Forår 2016

02324 – Videregående programmering

Gruppe nr.: 19

Afleveringsfrist: Lørdag den 07/05-2016 kl. 05:00

[PROJEKTOPGAVE – CDIO3]

Silas El-Azm Stryhn
s143599

Ramyar Hassani
s143242

Mikkel Hansen
s143603

Frank Thomsen
s124206

Martin Rødgaard
s143043

Denne rapport er afleveret via Campusnet

Denne rapport indeholder 18 sider inkl. denne side.

Timeregnskab

Timeregnskab (*hele antal timer*)					
	Silas	**Martin**	**Mikkel**	**Frank**	**Ramyar**
Analyse	**1**	**3**	**1**	**6**	**1**
Design	**5**	**1**	**2**	**3**	**3**
Implementering	**15**	**8**	**25**	**8**	**5**
Test	**0**	**0**	**3**	**3**	**1**
Dokumentation	**2**	**8**	**2**	**4**	**4**
I alt	**23**	**20**	**33**	**24**	**15**

Indholdsfortegnelse

Timeregnskab 1
Resumé 3
Indledning 3
Hovedafsnit 4
- Analyse 4
  - Kravspecifikation 4
  - Use Case Diagram 6
  - Use-case beskrivelser 7
- Design 10
  - Design-klassediagram 10
- Implementering 13
  - Manglende implementation 13
- Test 14

Konklusion 15
Bilag 16
- Git 16
- Databasen 16
- Systemkrav 17

Resumé

Vores rapport indeholder et analyseafsnit, hvor kravene til systemet bliver analyseret og inddelt i hvor vigtige de er. Dernæst har vi lavet et Use Case diagram, som viser hvordan vi havde tænkt os at vores system skulle fungere, de forskellige Use Cases er beskrevet i Use Case beskrivelserne. I vores design afsnit har vi lavet et klasse diagram, som viser hvordan vores system er bygget op og fungere sammen. Så har vi forklaret hvad vi har implementeret og hvordan vi har implementeret det i implementations afsnittet, og til sidst testet vores implementation i test afsnittet. I bilag, har vi vist hvordan man henter vores projekt, samt hvordan man sætter databasen op, så den fungere sammen med vores system.

Indledning

Denne rapport præsentere en gemmengang af udviklingen af et Web interface med back-end til operatør administration. Dette skal laves som i CDIO1, med at der er en superbruger(administator) som kan oprette andre operatører. Disse operatører skal så kunne ændre deres password, mens superbrugeren skal kunne opdatere de andre operatører i systemet, samt kunne deaktivere/aktivere dem.

Rapportens basale formål er at give en grundlæggende forståelse for tankegangen igennem forløbet, hvilke overvejelser der er gjort og hvordan vores implementation er stykket sammen. Rapporten gennemgår således en analyse af kravene og en udarbejdelse af diverse diagrammer, som skal give indblik i systemets opbygning.

Hovedafsnit

Analyse

I dette afsnit har vi analyseret opgaven, inden vi går i gang med implementeringen, for at få et overblik over hvilke krav der er til opgaven og vise vores tanker og overvejelser. Vi laver her en kravspecifikation, hvor vi viser hvor meget vægt vi lægger på diverse krav.

