Projektrapport: Olsker Cupcakes

Marts 2020

Klasse: l20dat2ef (e klassen)

Gruppe 407:

- **Navn:** Anne-Maj Andersen
  Mail: cph-al221@cphbusiness.dk
  GitHub: Anne-Maj

- **Navn:** Claes Lindholm
  Mail: cph-cl303 @cphbusiness.dk
  GitHub: por964

- **Navn:** Casper Thomassen
  Mail: cph-ct139@cphbusiness.dk
  GitHub: Goya90

- **Navn:** Jonas Brøchner-Nielsen
  Mail: cph-jb373@cphbusiness.dk
  GitHub: brochnernielsen

Link til projektet på DigitalOcean droplet: http://167.71.73.119:8080/RunningCC-1.0/

Link til projektet på GitHub: https://github.com/por964/Cupcakes2.sem

Indhold

Indledning

Virksomhedsbeskrivelse

Formålet med projektet er at programmere et website til virksomheden Olsker Cupcakes, som er et bageri der er specialiseret i produktion af cupcakes.

Overordnede krav

Virksomheden har beskrevet en række funktionelle krav som websitet skal opfylde. Websitets grundlæggende funktion er, at virksomhedens kunder kan bestille cupcakes via websitet til afhentning i en fysisk butik. Derudover skal der være funktioner til:

- opretning af administrator- og kundekonti
- optankning af kundekontos saldo via administratorkonto
- indkøbskurv
- ordrehåndtering

Teknologivalg

Til projektet er anvendt følgende teknologier:

- **Java 8** (herunder **Tomcat** og **JDBC**) til websitets back-end funktionalitet
- **HTML 5** til websitets front-end opbygning
- **CSS** (herunder **Bootstrap**) til websitets front-end styling
- **IntellJ IDEA version 2019.3.4** som udviklingsværktøj
- **MySQL** til databasen
- **DigitalOcean** som host af websitet og databasen
- **Adobe XD** til mockup

Krav

Virksomhedens vision

Visionen for værdien som websitet vil tilføre virksomheden, er ikke beskrevet af virksomheden selv. Men det må formodes at virksomheden ønsker at øge sin omsætning, da muligheden for at bestille online alt andet lige må formodes at føre til en stigning i ordre. Derudover kan det også øge virksomhedens effektivitet på sigt, da ordre bestilt via websitet ikke kræver lønomkostninger til modtagelse af ordren, som det ellers gør i den fysiske butik.

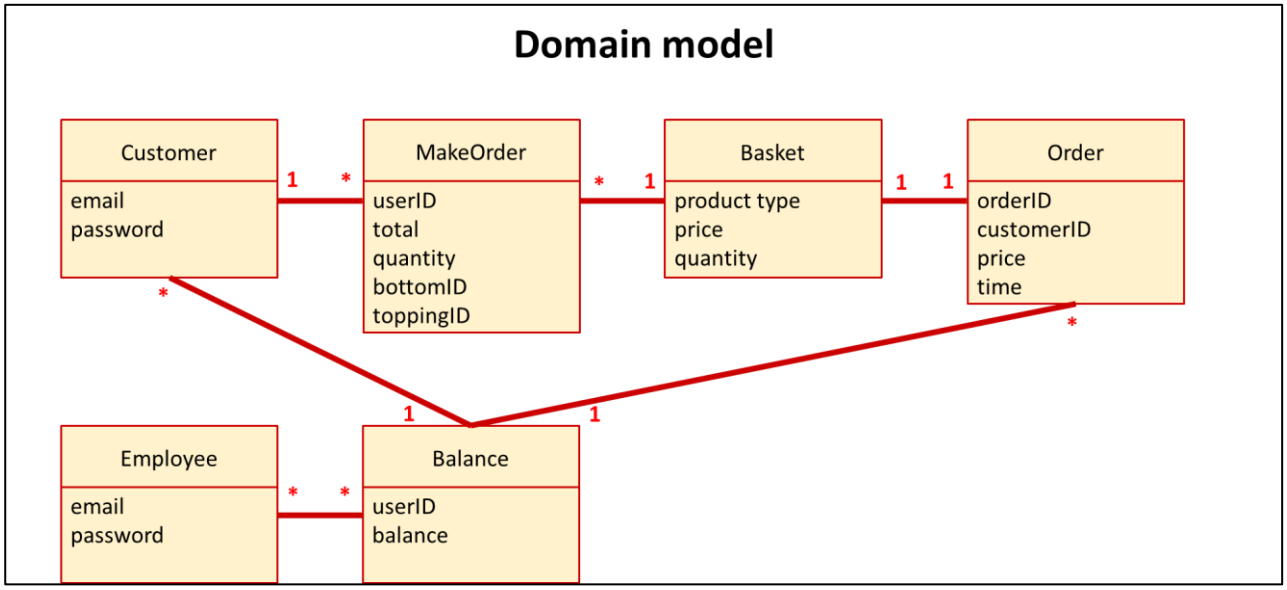
Userstories

Virksomheden har beskrevet en række userstories som er listet her:

- US-1: Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, sådan at jeg senere kan køre forbi butikken i Olsker og hente min ordre.
- US-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.
- US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i MySql, så en kunde kan betale for sine ordrer.
- US-4: Som kunde kan jeg se mine valgte ordrelinier i en indkøbskurv, så jeg kan se den samlede pris.
- US-5: Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne min email på hver side (evt. i topmenuen, som vist på mockup'en).
- US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.
- US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.
- US-8: Som kunde kan jeg fjerne en ordre fra min indkøbskurv, så jeg kan justere min ordre.
- US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.

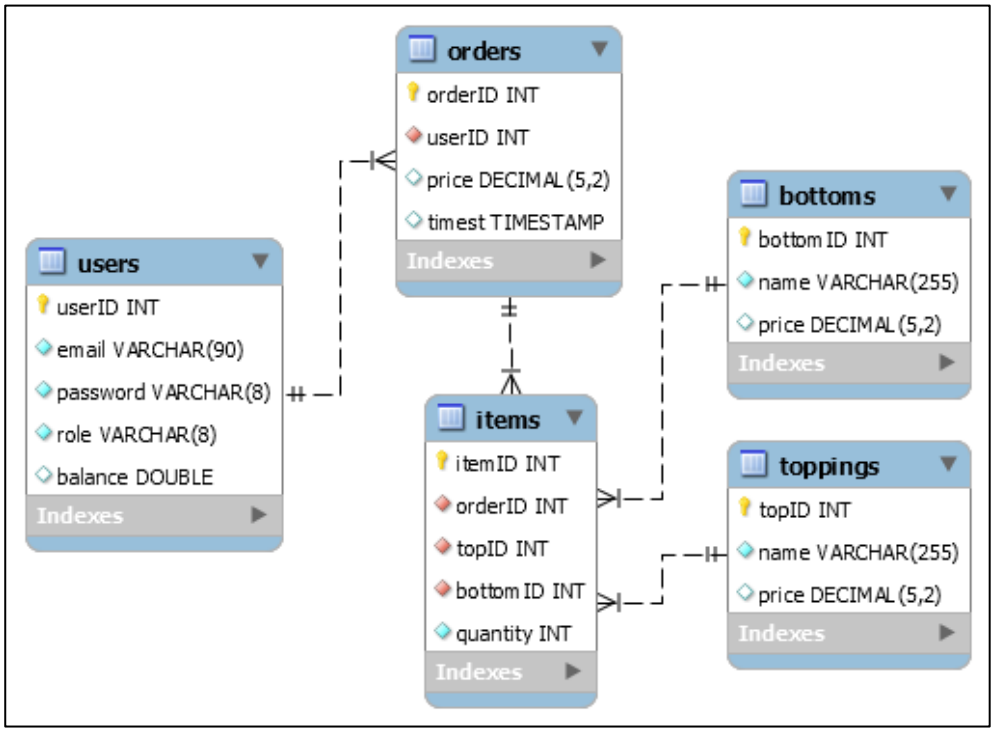
Domænemodel og ER diagram

Domænemodel



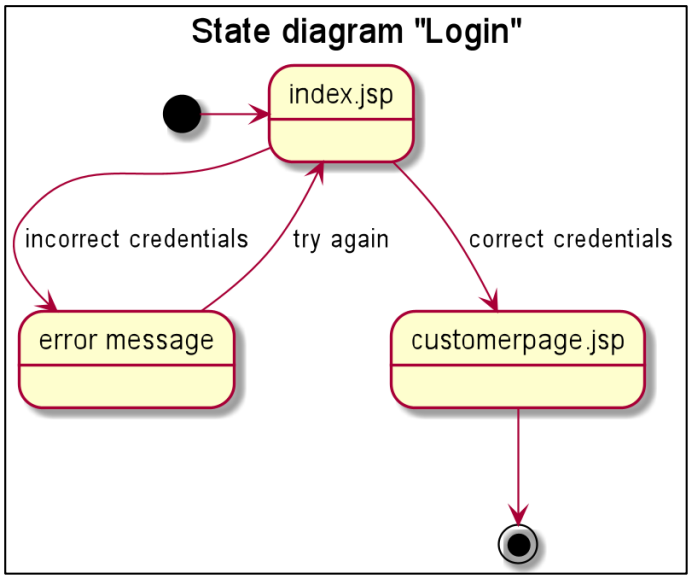
Vi har valgt at lave en meget simpel domænemodel, som viser de centrale funktioner i projektet. Derfor har vi også vedlagt et komplet klassediagram i bilag 1 som PlantUML kode.

