

Delrapport 2

PROJEKTMANAGER:
PATRICK KRØLL BRANDT - 081194
PTXDK@LIVE.DK

PROJEKTGRUPPE:
BJØRN RIISE BRENDORP - 220793
BJORN.BRENDORP@GMAIL.COM

EJSTRUP - 170782
CARSTENEJSTRUP@HOTMAIL.COM

PHUC TIEN NGUYEN - 010993
PUKKIE93@GMAIL.COM

INSTRUKTOR: KASPER PASSOV
KASPER.PASSOV@GMAIL.COM

Contents

1	Abstract	5
2	The IT-project objectives and frameworks	5
2.1	System Definition - FACTOR	5
3	Requirements specification for IT solution	6
3.1	Functional vs Non-functional	6
3.1.1	Functional Requirements	6
3.1.2	Nonfunctional Requirements	6
3.2	Use Case Diagram	7
3.3	Specified Use Cases	7
4	Class Diagram	9
5	Systemdesign	11
6	UI design and interaction	12
6.1	Illustration of different Views	12
6.2	Illustration of View Relations	13
7	Annex 1: VersionControl	14
8	Annex 2: Changelog	14
9	Annex 3: Timeline	14

Literature Review

Gould, J. D. and Lewis, C. (1985), Designing for usability: key principles and what designers think. Commun. ACM 28, 3 (March 1985), 300-311.

This article is about the view-points of Gould J.D. and Lewis C on the topic of designing systems. They have three major principles, they find to be important for system-development. These points are as follows:

- 1) Early focus on users and tasks - Understand the users and design the system interface and usability accordingly.
- 2) Empirical measurements - have users test simulations and prototypes while recording the results of these tests.
- 3) Iterative design - handling the errors and corrections found during testing. This makes the process iterative, as you design, test, correct, repeat.

These principles were made to ensure a program as well-suited for the user as possible. To test if these principles were used in practice, an experiment in 1981-82 was run among 447 programmers. The majority of the programmers in the test mentioned 0-1 of the principles. A majority also mentioned implementation of the early user focus in their system-designing. To conclude this test, the authors mention that they acknowledge that they in no way claim, that these principles are the universally best guidelines.

To specify the procedure of designing a program based on these principles, they went into a bit more detail with them:

To have proper focus on interaction with users early on, it is not enough simply to read about them, or receive information from 2nd-hand sources. This has to be built through interaction. To accomplish the principle of empirical measurements, one must design good and creative tests for different users to try. In the end, one must analyse the data so that the optimal parameters for the program can be found. In the end of each iterative cycle, one must implement the new design-choices based on user input. This is done to make the interface as user-friendly and audience-specific as possible, while still taking user diversity into account.

Next the authors write about five common points they face when advocating these principles.

- 1: Programmers evaluate these points to not be important enough to be worth following.
- 2: Programmers misunderstanding the points, as to i.e. testing one-self instead of getting users to test.
- 3: User diversity is misvalued
 - this can mean it is either undervalued or overvalued. It is mentioned that this might stem from the programmers' belief, that they know better than the user.
- 4: The programmer might hold conflicting approaches to these principles. One might believe that:
 - That if the fundamentals are good, user input is not necessary
 - That if one follows the guidelines for user-interface it is sufficient.
 - That one gets it right the first time (conflicts with the iterative approach)
- 5: The programmer might believe, that the iterative approach is too impractical:
 - The programming process will be lengthened
 - the iterative process is just expensive fine-tuning
 - If the system itself has neat perks, it will automatically succeed

The authors responded to most of these by saying, that different programmers value principles and design-choices differently. But the authors believe that it is at least one of the ways with highest success rate with which to approach system designing, although it might make lengthen the devel-

opment process.

Lastly they make a use-case for IBM's audio distribution, where they explain the process with which, they built up the system. All of the three principles were worked through in this process, which wraps up the article.

These principles fit very nicely with the iterative method, which is introduced in the book "Object oriented software engineering". Instead of making one design to begin with, that encompasses all of the system, its suggested that you make it iteratively and build it up in cooperation with the users.