Kravspecifikation

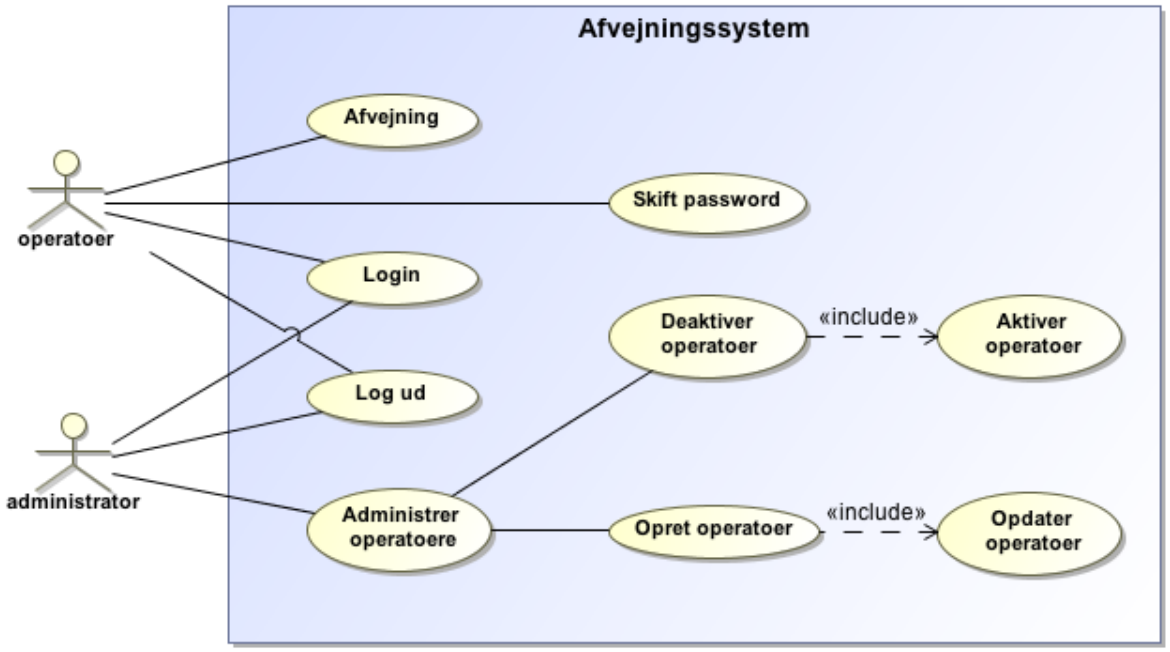
Vi skal lave et web interface med back-end til operatør administration. Ud fra opgaveformuleringen har vi udarbejdet en kravspecifikation. Derefter har vi inddelt hver enkelt krav i funktionelle og ikke funktionelle krav for derefter at bruge MoSCoW metoden på dem og kommentere vores valg.

Programmet skal være webbaseret.	Ikke funktionelt	Should have	Kunden ønsker at det skal være web-baseret og derfor er dette krav vigtigt, men det er ikke et funktionelt krav.
Webapplikationen skal skrives i GWT.	Ikke funktionelt	Could have	Vi synes det er mindre vigtigt, hvordan applikationen er lavet. Men vi vil selvfølgelig overholde kundes krav.
Det skal være muligt for en superbruger at kunne administrere operatørerne.	Funktionelt	Must have	Da dette er applikationens formål er dette krav selvfølgelig utrolig vigtigt.
Superbruger skal have mulighed for at oprette en operatør.	Ikke funktionelt	Must have	Da superbrugeren skal kunne administrere operatørerne er det også vigtigt han kan oprette nogle.
Brugeren skal have mulighed for at slette en operatør.	Ikke funktionelt	Should have	Vi synes ikke det er hensigtsmæssigt at kunne slette data. Derfor blev vi enige med kunden om at implementere en aktiver / deaktiver funktion i stedet.

Superbruger skal have mulighed for at rette en operatør.	Ikke funktionelt	Should have	Vi synes dette krav er vigtigt da det er en stor del af at kunne administrere.
En operatør skal bestå af et operatør id.	Ikke funktionelt	Must have	Dette krav er super vigtigt da det er en god måde at kunne identificere sine brugere.
En operatør skal bestå af et navn.	Ikke funktionelt	Should have	Dette krav er godt at have med da det ofte er lettere at kunne adskille personer ud fra navne kontra numre og det vil derfor være lettere for en administrator at administrere.
En operatør skal bestå af et initial.	Ikke funktionelt	Could have	Vi synes ikke dette krav er særlig vigtigt men da det ikke kræver så meget at få med, vil vi selvfølgelig tilføje det.
En operatør skal bestå af et cpr nummer.	Ikke funktionelt	Could have	Da vi allerede har et operatør id synes vi ikke dette krav er særlig vigtigt.
En operatør skal bestå af et password.	Ikke funktionelt	Must have	Dette krav er vigtigt da det betyder at kun ønskede folk har adgang til applikationen og specielt da den er web-baseret.

Use Case Diagram

Vores Use case diagram giver os et overblik over hvad vores tanker er omkring designet af vores system. Diagrammet viser os overordnet hvad systemet indeholder og hvilke muligheder de to aktører har.