ER diagram



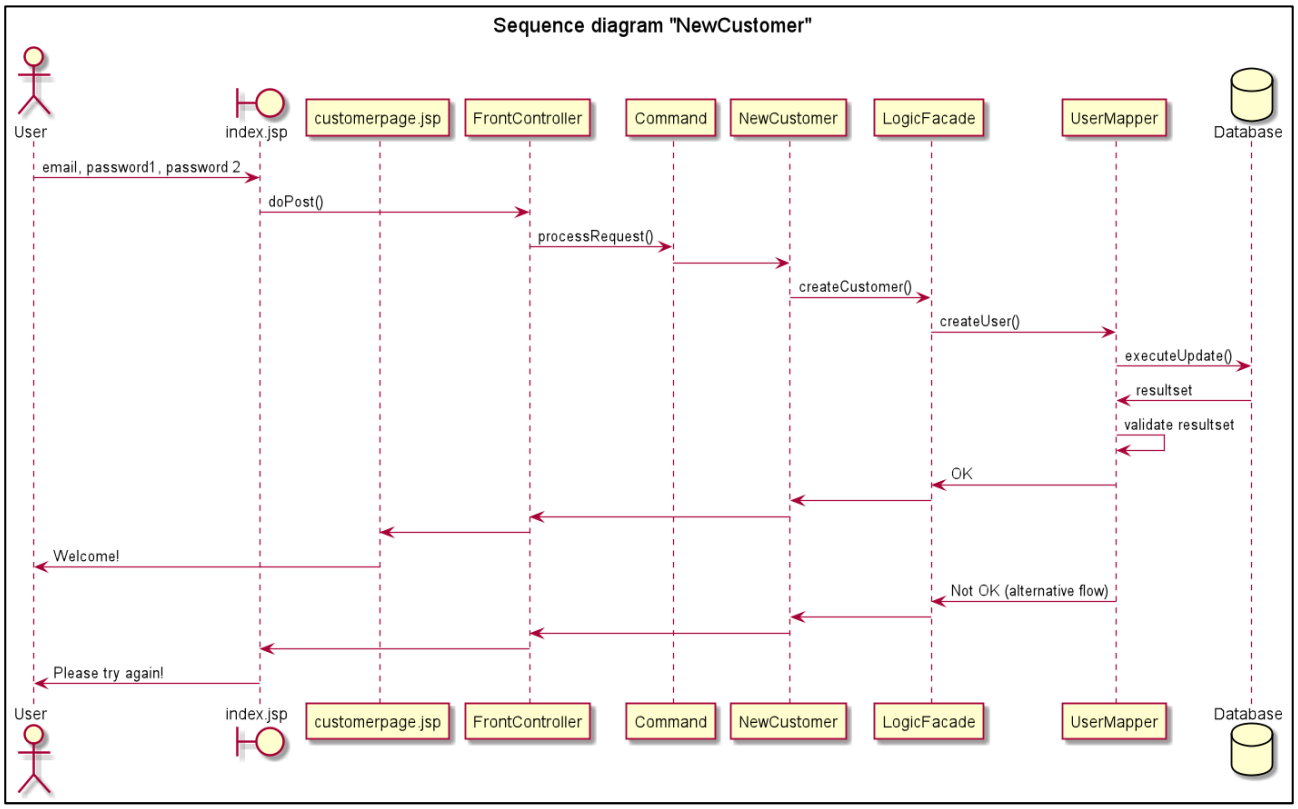
3. normalform er overholdt for tabellerne i databasen.