We find that these principles are very important aswell, although, to some degree, you have to make a majority of decisions yourself. Often times, users are absolute novices to programming, and frequently not aware of possibillities and limitations. You shouldn't be limited in possibillities because of a client or users requests to a program, but at the same time, things shouldn't be implemented contrary to the users wishes.

textbfParnas, D. L. and Clements, P. C. (1986), A rational design process: How and why to fake it. IEEE Trans. Softw. Eng. 12, 2 (Feb. 1986), 251-257. The original version from Proceedings of the TAPSOFT conference, Formal Methods and Software Development, Berlin 1985, pp. 80-100. A rational design process: How and why to fake it. Parnas, D. L. and Clements, P. C.

In 1985, David Lorge Parnas and Paul C. Clements at the Proceeding of the TAPSOFT conference, they gave their bid on how a god system should be designed and how it should be accompanied by documentation, which should be given full attention in such a way that a system can follow this document as a complete guide.

In their article, Parnas and Clements, claims that the design process of a program will never be more than an idealization and there are too many factors in play preventing the software program to be developed in a rational way. Factors such as the communication about the desired systems functionality, even if the people commissioning a system could fully explain their needs and desires, no human can comprehend the full detail needed to complete a system.

Parnas and Clements believe that designers need guidance, because it is easy to get overwhelmed when systems get to big. If we could design a rational standard process, we could easily identify where in the process a project is at any given time, or locate areas in need of work. Outsiders should always review a good managed project regularly and with a standard process, this would be a lot easier.

The description of the development should be done in stages and thereby clearly specify what product is to be worked on next and the criteria. A stage product should specify the skills and information needed in order to be completed. This kind of description helps when reviewing a project and measure its progress.

A requirement document done by the user, can guide system developers to the behaviour desired and thereby avoiding having to make requirement decisions during the design process. Without a requirement document, duplication and inconsistency in the system is of high possibility to arise. This document should be complete in the sense that a complete product satisfy all statements in the

document. If a system might have to change in the future, then the requirement document should define areas considered likely to change. Constraints and undesired events in the system should also be present in the document.

Parnas and Clements now believe that when all documentation and design choices have been carried out, it is finally time to start programming which should go by quickly and smoothly now.

As Parnas and Clements points out, many of today's documentation is incomplete and insufficient, mostly because programmers find them as a necessary evil and therefore do not give this part of the work the necessary dedication. The documents often contain boring prose, where a diagram or a formula could better and easier for future reading and understanding. A document should go through the same iterate review technique as a written system.

By writing complete documents before programming, Parnas and Clements believe that it can fake the idea of an ideal process. It is impossible to write the perfect document, and problems will arise while writing executable code, these points should then be corrected in the document. The document should still contain all the alternative design choices that we did not implement into the system, with detailed explanation why.

To conclude we believe that Parnas and Clements have a good point in the fact that programming is an iterative process and therefore it is not possible to write a perfect document before working on executable code. However, given a good standard approach and dedication, the documentation can come close to being perfect or as Parnas and Clements puts it, faked, thereby giving programmers the best possible options to complete and make systems that fits the desired goals.



1 Abstract

This project was given to use by the company ~~FalkAndersensVine~~, which is a small company, that works as the middleman between the wine producers with customers. The project is regarding the making of a system, that works on most mainstream devices (i.e. computer, Ipad, tablet devices), which can keep track of their sales, taxes and redundant wine. We decided to implement this system as a webpage, so that it can be accessed on the multiple different devices. We will connect it to a database, and use the programming language PHP to make the search, add/remove and other necessary functions. Lastly we will link it to their current webpage, so that their customers have easy access to their catalog, and they themselves have easy access to the administrative parts of the webpage.