Use-case beskrivelser

Use Case Name:	Login
Use Case ID:	1
Actors:	Operatør Administrator
Description:	Login i systemet
Preconditions:	Systemet er startet op
Postconditions:	Logget ind
Main flow:	1. Login som operatør 2. Login som administrator
Alternative Flow:	1. Log ud (Use Case ID 7)

Use Case Name:	Administrer operatorer
Use Case ID:	2
Actors:	Administrator
Description:	Superbrugeren kan administrere operatørerne i systemet
Preconditions:	Logget ind som administrator
Postconditions:	Vælg ny Use Case (ID 3, ID 4)
Main flow:	1. Administrer brugere
Alternative Flow:	1. Log ud (Use Case ID 7)

Use Case Name:	Opret operatoer
Use Case ID:	3
Actors:	Administrator
Description:	Opret ny bruger
Preconditions:	Logget ind som administrator
Postconditions:	Ny bruger oprettet
Main flow:	1. Skriv operatør ID (opr_id) 2. Skriv operatør navn (opr_navn) 3. Skriv initialer (ini) 4. Skriv CPR nummer (cpr) 5. Skriv password (password)
Alternative Flow:	1. Log ud (Use Case ID 7)

Use Case Name:	Deaktiver operatoer
Use Case ID:	4
Actors:	Administrator
Description:	Deaktiver en operatør
Preconditions:	Logget ind som administrator
Postconditions:	Operatør deaktiveret, kan aktiveres igen
Main flow:	1. Deaktivér
Alternative Flow:	1. Log ud (Use Case ID 7)

Use Case Name:	Skift password
Use Case ID:	5
Actors:	Operatør
Description:	Skifte password
Preconditions:	Logget ind som operatør
Postconditions:	Password skiftet
Main flow:	1. Indtast gammelt password 2. Indtast nyt password 3. Gentag nyt password
Alternative Flow:	1. Log ud (Use Case ID 7)

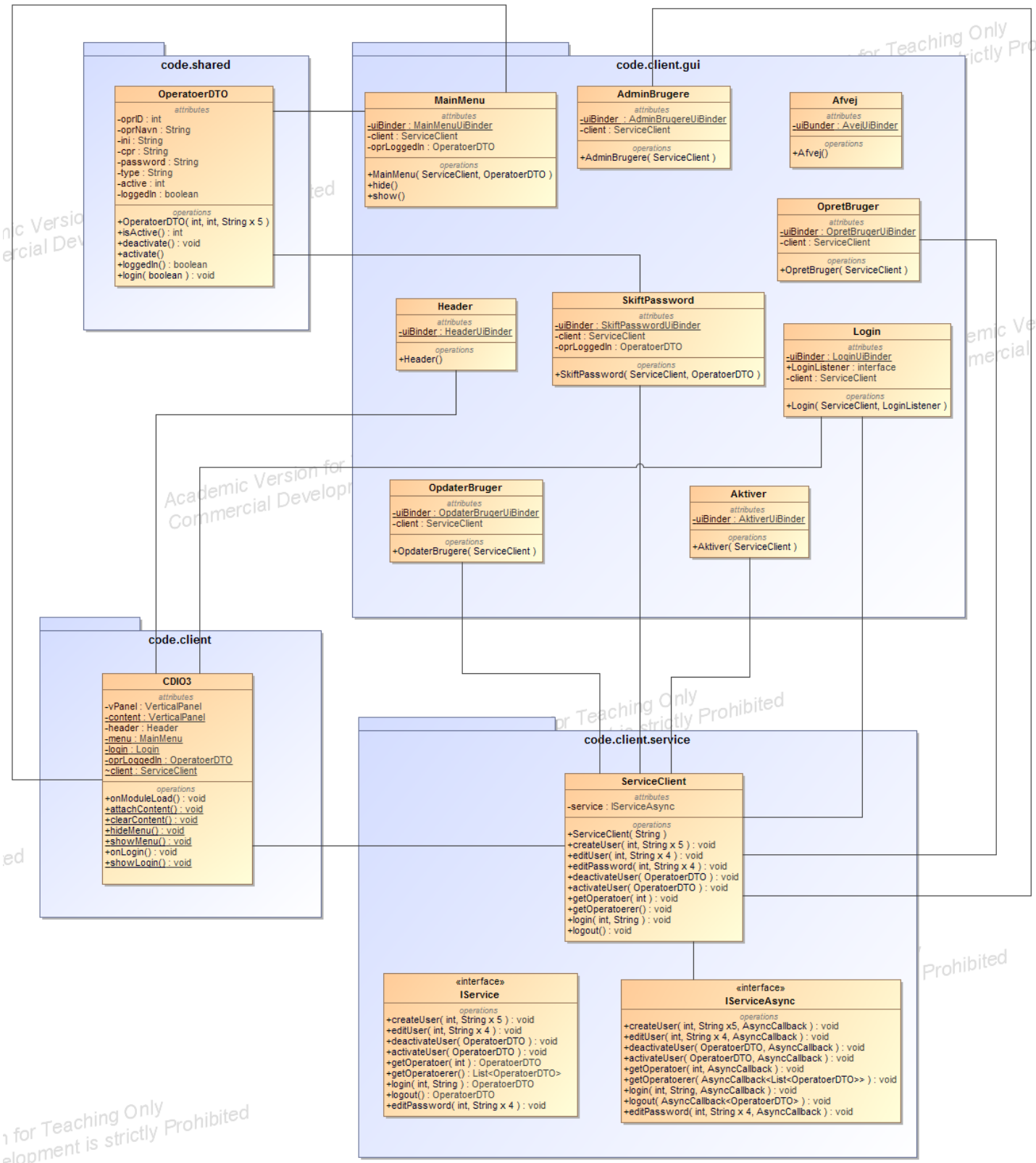
Use Case Name:	Afvejning
Use Case ID:	6
Actors:	Operatør
Description:	Lave en afvejning (ikke implementeret)
Preconditions:	Logget ind som operatør
Postconditions:	Afvejning foretaget
Main flow:	1. Ikke implementeret
Alternative Flow:	2. Log ud (Use Case ID 7)

