**O**ffice

**F**iling

**S**ystem

Cursist: Chris Jansen

Cursus: Programmeren 4

Schooljaar: 2014-2015

Table of Contents

[1 Application Proposal 2](#_Toc409986921)

[1.1 Introduction 2](#_Toc409986922)

[1.1.1 Motivation 2](#_Toc409986923)

[1.1.2 Problem analysis 2](#_Toc409986924)

[1.1.3 Alternative solutions 2](#_Toc409986925)

[1.2 Project structure 2](#_Toc409986926)

[1.2.1 Database 2](#_Toc409986927)

[1.2.2 Application 3](#_Toc409986928)

[1.2.3 Website 3](#_Toc409986929)

[2 Pre-Requisites 3](#_Toc409986930)

[2.1 System Requirements 3](#_Toc409986931)

[2.1.1 Windows 3](#_Toc409986932)

[2.1.2 Mac OS X 3](#_Toc409986933)

[2.1.3 Linux 4](#_Toc409986934)

[2.2 Software Requirements 4](#_Toc409986935)

[2.3 Functional Requirements 4](#_Toc409986936)

[2.4 Quality Requirements and Test Environment 4](#_Toc409986937)

[3 Analyses 5](#_Toc409986938)

[3.1 Website - User 5](#_Toc409986939)

[3.1.1 Use-Case - User 5](#_Toc409986940)

[3.1.2 Class Diagram - User 8](#_Toc409986941)

[3.1.3 User - Activity Diagram 9](#_Toc409986942)

[3.2 Application - Administrator 10](#_Toc409986943)

[3.2.1 Use-Case - Administrator 10](#_Toc409986944)

[3.2.2 Class Diagram - Administrator 12](#_Toc409986945)

[3.2.3 Component Diagram - Administrator 13](#_Toc409986946)

[3.2.4 Administrator - Activity Diagram 14](#_Toc409986947)

[3.3 Database 14](#_Toc409986948)

[3.3.1 Entity Relationship Diagram 14](#_Toc409986949)

[3.4 Programme Design 15](#_Toc409986950)

[3.4.1 Repository 15](#_Toc409986951)

[3.4.2 Used technologies 15](#_Toc409986952)

[3.4.3 Coding 15](#_Toc409986953)

[3.4.4 Testing 15](#_Toc409986954)

[3.4.5 Troubleshooting 15](#_Toc409986955)

[3.4.6 Operations 16](#_Toc409986956)

[4 Solution 16](#_Toc409986957)

[4.1 The Programme 16](#_Toc409986958)

[4.2 The website 17](#_Toc409986959)

[4.3 Achieved features 17](#_Toc409986960)

[4.4 Omitted features 17](#_Toc409986961)

[4.5 Summary 18](#_Toc409986962)

Office Filing System

# Application Proposal

## Introduction

Maintaining efficient and ordered office administration is a common problem. Depending on the office, large number of forms are generated on a daily base and keeping track of all of them can be troublesome. Development of applications enabling effective and feasible management of office files is of considerable importance. The application proposed in this study is aimed at allowing new documents to be added to a database and modified in an efficient and user-friendly manner. A Java-based application enabling administrators to manage the database will be developed, while a Java/Html-based website will allow the users to perform CRUD operations. All data will be stored on an USBWebserver.

### Motivation

*Home administration can bring a lot of paperwork or when receiving a phone Call you don’t always have the time to find your agenda or login to use online administration facilities. The purpose of this program is to make quick notes of appointments, new contacts and insert forms into a database as quickly as possible without to much hassle.*

### Problem analysis

The programmes that have the same functionality as OFS can have many drawback. Many of these applications are have ‘too” many options and even adding a new contact can be tedious to find the functions as those platforms layout over time can change a lot, this can lead to time consuming to find even the most basic options.

### Alternative solutions

There are currently many alternatives that manage administration. Some examples are Google+, Windows Agenda, Mobile phone apps and many others.