2 The IT-project objectives and frameworks

2.1 System Definition - FACTOR

Functionality:

- New wine, when Falkandersenvine has a new wine for sale, it can be implemented into the system
- Update wine, when Falkandersenvine receives more of an already existing wine in the database and the number needs to be updated
- Remove wine, when a wine is no longer being sold and taken out of stock
- Provides a good overview

Application domain:

- The administrator (Falkandersenvine)
- Customers

Conditions:

The conditions to use this program are the following:

- One must have internet access in order to access the webpage
- It must be accessible on mobile phones

Technology:

- VPS server or Raspberry PI
- Apache
- PHP
- Jason
- HTML
- CSS

**Objects:**

- Wine
- Transaction
- User
- Budget
- View

Responsibility:

- Keep a overview of the number of wines

3 Requirements specification for IT solution

3.1 Functional vs Non-functional

3.1.1 Functional Requirements

Functions for the customers:

- A database that gives an overview of the wines, based on the actual amount of wines currently in stock.
- A search function to make browsing the wine selection easier

Functions for wine-cataloger:

- add, remove or edit wines from their catalogs.
- add, remove or edit transactions and keep track of balance
- add, remove or edit redundant wine

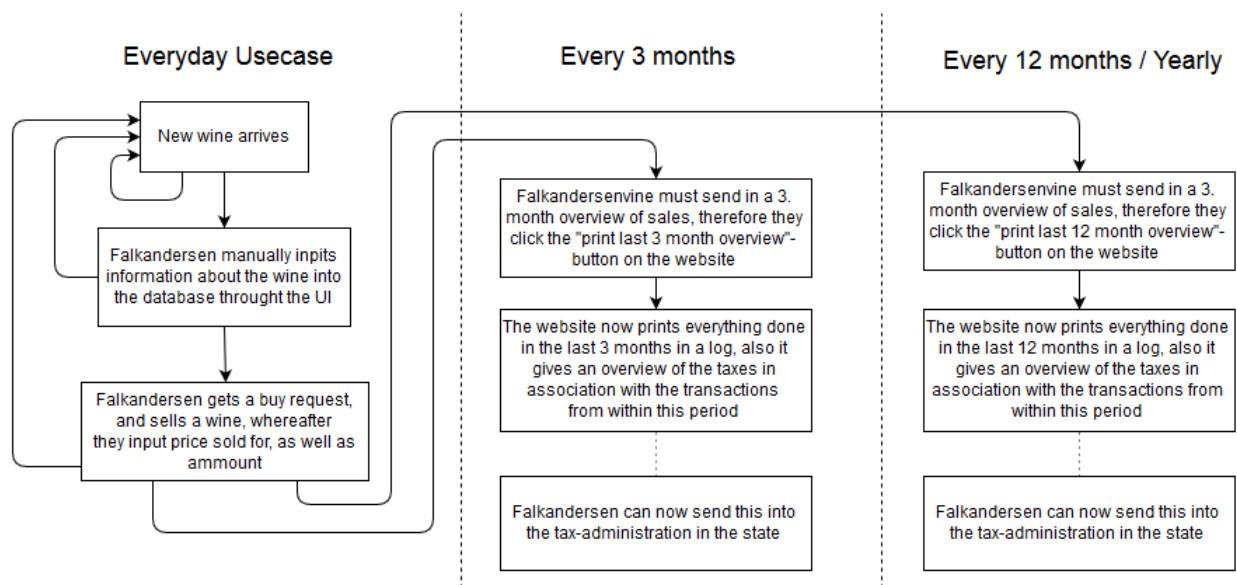
Functions for admins:

- Add, remove and edit users

3.1.2 Nonfunctional Requirements

- It must be browser-independent
- It must be accessible on most mainstream devices
- It must be user-friendly and not have high requirements to the user's knowledge of it

3.2 Use Case Diagram



The Diagram describes a general workflow in the system, in general which processes that must be done to accomplish meaningful results in the next task, for example, if there has not been made any new transactions in the last 3 months the system will however generate 3 months of logs and so on, but it should also prompt the user to makes sure that the user knows that nothings has been inserted the last 3 months, and that the system therefore has nothing interesting to show.