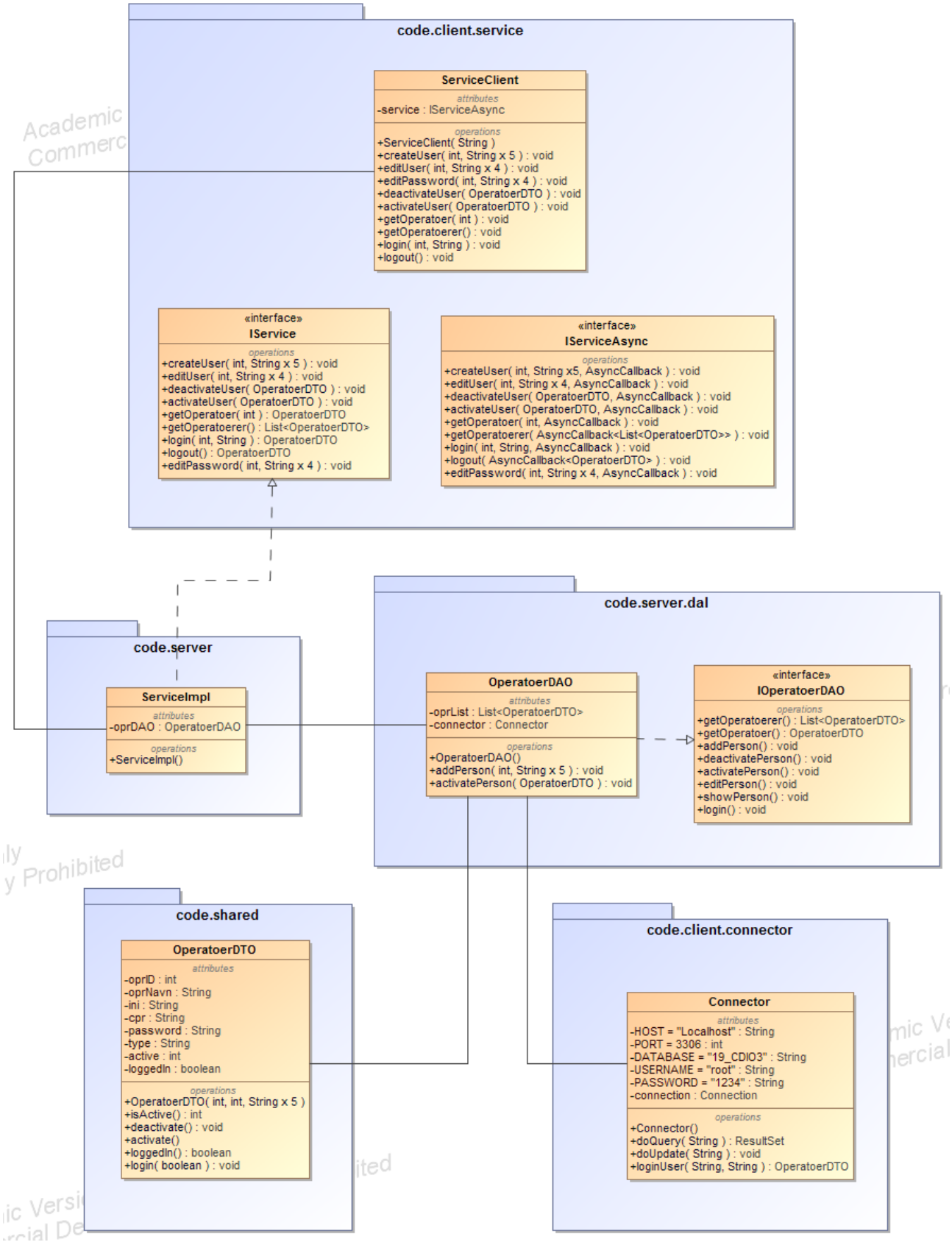
Use Case Name:	Log ud
Use Case ID:	7
Actors:	Operatør Administrator
Description:	Logge ud
Preconditions:	Logget ind
Postconditions:	Logget ud
Main flow:	1. Logge ud