Tilstandsdiagram



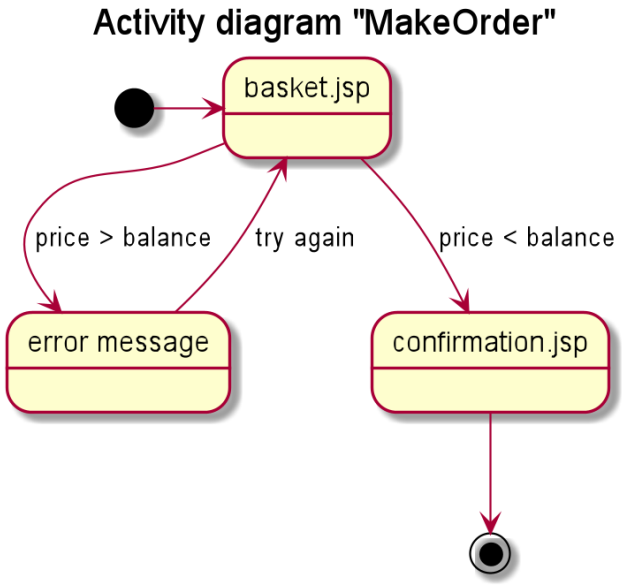
Koden for PlantUML diagrammet findes i bilag 2.

Sekvensdiagram



Koden for PlantUML diagrammet findes i bilag 3.

Aktivitetsdiagram



Koden for PlantUML diagrammet findes i bilag 4.

Særlige forhold

- I forhold **sikkerhed** er kodeordet ”hidden” ved oprettelse af bruger. Derudover har vi ikke lavet yderligere tiltag for at forbedre sikkerheden, ud over den der allerede er indbygget i skabelonen vi har fået udleveret – herunder WEB-INF systemet.
- Mange informationer gemmes i **sessions**, som fx:
  - arraylister med kunder, ordrer, og attributter såsom balance, email osv.
  - beskeder på balance.jsp og showAllOrders.jsp
  - diverse attributter fx user, role, email og userID i User klassen
- **Exceptions** håndteres gennem LoginSampleException, ClassNotFoundException og SQLExceptions
- Der er brugt to **brugertyper** i projektet: ’customer’ og ’employee’. De holdes adskilt allerede ved login, på den måde at ’customer’ ikke får adgang til employeepage.jsp og det dertilhørende funktion - og omvendt

Status på implementering

Vi har i stort træk implementeret alt hvad vi havde planlagt. Vi har dog ikke stylet vores sider så meget som vi kunne have ønsket, da vi har fokuseret på at funktionaliteten skulle være i orden først. Det er fx knapper hvor vi gerne ville have brugt Bootstrap, men i stedet er de blevet mere simple.