3.3 Specified Use Cases

Use Case 1

Use case name	UpdateWineSelection
Participating actors	Initiated by Administrator Communicates to Costumers
Flow of events	<ol style="list-style-type: none"> 1. The administrator have a new vine and wishes to implement it into the database. 2. ADMIN responds by presenting a form to the administrator 3. The administrator fills out the form by selecting vine type, year, origin, price and so on. After filling out the form it is submitted 4. ADMIN receives the form and updates the database, CostumerView and BudgetView
Entry condition	The administrator is logged into ADMIN
Exit condition	New wine is implemented in the Wine Selection
Quality requirements	Customers do not have access to BudgetView and UpdateWineSelection

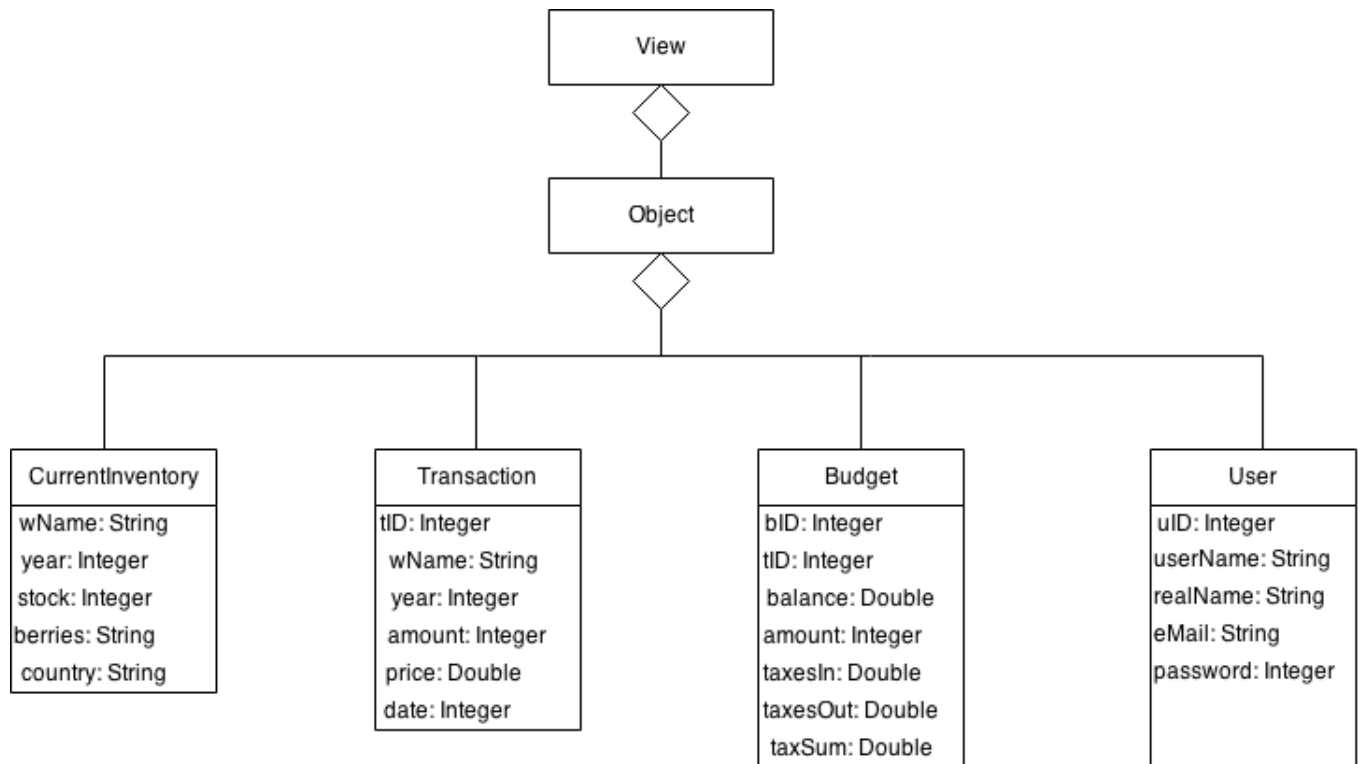
Use Case 2

Use case name	SKATYearlyReport
Participating actors	Initiated by Administrator Communicates to Administrators email
Flow of events	1. ADMIN sends an email to administrator that it is time for the yearly rapport to SKAT 2. Administrator opens Economy and finds necessary information to SKAT 3. Administrator takes the information out from the site, and sends it into SKAT