Design

Design-klassediagram

I vores designafsnit har vi udarbejdet et klasse diagram, der viser hvilke klasser vores system indeholder samt relationerne mellem disse. Da det er et stort projekt, har vi valgt at dele diagrammet op i to dele, for at gøre det mere overskueligt. Diagrammet bliver skilt ved code.client.service pakken, og fortsættes på efterfølgende side.


code.shared
OperatoerDTO
attributes
-oprID : int
-oprNavn : String
-ini : String
-cpr : String
-password : String
-type : String
-active : int
-loggedIn : boolean
operations
+OperatoerDTO(int, int, String x 5)
+isActive() : int
+deactivate() : void
+activate()
+loggedIn() : boolean
+login(boolean) : void
code.client.gui
MainMenu
attributes
-uiBinder : MainMenuUiBinder
-client : ServiceClient
-oprLoggedIn : OperatoerDTO
operations
+MainMenu(ServiceClient, OperatoerDTO)
+hide()
+show()
AdminBrugere
attributes
-uiBinder : AdminBrugereUiBinder
-client : ServiceClient
operations
+AdminBrugere(ServiceClient)
Afvej
attributes
-uiBunder : AvejUiBinder
operations
+Afvej()
OpretBruger
attributes
-uiBinder : OpretBrugerUiBinder
-client : ServiceClient
operations
+OpretBruger(ServiceClient)
Header
attributes
-uiBinder : HeaderUiBinder
operations
+Header()
SkiftPassword
attributes
-uiBinder : SkiftPasswordUiBinder
-client : ServiceClient
-oprLoggedIn : OperatoerDTO
operations
+SkiftPassword(ServiceClient, OperatoerDTO)
Login
attributes
-uiBinder : LoginUiBinder
+LoginListener : interface
-client : ServiceClient
operations
+Login(ServiceClient, LoginListener)
OpdaterBruger
attributes
-uiBinder : OpdaterBrugerUiBinder
-client : ServiceClient
operations
+OpdaterBrugere(ServiceClient)
Aktiver
attributes
-uiBinder : AktiverUiBinder
operations
+Aktiver(ServiceClient)
code.client
CDIO3
attributes
-vPanel : VerticalPanel
-content : VerticalPanel
-header : Header
-menu : MainMenu
-login : Login
-oprLoggedIn : OperatoerDTO
~client : ServiceClient
operations
+onModuleLoad() : void
+attachContent() : void
+clearContent() : void
+hideMenu() : void
+showMenu() : void
+onLogin() : void
+showLogin() : void
code.client.service
ServiceClient
attributes
-service : IServiceAsync
operations
+ServiceClient(String)
+createUser(int, String x 5) : void
+editUser(int, String x 4) : void
+editPassword(int, String x 4) : void
+deactivateUser(OperatoerDTO) : void
+activateUser(OperatoerDTO) : void
+getOperatoer(int) : void
+getOperatoerer() : void
+login(int, String) : void
+logout() : void
«interface»
IService
operations
+createUser(int, String x 5) : void
+editUser(int, String x 4) : void
+deactivateUser(OperatoerDTO) : void
+activateUser(OperatoerDTO) : void
+getOperatoer(int) : OperatoerDTO
+getOperatoerer() : List<OperatoerDTO>
+login(int, String) : OperatoerDTO
+logout() : OperatoerDTO
+editPassword(int, String x 4) : void
«interface»
IServiceAsync
operations
+createUser(int, String x5, AsyncCallback) : void
+editUser(int, String x 4, AsyncCallback) : void
+deactivateUser(OperatoerDTO, AsyncCallback) : void
+activateUser(OperatoerDTO, AsyncCallback) : void
+getOperatoer(int, AsyncCallback) : void
+getOperatoerer(AsyncCallback<List<OperatoerDTO>>) : void
+login(int, String, AsyncCallback) : void
+logout(AsyncCallback<OperatoerDTO>) : void
+editPassword(int, String x 4, AsyncCallback) : void


