Danmarks Tekniske Universitet

CDIO 1

Diplom Software

23. februar 2018

Gruppe 22

Gruppemedlemmer:

Thomas Mattsson - s175206
Rasmus Gregersen - s175221
Patrick Marcell Madsen - s175209
Søren Kaare Rasmussen - s175202
Dan Sørensen Drachmann- s155384

1. Indledning

1.1 Problemformulering/Case:[1]

Vi har fået et nyt projekt, hvor skal udvikle et afvejningssystem. Til start skal vi få brugeren til at få adgang til programmet. Her skal brugeren kunne oprette sig selv, vise en liste over de registrerede, opdatere en bruger, slette en bruger og afslutte programmet (dette illustreres senere i vores use case diagram).
De registrerede brugere skal kunne gemmes inde i en disk/database, som programmet skal tilgå og hente data fra.

1.2 Formål, Værktøjer og Metoder:

Formålet ved denne opgave er at styrke vores kompetencer i 3-lags modellen, som er en central del af et system som dette. Derudover har opgaven også til formål at introducere os til, hvordan man gemmer data i en database som MySQL. Da vi senere i 3 ugers projektet skal lave et afvejningssystem, ville koden fra denne opgave kunne genbruges til håndtering af de brugere som systemet har.

I analysefasen tydeliggøre vi kundens vision i form af en kravspecifiktion, hvor vi derefter anvender UML artefakterne: use case diagram og en use case beskrivelse, der er med til at formidle kundens krav.
Nede i designfasen benytter vi os af designklassediagram og pakkediagram fra UML, for at sikre bæredygtighed og genanvendelse i vores arkitektur ved brug af 3-lags modellen. Senere kommer vores implementeringer af selve projektet.

Vores værktøjer:

- Eclipse IDE
- GitHub
- Google Docs

Vores teknologier:

- Java
- MySQL

[1] (Taget fra Kundens Vision CDIO_D1)

2. Analyse

2.1 Kravspecifikation

Kunden ønsker et afvejningssystem, som kan bruges af hendes/hans ansatte. Ud fra kundens vision har vi så udviklet kravspecifikationer til programmet i første fase og vil videreudvikle kravene og funktionalitet yderligere i de næste faser.

Nedenstående kravspecifikationer til vores program:

Funktionelle krav

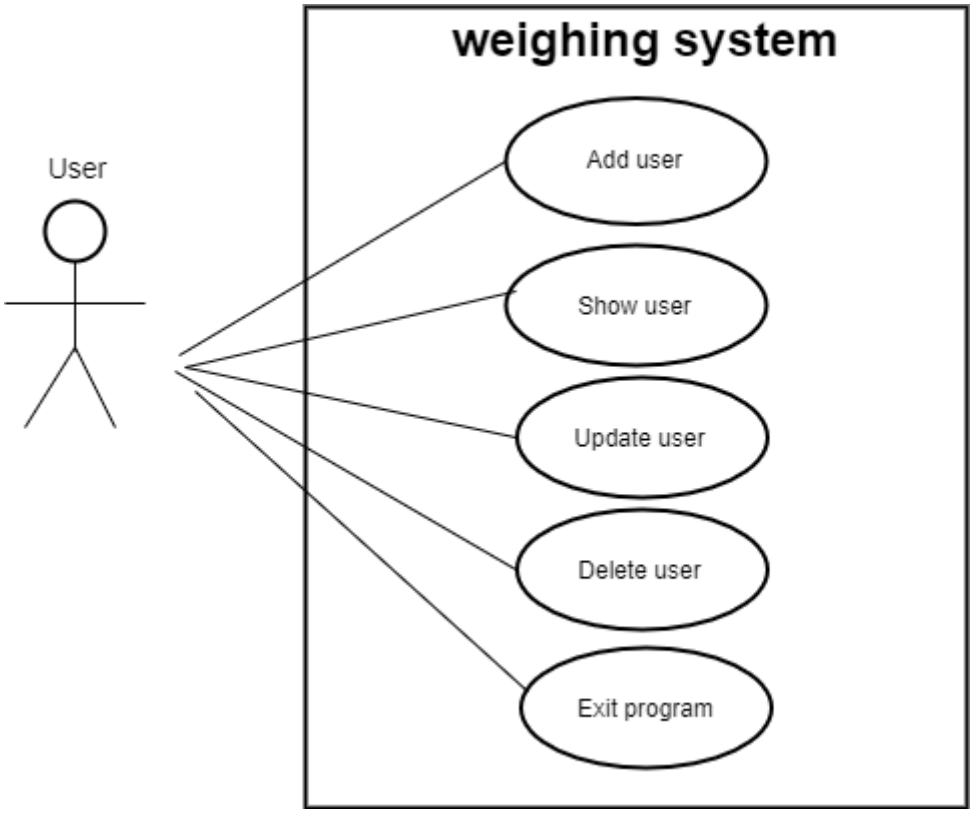
1. Det er nødvendigt at programmet kan **oprette** flere brugere.
2. Programmet skal kunne vise kunne **vise** alle registrerede brugere.
3. Der skal kunne **slettes** brugere.
4. Man skal kunne **ændre/rette** oplysninger på brugerne.
5. Brugeren skal kunne **afslutte** programmet uden besværligheder.