Entry condition	The administrator is logged into ADMIN Then administrator is logged into his email
Exit condition	The information has been checked and sent to SKAT
Quality requirements	Customers do not have access to BudgetView and UpdateWineSelection

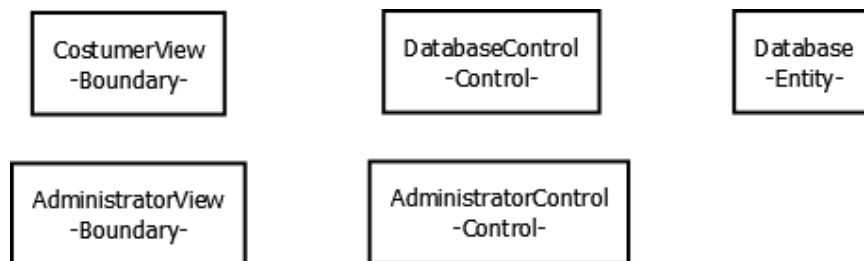
Use Case 3

Use case name	CostumerOrdersWine
Participating actors	Initiated by Costumer Communicates to Falkandersenvine.dk and Administrator
Flow of events	1. Costumer enters Falkandersenvine.dk and seach for wine selection 2. Falkandersenvine.dk shows the present wine selection to costumer 3. Costumer chooses a wine and contact Falkandersenvine to order 4. Falkandersen receives order, sends the wine to costumer and enters payment detail into Economy
Entry condition	The costumer is logged into Falkandersenvine.dk The administrator is logged into ADMIN
Exit condition	The costumer payes and recieves orderd wine
Quality requirements	Customers do not have access to BudgetView and UpdateWineSelection