code.client.service
ServiceClient
attributes
-service : IServiceAsync
operations
+ServiceClient(String)
+createUser(int, String x 5) : void
+editUser(int, String x 4) : void
+editPassword(int, String x 4) : void
+deactivateUser(OperatoerDTO) : void
+activateUser(OperatoerDTO) : void
+getOperatoer(int) : void
+getOperatoerer() : void
+login(int, String) : void
+logout() : void
«interface»
IService
operations
+createUser(int, String x 5) : void
+editUser(int, String x 4) : void
+deactivateUser(OperatoerDTO) : void
+activateUser(OperatoerDTO) : void
+getOperatoer(int) : OperatoerDTO
+getOperatoerer() : List<OperatoerDTO>
+login(int, String) : OperatoerDTO
+logout() : OperatoerDTO
+editPassword(int, String x 4) : void
«interface»
IServiceAsync
operations
+createUser(int, String x5, AsyncCallback) : void
+editUser(int, String x 4, AsyncCallback) : void
+deactivateUser(OperatoerDTO, AsyncCallback) : void
+activateUser(OperatoerDTO, AsyncCallback) : void
+getOperatoer(int, AsyncCallback) : void
+getOperatoerer(AsyncCallback<List<OperatoerDTO>>) : void
+login(int, String, AsyncCallback) : void
+logout(AsyncCallback<OperatoerDTO>) : void
+editPassword(int, String x 4, AsyncCallback) : void
code.server
ServiceImpl
attributes
-oprDAO : OperatoerDAO
operations
+ServiceImpl()
code.server.dal
OperatoerDAO
attributes
-oprList : List<OperatoerDTO>
-connector : Connector
operations
+OperatoerDAO()
+addPerson(int, String x 5) : void
+activatePerson(OperatoerDTO) : void
«interface»
IOperatoerDAO
operations
+getOperatoerer() : List<OperatoerDTO>
+getOperatoer() : OperatoerDTO
+addPerson() : void
+deactivatePerson() : void
+activatePerson() : void
+editPerson() : void
+showPerson() : void
+login() : void
code.shared
OperatoerDTO
attributes
-oprID : int
-oprNavn : String
-ini : String
-cpr : String
-password : String
-type : String
-active : int
-loggedIn : boolean
operations
+OperatoerDTO(int, int, String x 5)
+isActive() : int
+deactivate() : void
+activate()
+loggedIn() : boolean
+login(boolean) : void
code.client.connector
Connector
attributes
-HOST = "Localhost" : String
-PORT = 3306 : int
-DATABASE = "19_CDIO3" : String
-USERNAME = "root" : String
-PASSWORD = "1234" : String
-connection : Connection
operations
+Connector()
+doQuery(String) : ResultSet
+doUpdate(String) : void
+loginUser(String, String) : OperatoerDTO

Implementering

Se bilag for vejledning til installation og systemkrav.