Bilag 1 – PlantUML kode for komplet klassediagram

```
@startuml

set namespaceSeparator none

title Cupcake project - domain model


package HttpServlet <<Rectangle>> {
class HttpServlet {
}
}

package cupcakeUtil <<Rectangle>> {

+class Initializer {
-bottomlist: List<Bottom> <<static>>
-topList: List<Topping> <<static>>
+getBottomList() <<static>>
+getTopList() <<static>>
}
}

package PresentationLayer <<Rectangle>> {

+abstract class Command {
-void initCommands() <<static>>
-Command() <<static>>
-Command(HttpServletRequest request) <<static>>
-{abstract} execute()
}

+class Balance extends Command {
execute()
userID: int
double: beloebTilIndsaettelse
}

+class DeleteItem extends Command {
execute()
counter: int
price: Double
total: Double
}

+class FrontController extends HttpServlet {
#void processRequest()
#void doGet()
#void doPost()
+getServletInfo()
}

+class Logout extends Command {
execute()
}

+class MakeOrder extends Command {
execute()
brugerID: int
total: Double
saldo: Double
}

+class NewCustomer extends Command {
execute()
email: String
```

```
password1: String
password2: String
}

+class NewEmployee extends Command {
execute()
email: String
password1: String
password2: String
}

+class Redirect extends Command {
execute()
destination: String
}

+class ShowAllCustomers extends Command {
execute()
customerList <<ArrayList>>
}

+class ShowAllOrders extends Command {
execute()
orderList <<ArrayList>>
}

+class Login extends Command {
execute()
email: String
password: String
}
}

package FunctionLayer <<Rectangle>> {


+class Bottom {
-bottomID: int
-name: String
-price: Double
+Bottom(name, price)
+Bottom(bottomID, name, price)
+getBottomID()
+getName()
+getPrice()

}

+class Item {
-topping: String
-bottom: String
-quantity: int
-itemPrice: Double
+Item(topping, bottom, quantity, itemPrice)
+getTopping()
+getBottom()
+getQuantity()
+getItemPrice()
}

+class LogicFacade {
+login(email, password) <<static>>
+createCustomer(email, password) <<static>>
+createEmp(email, password) <<static>>
+updateBalance(userID, beloebTilIndsaettelse) <<static>>
+showBalance(email) <<static>>
+showAllOrders() <<static>>
+getTops() <<static>>
+getBottoms+() <<static>>
+showAllCustomers+() <<static>>
}
```

```
+class LoginSampleException extends Exception {
+LoginSampleException(msg)
}

+class Order {
-ID: int
-customerID: int
-price: double
-time: Timestamp
-items: List<Item>
+Order(ID, customerID, price, time)
+Order(customerID, price)
+void addCake(item)
+getID()
+void setID()
+getCustomerID()
+void setCustomerID(customerID)
+getPrice()
+void setPrice(price)
+getTime()
+void setTime(time)
+getItems()
+void setItems(items)
}

+class Topping {
-topID: int
-name: String
-price: Double
+Topping(name, price)
+Topping(topID, name, price)
+getTopID()
+getName()
+getPrice()
}

+class User {
-UserID: int
-email: String
-password: String
-role: String
-balance: double
+User(userID, email, password, role, balance)
+User(email, password, role, balance)
+User(email, password, role)
+getUserID()
+void setUserID(userID)
+getEmail()
+void setEmail(email)
+getPassword()
+void setPassword(password)
+getRole()
+void setRole(role)
+getBalance()
+void setBalance(balance)
+toString()
}
}

package DBAccess <<Database>> {

+class UserMapper {
+void createUser(user) <<static>>
+login(email, password) <<static>>
+void createEmp(user) <<static>>
+void updateBalance(userID, beloebTilIndsaettelse) <<static>>
+getBalance(userID) <<static>>
+void payOrder(user) <<static>>
+showBalance(user) <<static>>
+showAllCustomers(user) <<static>>
```

```
}

+class Connector {
-URL: String <<static>>
-USERNAME: String <<static>>
-PASSWORD: String <<static>>
-singleton: Connection <<static>>
+void setConnection(con) <<static>>
+connection() <<static>>
+void setDBCredentials() <<static>>
}

+class CupcakeMapper {
+getBottomName(id) <<static>>
+getToppingName(id) <<static>>
+getToppingPrice(id) <<static>>
+getBottomPrice(id) <<static>>
+getBottomID(name) <<static>>
+getToppingID(name) <<static>>
+getBottoms() <<static>>
+getTops(id) <<static>>
}

+class OrderMapper {
+showAllOrders() <<static>>
+lastOrderID() <<static>>
+void createOrder() <<static>>
+void insertItems() <<static>>
}
}


@enduml
```

Bilag 2 – PlantUML kode for tilstandsdiagram

```
@startuml

Title State diagram "Login"
State "error message" as error

[*] -> index.jsp
index.jsp --> customerpage.jsp : correct credentials
error --> index.jsp : try again
index.jsp --> error : incorrect credentials
customerpage.jsp --> [*]

@enduml
```

Bilag 3 – PlantUML kode for sekvensdiagram

```
@startuml

title Sequence diagram "NewCustomer"

actor User
boundary index.jsp
participant customerpage.jsp
participant FrontController
participant Command
participant NewCustomer
participant LogicFacade
participant UserMapper
database Database

User -> index.jsp : email, password1, password 2
index.jsp -> FrontController : doPost()
FrontController -> Command : processRequest()
Command -> NewCustomer :
NewCustomer -> LogicFacade : createCustomer()
LogicFacade -> UserMapper : createUser()
UserMapper -> Database : executeUpdate()
Database -> UserMapper : resultset
UserMapper -> UserMapper : validate resultset
UserMapper -> LogicFacade : OK
LogicFacade -> NewCustomer
NewCustomer -> FrontController
FrontController -> customerpage.jsp
customerpage.jsp -> User : Welcome!
UserMapper -> LogicFacade : Not OK (alternative flow)
LogicFacade -> NewCustomer
NewCustomer -> FrontController
FrontController -> index.jsp
index.jsp -> User : Please try again!
@enduml
```

Bilag 4 – PlantUML kode for aktivitetsdiagram

```
@startuml

Title Activity diagram "MakeOrder"
State basket.jsp
State "error message" as error
State confirmation.jsp

[*] -> basket.jsp
basket.jsp --> confirmation.jsp : price < balance
error --> basket.jsp : try again
basket.jsp --> error : price > balance
confirmation.jsp --> [*]

@enduml
```