CDIO del_3

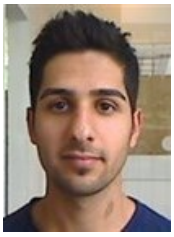
02324 Videregående programmering

4. Maj 2018


Mathias
Fager
s175182


Niklaes
Jacobsen
s170423


Sebastian
Stokkebro
s160198


Alan
Jafi
s122417


Burhan
Shafiq
s175446

Indhold

Tidsforbrug

Herunder ses vores tidsforbrug på projektet, opdelt pr. pers. angivet i hele timeantal

Navn	**Analyse**	**Design**	**Implementering**	**Projektplan**	**Totalt**
Alan	5	10	60	5	80
Burhan	5	10	60	5	80
Mathias	5	10	60	5	80
Niklaes	5	10	90	5	110
Sebastian	5	10	80	5	100

1 Indledning

I dette projekt vil vi beskæftige os med et system, som kan håndtere en brugers data mellem et front-end web interface og et back-end system. Back-end systemet lagrer, og ændrer i en ekstern database.

Back-end delen er REST-baseret og ved hjælp af de basiske HTTP metoder GET, POST, PUT, DELETE kan vi følge CRUD-modellen (Create, Read, Update, Delete) til at modificere dataen på serveren. Til vores projekt har vi gjort brug af GET, POST og DELETE.

Front-end delen består af en dynamisk single page application, skrevet i html/css/javascript, som genskriver den samme side når brugeren interagerer med den, fremfor at der skal loades en helt ny side fra serveren.

2 Analyse

2.1 Kravspecifikation

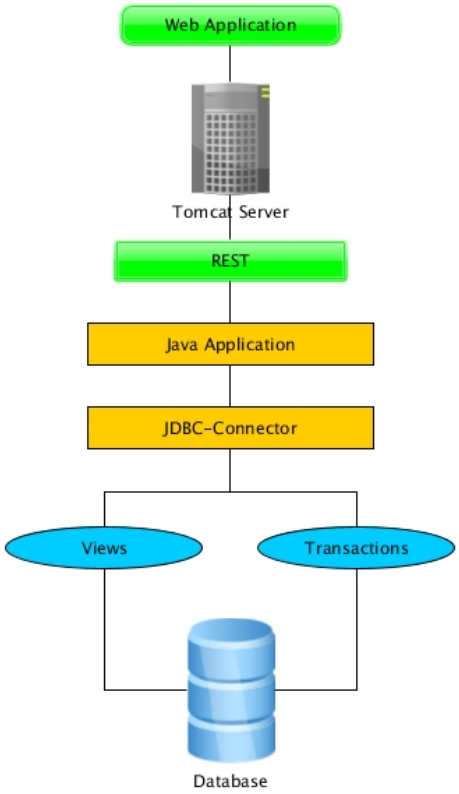
- HTML/CSS hjemmeside med funktionelt JavaScript.
- Hjemmesiden er en single page application.
- Hvis muligt, always online for både hjemmeside og database, med en lokal database der opdateres hver gang der forbindes til den online database.
- Oprette brugere med forskellige roller, eventuelt flere roller per bruger.
- Admin skal kunne tilføje følgende roller: Admin, Pharmacist, Produktionsleder, Laborant.
- Back-end systemet, skal opsættes med en REST.
- Data lagres i en persistent database mellem sessioner.
- Imellem REST og JavaScript, skal der bruges til/fra REST-backenden er serialiseret på en måde (eks. JSON) så andre applikationer kan anvende servicen.
- Datalaget ønskes afkoblet med et interface IUserDAO.
- IUserDAO, der indeholder definition/erklæring af de nødvendige metoder.