4 Class Diagram

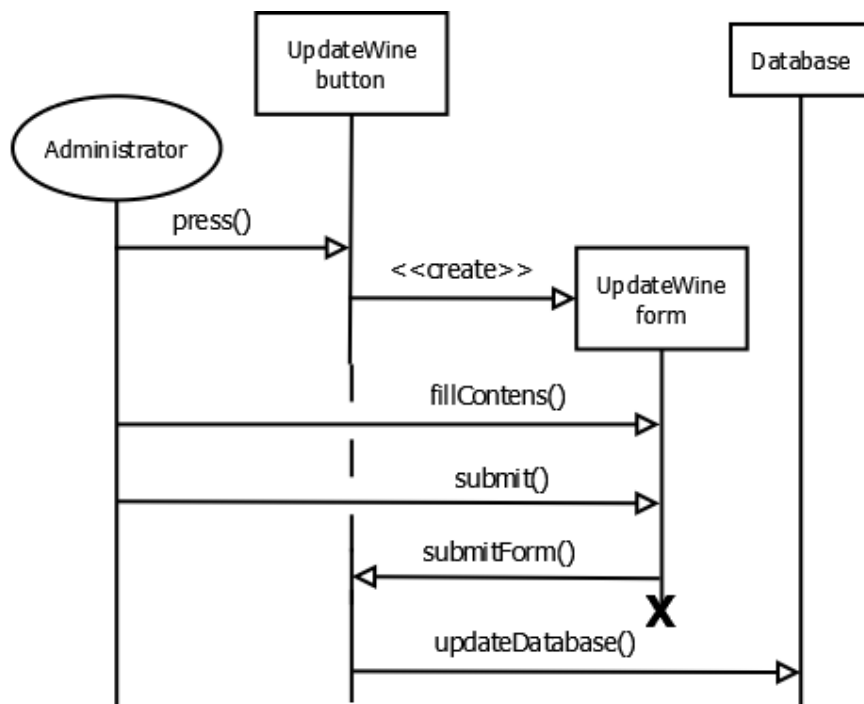


(e) En BCE-model, der viser jeres Boundary-, Control- og Entity-objekter

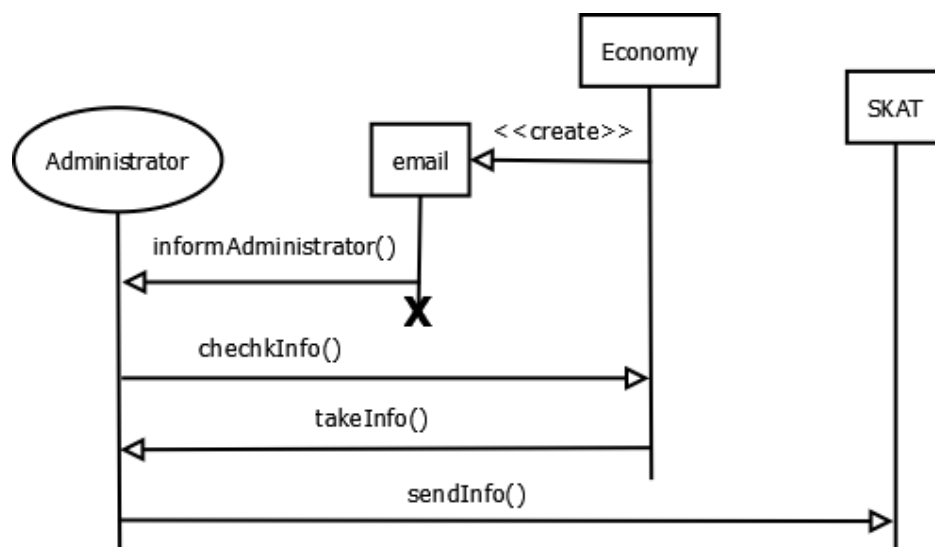


BCE model shows how there are two boundary objects, one for the costumers and one for the administrators. Two control objects, one which control the wine database and one for the economy. The database is our entity containing all information about the wine.

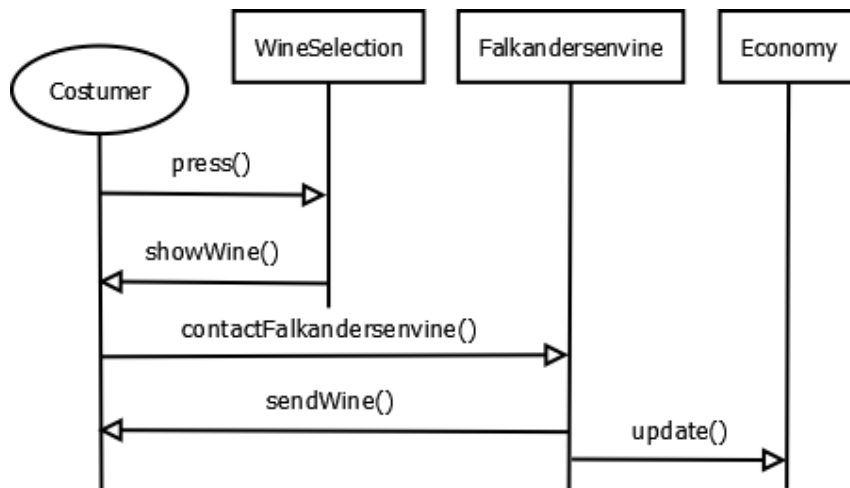
(f) Sekvens-diagrammer over de 3 use-cases specificeret i punkt (c)



The model shows how the administrator when logged into ADMIN on Falkandersenvine.dk enters the UpdateWine section, this opens a new page where the administrator inputs all the info about the new wine and submits it. UpdateWine receives this info and update the database with the new wine.



