhetstester for server, og kun klassene ClientInitializer og QuizClientHandler er testet med enhetstester. Dette skyldes at modulen commons inneholder de klassen som typisk sett ville vært testet med enhetstester.

5.3 Fellesbiblioteket

Commons-modulen inneholder dataobjekter det er typisk å sende gjennom sockets. Den inneholder dessuten også et Annotation-API for Quiz i form av annotasjonene QuizField og Quizzable. Disse fungerer ved å annotere elementer i for eksempel Book-klassen slik at klassen QuizGenerator kan generere Quiz-objekter med spørsmål og svar.

6. Tenkte nye funksjoner

Ettersom løsningen er omfattende, er funksjoner kun implementert i forhold til hva som ble ansett som minimumskrav. Derfor kunne følgende funksjoner blitt implementert for å gjøre applikasjonen fullverdig for produksjon:

- Passordbeskyttede brukere
- Flere klasser for Quiz, med valg for klient om hvilke
- Mer informasjon om spillere, som f.eks poengsum per sesjon, high score per sesjon, høyeste antall riktige svar på rad.
- Web-grensesnitt for innlegging av Quiz-kompatible modeller på server

7. Kilder utenfor pensum

Flere av disse kildene er brukt i veldig stor grad. Det er dog mye å sette seg inn i, og flere konkrete eksempler er funnet andre steder.

http://projects.spring.io/spring-data-jpa/
http://projects.spring.io/spring-boot/
http://netty.io/wiki/user-guide-for-4.x.html
http://www.javafxdata.org/
https://github.com/TestFX/TestFX