Ikke-funktionelle krav

6. Programmet giver besked til brugeren, hvilke input det skal for at kører den.
7. Programmet skal køre uden bemærkelsesværdige forsinkelser[2].
8. Programmet skal fungere på DTU's databarer.
9. Programmet skal have en koldstart på under 3 sekunder og en varmstart på under 2 sekunder.

[2] En bemærkelsesværdig forsinkelse er alt over 300 millisekunder.

2.2 Use case diagram



Figur 2.1: Use case diagram af afvejningssystemet

Ovenfor ses vores use case diagram af afvejningssystemet, som indeholder de use cases, systemet skal understøtte. Vores use case opfylder de funktionelle kravspecifikationer vi har tillagt os, som brugeren skal kunne anvende i systemet. *Add user* gør det muligt at registrere brugerne ind i systemet, mens *show user* viser alle informationer om brugeren bortset fra adgangskoden. *Update user* gør det muligt for en bruger at opdatere sine oplysninger og *delete user* fjerner så en registreret bruger. Til sidst skal brugeren også, kunne afslutte programmet, som er *exit program*.

2.3 Use case beskrivelse - Fully dressed:

Use Case Sections	Comments
Use Case Navn	Add user
Scope	Weighing system
Niveau	User-goal
Primær Aktør	User
Interessenter	Ingen

Forudsætning	Programmet er installeret og klar til brug.
Success garanti	Programmet har registreret en bruger til disken/databasen.
Primære Scenarie	**1.** Brugeren skal indtaste userName, initialer, CPR-nummer og roller. **2.** Systemet genererer en UserID ud fra databasens plads. **3.** Systemet genererer et password til brugeren **4.** Alle data af brugeren bliver derefter gemt inde i databasen (MySQL)
Alternativ flow	***a.** Hvilken som helst tid, hvis systemet crasher: **1.** Brugeren genstarter systemet. **2.** Brugeren følger derefter det primære scenarie 1-4.
Specielle krav	Programmet giver besked til brugeren, hvilke input der skal til for at kører den. Programmet skal køre uden bemærkelsesværdige forsinkelser. Programmet skal fungere på DTU's databarer. Programmet skal have en koldstart på under 3 sekunder og en varmstart på under 2 sekunder.
Frekvens	Indtil programmet afsluttes
Andet	Intet