Til brugerflade-implementeringen af vores projekt har vi benyttet os af open source værktøjet Google Web Toolkit (GWT), som i bund og grund omdanner ren java kode til javascripts der kan vises som en web side. I GWT har vi benyttet os af UiBinder, som gør det lettere at style de forskellige komponenter. Vi har i høj grad også benyttet os af remote procedure calls (RPC) til at skifte mellem de forskellige sider i vores GUI alt efter om et metodekald gik godt eller dårligt. Dette er gjort for at sikre, at man ikke blev ledt ind på en side hvor man enten ikke skulle have adgang eller sikre mod, at der senere i forløbet ville ske fejl på grund af manglende informationer. Vi har brugt RPC til at kommunikere fra client siden til server siden af vores applikation. Fordelen ved RPC i den forbindelse er, at vi kan benytte asynkrone callbacks, så hele vores web side ikke "fryser" mens, at clienten venter på, at serveren svarer. På server siden har vi koblet en database til, så alle ændringer der bliver lavet bliver gemt og kan hentes når det skal bruges. For at kunne snakke med denne database har vi en Connector klasse. Denne bliver omtalt nede i Database afsnittet i bilag.

Manglende implementation

Vi fik desværre ikke implementeret, at vores webside kan huske hvem der er logget ind, og derfor kan man heller ikke benytte "Skift password" funktionen. Der er desværre heller ikke lavet de forskellige brugertyper (administrator / operatør) og det er igen fordi systemet ikke husker på, hvem der er logget ind. Derudover har vi ikke fået implementeret input validering.

Test

Vi har lavet et par tests der tester vores DAO og sikrer, at de kald vi laver til databasen også virker.

```
@Test
public void testGetOperatoer() throws Exception {
    OperatoerDAO dao = new OperatoerDAO();
    OperatoerDTO opr = dao.getOperatoer(1);
    assertNotNull("Operator was null, check your database", opr);
}
```

På billedet ovenover tester vi på, om vi rent faktisk får en operatoer retur, og hvis ikke vi gør skriver den en meningsfuld besked.

```
@Test
public void testGetOperatoerList() throws Exception {
    OperatoerDAO dao = new OperatoerDAO();
    List<OperatoerDTO> oprList = dao.getOperatoerer();
    assertNotNull("ArrayList was null, check your databse", oprList);
}
```

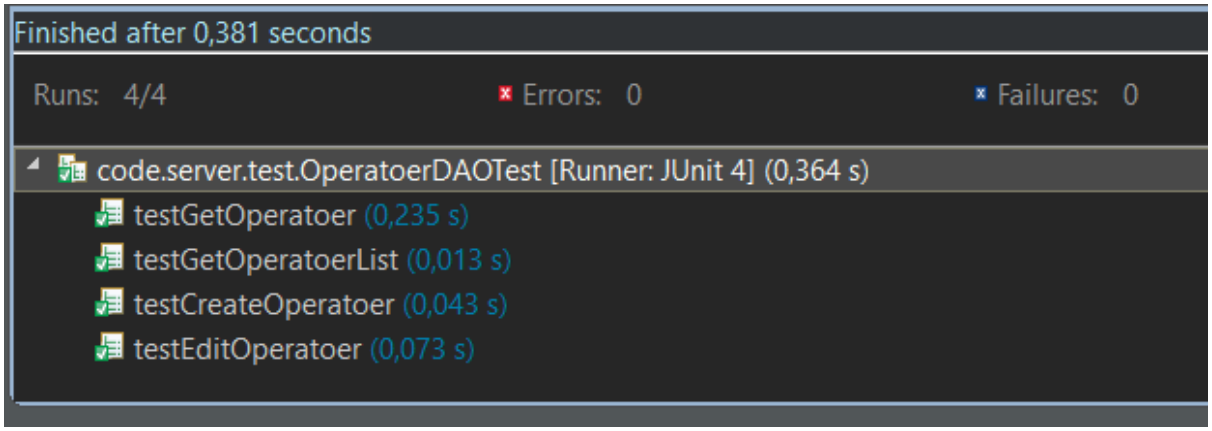
Her tester vi vores getOperatoerer() som laver en ArrayList med alle de operatører vi har i databasen og returnerer den.

```
@Test
public void testEditOperatoer() throws Exception {
    OperatoerDAO dao = new OperatoerDAO();
    dao.editPerson(1, "Søren", "S", "0011223344", "Test1234");
    OperatoerDTO opr = dao.getOperatoer(1);

    assertEquals("Søren", dao.getOperatoer(1).getOprNavn());
}
```