3 Design

3.1 Overvejelser af design

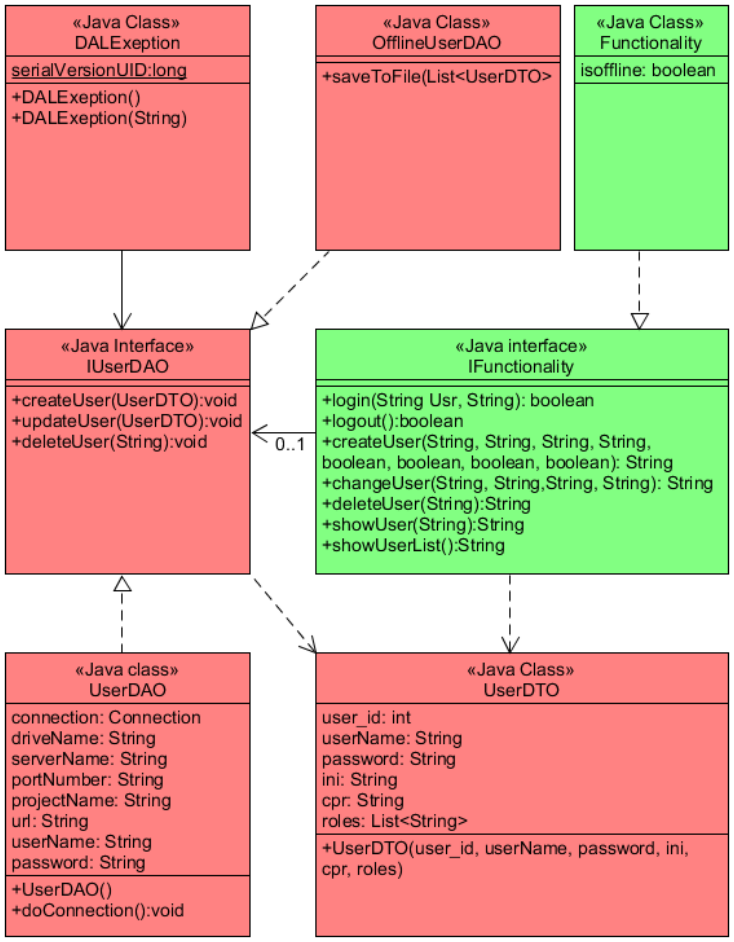
Vi vidste allerede fra start af projektet at det skulle sættes op efter 3-lags modellen. Denne beslutning skyldes at projektet ligger meget fint op til denne form for arkitektur.

Brugergrænseflade kommer til at bestå af en hjemmeside. Her håndterer vi alle bruger-input og sender det videre til funktionalitetslaget. I funktionalitetslaget er der java-laget, der vil stå for at håndtere input fra brugergrænsefladen, og vil derfra udføre relevante metoder.
Udover dette, skal programmet også kunne interagere med en database og håndtere at sende samt modtage data fra den.



Figur 1: Visuelt billede af hvordan systemet er opsat

3.2 Klassediagram



Figur 2: Klassediagram af JAVA-laget

Som det kan ses på billedet, er de røde klasser fra DATA pakken og de grønne tilhører funktionalitets pakken. Vi kan se, hvor mange instanser der kommer til at forekomme af de forskellige klasser. Det kan ses på vores data klasser at de udelukkende kommer til at stå for at modtage og interagere med dataen fra vores SQL database og skabe forbindelse til vores server. Alt interaktion med hjemmesiden, og de forskellige funktioner der vil skal kunne benyttes på siden, vil blive håndteret af vores funktionalitets klasse. Dette er for at skabe en høj kohæssion iblandt vores programs klasser.

4 Implementering

Vores implementeringsovervejelser har ligget i, hvordan man nemt kan skabe en forbindelse mellem programmets tre lag. Vi starter fra bunden af vores trelagsmodel, netop databasen. For at forbinde databasen med java (funktionalitetslag) har vi benyttet os af en JDBC (Java DataBase Connectivity). Ved hjælp af JDBC'en kan vi netop sende query kommandoer direkte til databasen fra vores java program. Men før vores java program kan begynde at spørge databasen om informationer, skal den først bruge noget input fra vores hjemmeside (Brugergrænsefladen), derfor skal der også være en forbindelse mellem programmet og hjemmesiden. Dette har vi fået gjort ved hjælp af javascript og REST, der hjælper os med at få kommunikationen mellem programmet og hjemmeisden op at køre.

5 Dokumentation

I vores kode har vi valgt at bruge javaDoc til at dokumentere vores kode. De metoder der har oprindelse fra et interface, laver vi javaDoc på metoderne i interfacet i stedet. Med javaDoc, kan man generere en hjemmeside, som beskriver koden.

6 Projektplanlægning

For at kunne udnytte tiden korrekt, har vi brugt online-værktøjet 'Asana'. På den måde har vi kunnet udnytte tiden korrekt. Vi satte delmål og deadlines på for hver del af projektetet. Alt gik efter planen og vi blev færdige på det tidspunkt, som vi havde forudsagt.

7 Konklusion

Ud fra dette projekt kan vi hermed konkludere, at vi har et velfungerende program, der virker som det skal. Det har lykkes os at få skabt en forbindelse mellem alle programmets lag, ved hjælp af de nødvendige værktøjer. Vi har dog nogle få forbedringer som vi gerne vil implementerer i det næste projekt. Såsom anvendelsen af transaktioner i vores SQL database, hvilket ville gøre det muligt at lave roll-backs i tilfælde af noget skulle gå galt. Derudover vil en udtrækning af forbindelsen af SQL-forbindelsen, forbedre det store forbrug af connector objekter, som vi laver lige nu.