Figur 2.2: Fully dressed use case

3. Design

3.1 Designklassediagram:

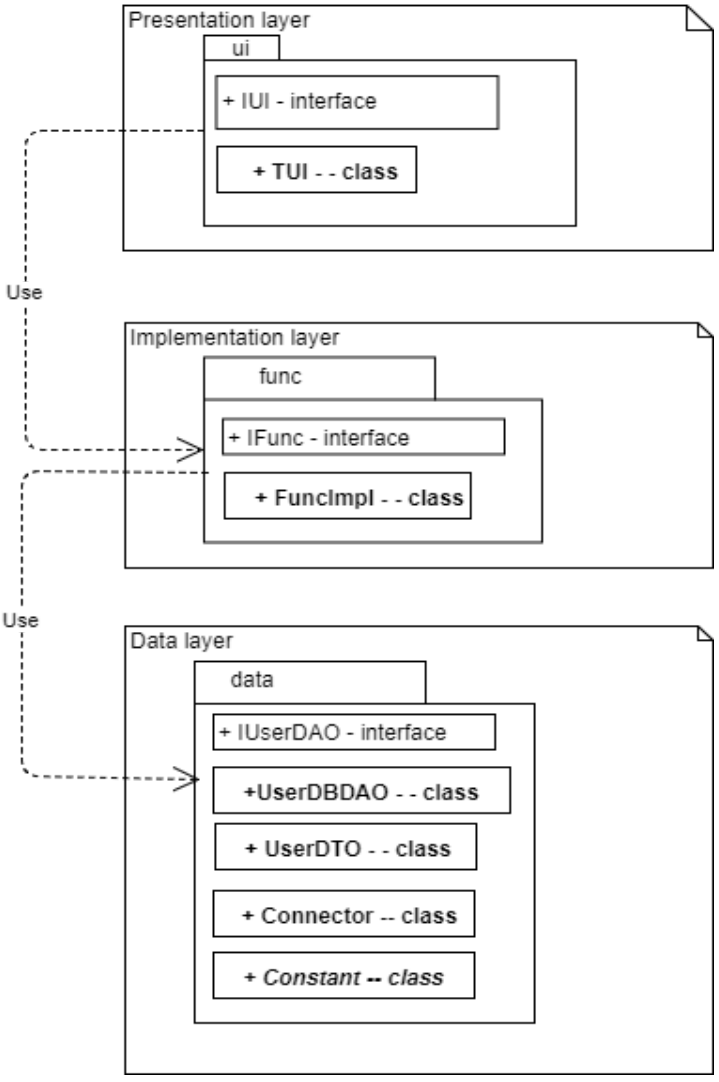
Figur 3.1: Designklassediagram over programmet

Figuren ovenover viser designklassediagrammet af systemet.

Diagrammet viser at tre af vores klasser implementerer hver deres interface og hvilke klasser der bruger hinanden.
Klasserne bruger hinanden ud fra tre-lags modellens principper.

3.2 Pakkediagram:

Da vi arbejder ud fra trelagsmodellen, hvor grænsefladen er det lag som brugeren interagerer med direkte og derefter tager vi brugerens input fra grænsefladen til funktionslaget for arbejde med brugerens input. Under funktions-laget ligger datalaget, som skal opbevare vores data.



Figur 3.2: Pakkediagram over systemet

Ovenover er illustreret en pakkediagram over vores program. Øverst ligger grænsefladen/præsentationslaget, som indeholder en TUI med interface IUI. Grænsefladen bruger funktionslaget til at håndtere de data TUI'en får fra brugeren. Inde i funktionslaget ligger FuncImpl med interface IFunc, som håndterer inputs og kan oprette brugeren som et objekt. Dernæst giver funktionslaget data'en videre til datalaget, som oplagrer oplysningerne, hvor så funktionslaget så kan tilgå de oplysninger igen.

4. Implementering

Dette afsnit beskriver centrale metoder og klasser, som er særligt vigtige.