Her tester vi om vi kan redigere en bruger der er gemt i databasen. Når vi redigerer en bruger tillader vi ikke, at alle parametre bliver ændret, men kun ID, navn, initialer, cpr-nummer og password. Når vi tester om det hele gik godt kigger vi dog kun på én af parametrene, da alle er blevet ændret hvis bare den ene af dem er.

```java
@Test
public void testCreateOperatoer() throws Exception {
    OperatoerDAO dao = new OperatoerDAO();
    dao.addPerson(99, "Christian B", "CB", "2525258888", "Hej", "Hej");
    OperatoerDTO opr = dao.getOperatoer(99);

    assertNotNull("Operator was null, check your database", opr);
}
```

Her tester vi vores metode til at oprette brugere. Ligesom rediger brugere tidligere tager den de samme argumenter, men en enkelt undtagelse i form af, at man skal gentage passwordet her. Og endnu engang tester vi ved at se hvorvidt brugeren er oprettet i databasen med getOperatoer() metoden som vi sikrede os virkede længere oppe.



Her ses det så, at alle vores tests lykkedes.

Konklusion

Vi har fået lavet et system, som fungere, men som mangler lidt implementation for at være helt i mål. Vi mangler at få implementeret at en bruger kan skifte password, samt at det kun er superbrugeren som skal kunne administrere de andre brugere.

Bilag

Git

For at få fat i vores projekt, kan man importere det direkte til Eclipse vha. linket https://github.com/Chaaaaap/19_CDIO3_real.git. Dette gøres ved at trykke File → Import → Git → Projects som Git → Clone URI → Indtaster linket i URI-feltet → Next → Finish. Når dette er gjort henter Eclipse selv hele projektet ned. Når først man har hentet projektet ned til Eclipse, henviser vi til opsætning af databasen lidt længere nede, før man fortsætter. Når databasen er korrekt opsat, skal man blot højreklikke på projektet → Run As → Web Application(GWT Super Dev Mode). Når dette er gjort, compiler (behandler) Eclipse projektet, hvorefter der vises et link under fanen "Development Mode". Dette link kopieres og sættes ind i din internet browser.

Databasen

For at kunne bruge vores system, kræver det at man benytter den rigtige database. For at få adgang til den, kræver det for det første at den er oprettet, og sat op som den skal.
Vi har i vores rapport lavet lidt om i databasen end det script som vi fik udleveret fra "Pizza" databasen, og den har vi lagt op som en sql fil inde under mappen "war" i vores projekt.
For at kunne oprette den skal du gå ind under Database Development, og drag and drop database.sql filen. Derefter skal du execute all, ved at højreklikke på skærmen og vælge "Execute all", dernæst er databasen oprettet og sat op som ønsket.
Den sidste ting du skal være opmærksom på, før du kan connecte til databasen er, at gå ind i vores Connector.java klasse og ændre "password" til dit password, hvis du har et for at komme ind.
Så det skulle komme til at se ud som på billedet nedenunder, bare med dit password.

```
public class Connector {
    private final String HOST     = "Localhost";
    private final int    PORT     = 3306;
    private final String DATABASE = "19_CDIO3";
    private final String USERNAME = "root";
    private final String PASSWORD = "";
    private Connection connection;
```

Derefter har du adgang til databasen, som bliver brugt i vores system.

Systemkrav

Vores spil kan køres på alle maskiner der som minimum kører med Windows XP. Det eneste systemkrav der er, er at man skal have installeret JAVA 1.8, for at kunne bruge det library som vi har anvendt (jre 1.8.0_60). Man kan dog godt bruge en nyere udgave, så skal man blot ind og ændre i, hvilke libraries der er tilknyttet projektet. Det er ikke noget større problem, men hvis man ikke føler sig sikker på hvordan man tilføjer / fjerner libraries vil vi ikke anbefale det.