The model shows that Economy (on a preselected date) creates a new email and sends to the administrators, that is it time for the yearly report to SKAT. Administrators can then enter Economy and check everything is in order and take out the info which the administrator can now send in to SKAT.



The model shows how a costumer when logged into his personal account on Falkandersenvine.dk enters the wine selection, after looking through all the wine the costumer contacts Falkandersenvine (this is done outside the homepage) to order their wine, which Falkandersenvine then sends to the costumer. Falkandersenvine then log into thier personal account on Falkandersenvine.dk to update the economy.

5 Systemdesign

We use SQL for our database, also we intend to use the programming language PHP to design, most of the programming.

Our **database design** will look like this for now.

User(id, userName, realName, email, password(**MD5**))

CurrentInventory(wName, year, amount, berries, country)

Budget(id, transactionId, balance, amount, taxesIn, taxesOut, taxSum)

Transaction(id, wName, year, amount, price, date)

The following buttons will be the core for our project

- Insert wine into database
- Pull wine from database
- Remove wine from database
- Add a transaction
- Remove a transaction

We will need the following Views:

- Transaction(*)
- CurrentInventory(*)
- Budget(*)
- Catalog(wName, year, berries, country)

6 UI design and interaction

6.1 Illustration of different Views

Transaktioner

Menu

Du er logget ind som: Peter Jensen

Vin Navn	Oprindelse	Årgang	Illustration	Forespørg
<div><div></div></div>				

Send forespørgsel på Valgte vine

Illustrates the Transaction View from a Customer Perspective



Status

Menu

Du er logget ind som

Peter Jensen

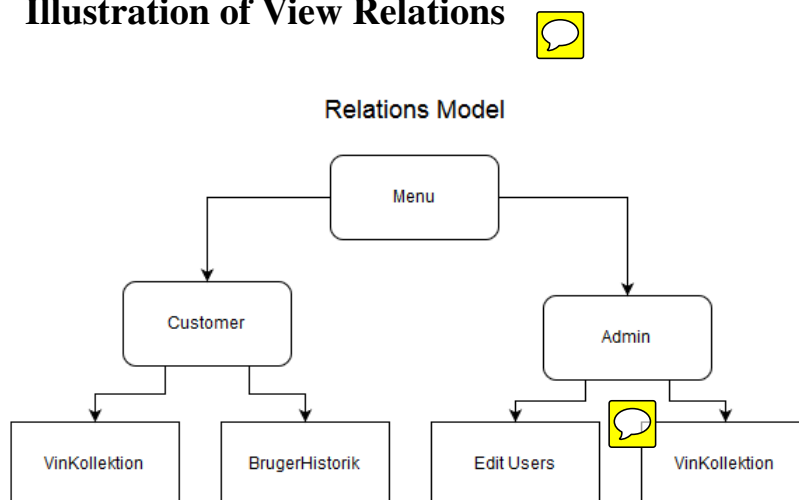
 Administrator

Vin Navn	Oprindelse	Årgang	Illustration	Status	Forespurgt af følgende brugere:	Rediger
<div></div>						

Tilføj Vin


Illustrates the **Status View** from an **Administrator** perspective

6.2 Illustration of View Relations



Illustrates the **Relations** between **Views**

7 Annex 1: VersionControl

Basic file structure has been initialized 

Following files has been edited: Index.php, Config.php, DB.php, init.php and Sanitize.php.

With these changes, we have a tiny Alpha version: The server has been setup and PHP-code is successfully connected to the database.

The project can be found at: <https://github.com/PtxDK/projekt-delta-2015>

8 Annex 2: Changelog

9 Annex 3: Timeline

First report delivered 24/03/2015

First report done and delivered on time.

First meeting with client 07/04/2015:

We had our first meeting with our clients. We agreed on the span of the project and talked some about interface and functions. The clients reception of our suggestion was very positive.

Second report delivered 22/04/2015:

Second report done and delivered on time.



Our system should be done on 31/04/2015:

We are expecting to be done, so we can focus on the presentation.