Johannes Fog - Carport Case - Rapport

Jacob Borg
Daniel Lindholm
Nikolaj Thorsen Nielsen
Stephan Marcus Duelund Djurhuus

May 24, 2018

Link til GitHub repository
Link til websitet
Link til Javadoc

Contents

1 Indledning

Vi er blevet kontaktet af Johannes Fog Værebro, der har spurgt om vi kan lave et program og tilhørende hjemmeside til deres salg af carporte.

1.1 Baggrund

Johannes Fog er en koncern der både har design & bolighuse og trælast & byggecenter. Vi vil i dette project fokusere på Fog Værebro, der har bedt os om at opdatere deres system. Fog Værebro sælger som de andre trælast & byggecentre, træ, byggematerialer og alt det du behøver til hus og have inden for f.eks. maling, bad og VVS, beslag, elartikler og lamper samt haveredskaber, grill og havemøbler, men derudover har de gjort det til deres varemærker at være specialister i carporte. Det er i den forbindelse at vi er blevet bedt om at udvikle et system, med tilhørende hjemmeside, til at erstatte deres nuværende system, da de har erkendt at det er outdated. Med systemmet skal man kunne gå på hjemmesiden og bestille en carport, systemmet skal kunne udregne et stykliste af materialer, en medarbejder skal kunne gå ind og give en kunde særlige tilbud, og en kunde skal kunne se sin ordre.

1.2 Teknologivalg

Projektet er et Maven 3.1 projekt skrevet i Netbeans IDE 8.2. Databasen, mysql 5.1.39, er lavet i MySQL Workbench 6.3ce, og ligger på en Ubuntu 16.04.3 x64 server, hvor websitet er deployet vha. apache-tomcat-8.0.32. Sprogene der er brugt i koden er java jdk 1.8.0_141, HTML5 og javax 7.0.

2 Krav

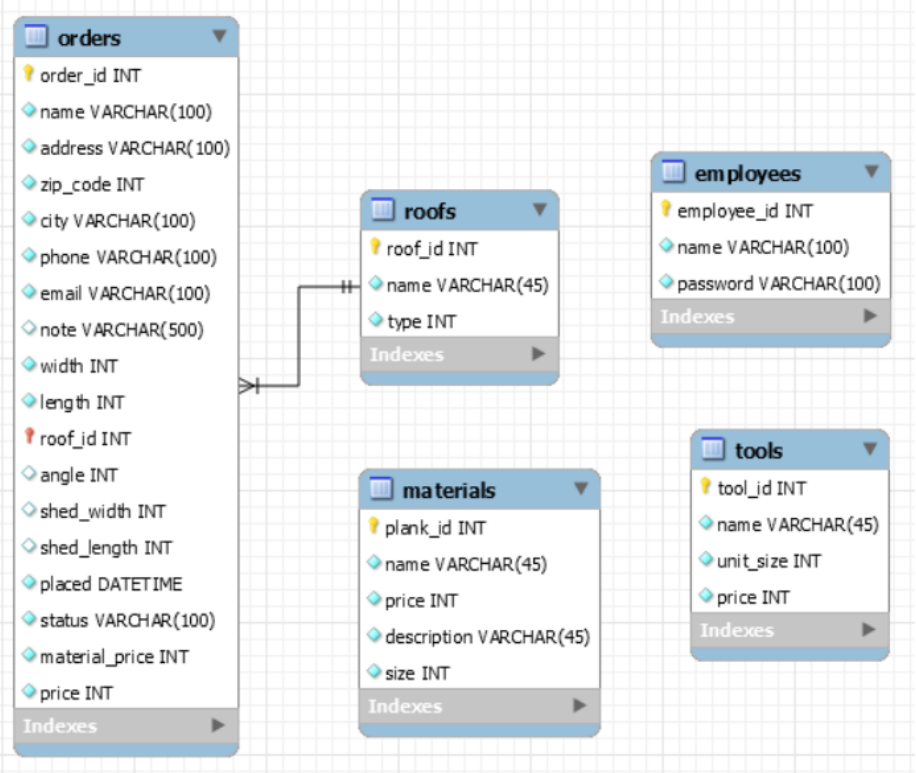
På websitet skal man som kunde kunne bestille en carport. Man skal kunne vælge om taget skal være med eller uden rejsning, og hvis man har valgt med skal man kunne vælge hvor meget hældning man vil have på taget. Man skal kunne vælge om man vil have et redskabskur, og hvor stort det skal være. Når man bestiller en carport skal man kunne indtaste sine oplysninger, navn, telefonnummer, email og adresse. Når man har bestilt en carport kan man se alle de ordre der er bestilt med den samme email. Man skal som bruger kunne se alle sine tidligere ordre ved at indtaste sin email.
Som medarbejder skal man kunne se alle ordre. Man skal kunne ændre status og pris på en ordre. Man skal kunne navne og priser på de forskellige typer træ, skruer og beslag.
Systemet skal kunne udregne en stykliste udfra en ordre.