4.1 TUI

Denne klasse fungerer som vores grænseflade. Klassen fungerer som et tekst interface, hvor brugeren får instrukser om hvilke operationer han/hun kan foretage sig. Klassen fungerer ved at have en switch case, som tager brugerens input som parameter og derefter kalder de korrekte metoder i funktionslaget afhængigt af hvilket input brugeren kommer med. På billedet nedenunder ses et udsnit af sådan en switch-case.

```
public void run() throws DALException { //Displays welcome screen
    System.out.println("Velkommen! Tryk på det tal, der svarer til det menupunkt du gerne vil tilgå");
    System.out.println("1: Find bruger");            //Get User
    System.out.println("2: Udskriv alle brugere");   //Get UserList
    System.out.println("3: Opret bruger");           //Create user
    System.out.println("4: Rediger bruger");         //Update user
    System.out.println("5: Slet bruger");            //Delete user
    System.out.println("6: Afslut program");         //Exit program
    Scanner sc = new Scanner(System.in);

    boolean hasEnded = false;
    while (!hasEnded) {
        int selection = sc.nextInt();
        switch (selection) {
        case 1: //getUser
            System.out.println("Indtast bruger ID'et på den bruger du gerne vil finde: ");
            int userId = sc.nextInt();
            f.getUser(userId);
            System.out.println("1"); //Only for test purposes
            break;
        case 2: //GetUserList
            System.out.println("Printer liste...");
            f.getUserList();
            System.out.println("2"); //Only for test purposes
            break;
```

Hele TUI-en er inde i et while-loop, som brugeren kan bryde ud af, og dermed slukke for programmet.

4.2 Funcimpl

Denne klasse fungerer som vores funktionslag. Klassen metoder bruges til at CRUD brugere i databasen/systemet.

Klassen implementerer interfacet IFuncimpl som specificerer fundamentet af klassen. Funktioner som getUser og getUserList fungerer ved at kalde tilsvarende metoder i datalaget, specifikt i en klasse som implementerer interfacet IUserDAO.

createUser er en af de mere interessante metoder i klassen, den tager de informationer som en bruger skal indeholde, på nær password, som klassen selv skal oprette, som følger DTUs password specifikationer og opretter derefter et UserDTO objekt som alt denne data sættes ind i. UserDTO objektet sendes nu videre til datalaget ved at kalde en metode også kaldt createUser i IUserDao som tager et UserDTO objekt som parameter opretter derefter en bruger i databasen med objektets informationer.

4.3 UserDBDAO

Denne klasse er en stor del af vores datalag, da det er her, hvor vi sender vores oplysninger til databasen samt kommer med queries for dem. Denne klasse er dén klasse i vores projekt, som sender vores SQL sætninger til vores database samt returnerer den data, som man lavede sit query for. Eksempelvis så returnerer metoden getUser i denne klasse den bruger, som blev fundet efter at SQL sætningen er blevet executed med det userId som getUser har som parameter.

5. Konklusion

Vi har anvendt en extreme programming som arbejdspraksis, hvor vi har gjort meget ud af par programmering og designing af vores arkitektur i systemet. Vi havde især sat stor fokus på, hvordan vi ville kunne anvende tre-lagsmodellen ud fra kundens vision og derefter få implementeret en MySQL database, der gør det muligt at oplagrer data, som senere kan tilgås.
Hele arbejdsgangen har været succesfuld og har ført til vores færdige produkt. Vi kan dernæst videreudvikle projektet i CDIO 2, så vi nærmer os kundens fuldendte afvejningssystem.
Grunden til at vi ikke har test, er fordi vi har lagt mere vægt på at få arkitekturen og databasen i orden, før vi videreudvikler projektet. Til næste CDIO opgave kan vi så arbejde mere med testmetoder.

5.1 Produkt:

Vores endelige produkt endte med at være et vellykket program, som opfylder alle vores funktionelle kravspecifikationer, som vi har tillagt os i vores første fase. Vi har prioriteret design og implementering af disk/database så produktet bliver bæredygtigt og nemt at arbejde med. Produktet finpudses videre med senere hen i forløbet til at opnå det ønskede produkt