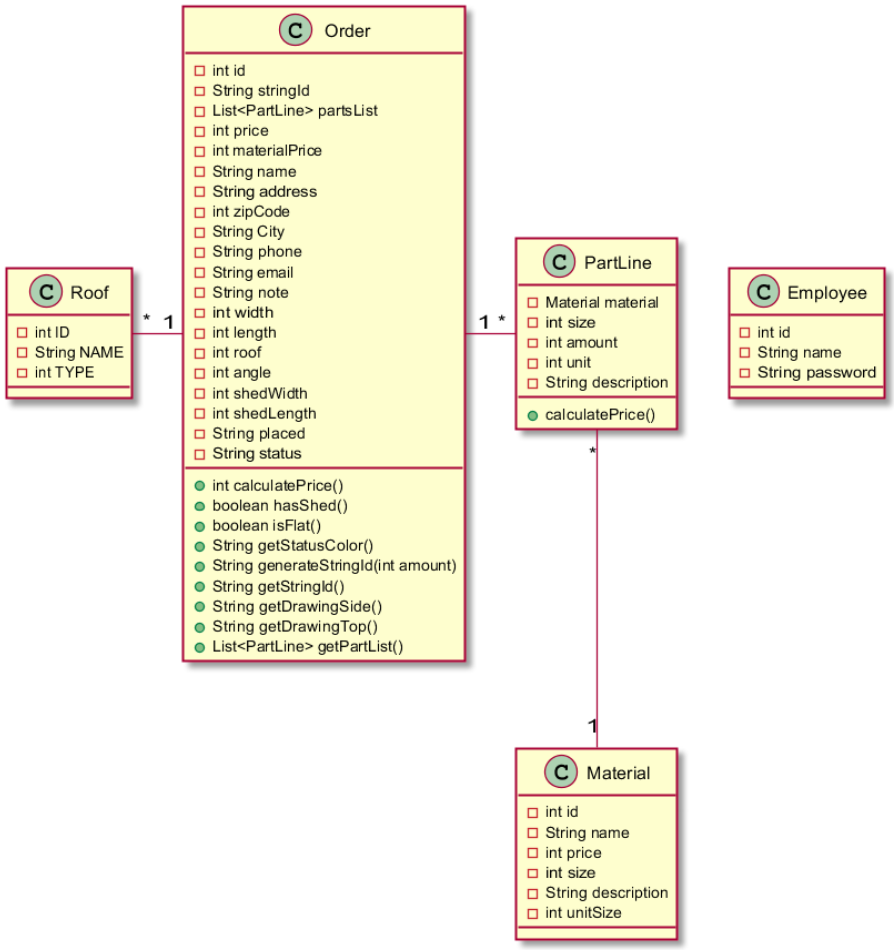
2.1 Overordnet beskrivelse af virksomheden

2.2 Arbejdsgange der skal IT-støttes

2.3 Scrum userstories

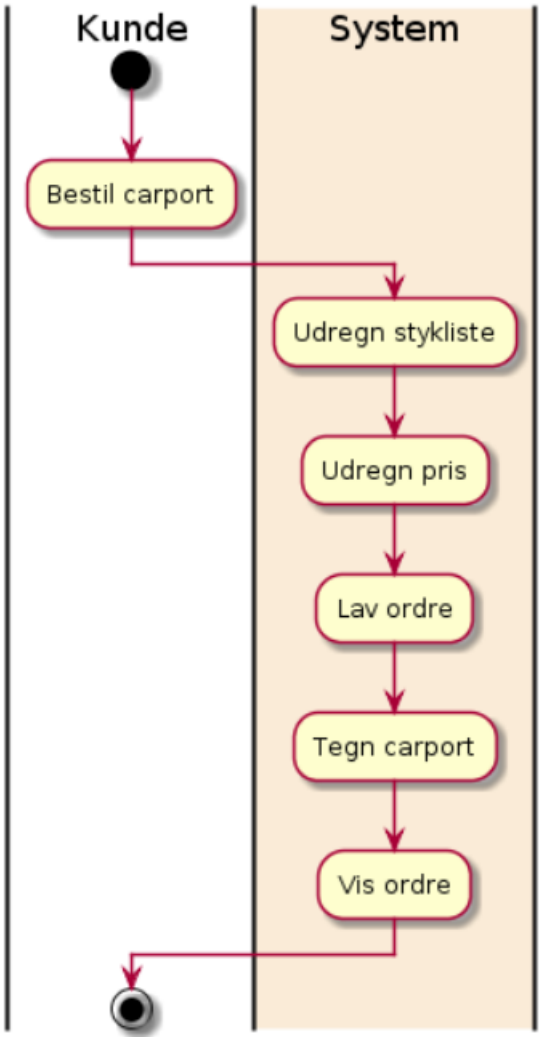
3 Domæne model og ER diagram

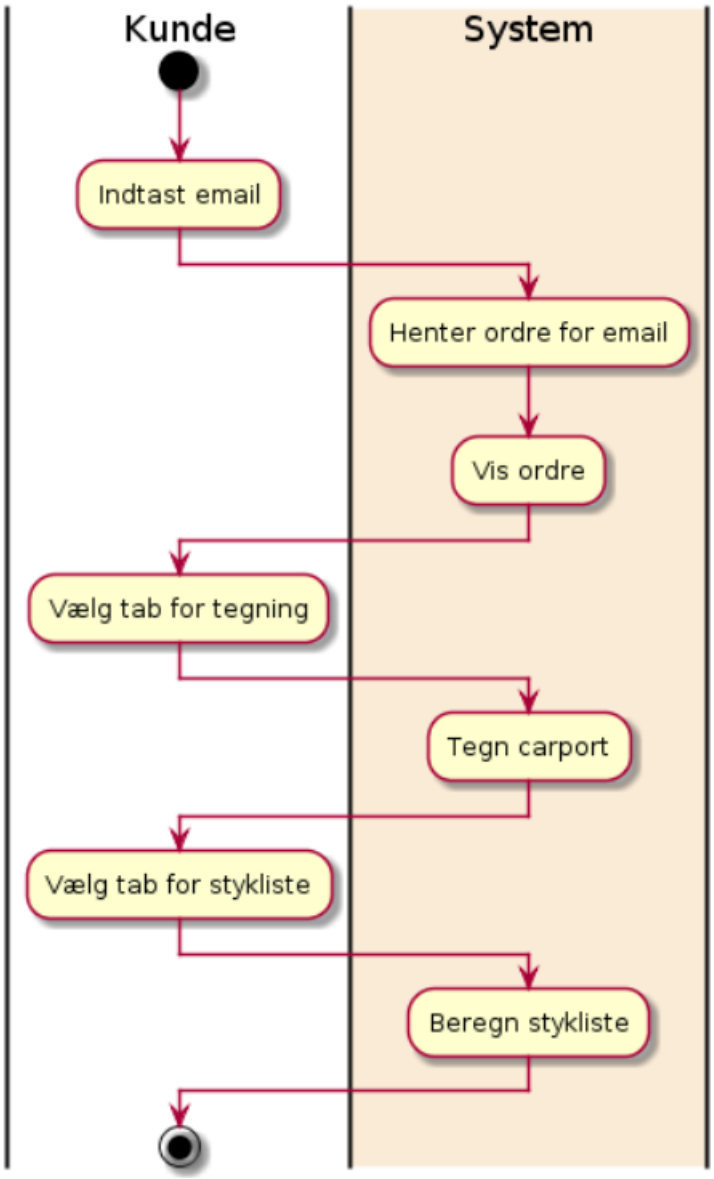



Order
int id
String stringId
List<PartLine> partsList
int price
int materialPrice
String name
String address
int zipCode
String City
String phone
String email
String note
int width
int length
int roof
int angle
int shedWidth
int shedLength
String placed
String status
int calculatePrice()
boolean hasShed()
boolean isFlat()
String getStatusColor()
String generateStringId(int amount)
String getStringId()
String getDrawingSide()
String getDrawingTop()
List<PartLine> getPartList()
Roof
int ID
String NAME
int TYPE
* 1
1 *
PartLine
Material material
int size
int amount
int unit
String description
calculatePrice()
Employee
int id
String name
String password
*
1
Material
int id
String name
int price
int size
String description
int unitSize

4 Navigationsdiagram

5 Sekvensdiagrammer





6 Særlige forhold

7 Udvalgte kodeeksempler

8 Status på implementation

9 Test

Som udvikler er test af programmet en vigtig del af processen. Lever koden ikke op til de givne test må den revurderes og testes igen, indtil alle testene går igennem. En måde at sikre at alle metoder gennemføre deres test er at bruge Test Driven Developmemt (TDD). TDD er en god måde at sikre signe test på, da man starter med testen og derefter udarbejder metoden til at løse den. Desværre lærte vi først denne udviklingsmetode midvejs gennem forløbet hvilket gjrorde den svær at implimentere, da det er en metode der skal implementeres i starten af projektet.
Vores testmiljø består af JUnit test og et plugin som hedder TikiOne JaCoCoverage. Vores JUnit Mapper test er koblet til vores aktuelle database, hvilket vi ved ikke er den optimale måde at teste på. Der er andre måder, som f.eks. at oprette diverse test tables i databasen som er identiske til de aktuele og som kun har forbindelse til vores test run. Dette vil kun sikre at vores test ikke skabte konflikt i vores program senere hen, men vil ikke reducere forbindelsen mellem program og server. De resterende JUnit test er blevet udarbejdet på den traditionelle whitebox måde, da der ikke er nogle specielle faktore der spiller ind. Whitebox tetsing er en test metode der giver person som tester mulighed for at følge med i kode contra Blockbox testing hvor der ikke er adgang til koden. Vi har brugt et pugin der heder TikiOne JaCoCoverage, som skaber at overblik over hvilke metoder der er blevet testet.

10 Process

10.1 Arbejdsprocessen faktuelt

10.2 Arbejdsprocessen reflekteret