## Project structure

### Database

|  |  |
| --- | --- |
| C:\Users\CJansen\Desktop\database-design-development.jpg | For this assignment I will use a local database as a proof of concept. The database will contain 3 tables to store the necessary information.   * The first one will hold detailed information about invoices. * The second table will hold an address book. * The third table will keep track of a basic agenda. |
|  |  |

### Application

|  |  |
| --- | --- |
| C:\Users\CJansen\Desktop\software.jpg | The application will be used as a database administrative tool. The tool will provide an interface that will enable basic CRUD operations for administrators. This will allow to increase the database functionality and decrease the database maintenance time.  *Optional: If possible I would like to add a place to store images or pdf’s of the scanned invoice documents. These can either have their own file system or be included as blob data types in the database.* |

### Website

|  |  |
| --- | --- |
| C:\Users\CJansen\Desktop\website-maintenance.jpg | The website will provide an interface for a user to insert or remove invoices, addresses or appointments into/from the database.  *Optional: The website would have access to a scanner API that will automatically scan and store a digital duplicate of the invoice in the database.* |

# Pre-Requisites

## System Requirements

### Windows

* Windows 8 (Desktop)
* Windows 7
* Windows Vista SP2
* Windows Server 2008 R2 SP1 (64-bit)
* Windows Server 2012 (64-bit)
* RAM: 128 MB
* Disk space: 124 MB for JRE; 2 MB for Java Update
* Processor: Minimum Pentium 2 266 MHz processor
* Browsers: Internet Explorer 9 and above, Firefox, Chrome

### Mac OS X

* Intel-based Mac running Mac OS X 10.8.3+, 10.9+
* Administrator privileges for installation
* 64-bit browser
* A 64-bit browser (for example Safari, Firefox, or Chrome) is required to run Oracle Java on Mac OS X.

### Linux

* Oracle Linux 5.5+1
* Oracle Linux 6.x (32-bit), 6.x (64-bit)2
* Oracle Linux 7.x (64-bit)2
* Red Hat Enterprise Linux 5.5+1, 6.x (32-bit), 6.x (64-bit)2
* Ubuntu Linux 12.04 LTS, 13.x
* Suse Linux Enterprise Server 10 SP2+, 11.x
* Browsers: Firefox

## Software Requirements

* JAVA application JRE
* Browser: Firefox, Chrome, IE

## Functional Requirements

* You need to be able to add, edit and remove appointments from the agenda.
* You need to be able to add, edit and remove addresses from the database.
* You need to be able to insert, update and remove received invoices into the database.

## Quality Requirements and Test Environment

|  |  |
| --- | --- |
| Quality Requirements | Test Environment |
| * The software needs to be user-friendly. * The software needs to be fast responding. * The software input needs to be validated. * The website needs to have a login page. | * JAVA Version 8 * Glassfish 4.1 * NetBeans 8.0.2 * Windows 8.1 * USBWebserver 8.6 |

# Analyses

## Website - User

### Use-Case - User

User: The user is able to manipulate records in the database.

|  |  |
| --- | --- |
|  | |
| User - Invoices | |
| Title: | Add Invoice |
| Actor: | User |
| Description: | The user can add a new invoice to the system |
| Flow: | * The user fills in the invoice form with the invoice details * The system validates the invoice * The system adds the invoice to the database |
| Pre-conditions: | * The user is logged in * The invoice does not exist yet |
| Exceptions: | * The invoice already exists in the database * The form is filled out wrong * The database is in use |
| Post-conditions: | The invoice is added to the database |

|  |  |
| --- | --- |
| Title: | Update Invoice |
| Actor: | User |
| Description: | The user can update an existing invoice to the system |
| Flow: | * The user queries an invoice * The user edits the invoice form * The system validates the invoice * The system updates the invoice in the database |
| Pre-conditions: | * The user is logged in * The invoice does exist |
| Exceptions: | * The invoice cannot be found * The form is filled out wrong * The database is in use |
| Post-conditions: | The invoice is updated in the database |

|  |  |
| --- | --- |
| Title: | Remove Invoice |
| Actor: | User |
| Description: | The user can remove an invoice from the system |
| Flow: | * The user queries an invoice * The system validates the query * The system removes the invoice from the database |
| Pre-conditions: | * The user is logged in * The invoice does exist |
| Exceptions: | * The invoice cannot be found in the database * The query is filled out wrong * The database is in use |
| Post-conditions: | The invoice is removed from the database |

|  |  |
| --- | --- |
| User - Address book | |
| Title: | Add Address |
| Actor: | User |
| Description: | The user can add a new address to the system |
| Flow: | * The user fills in the address form with the address details * The system validates the address * The system adds the address to the database |
| Pre-conditions: | * The user is logged in * The address does not exist yet |
| Exceptions: | * The address already exists in the database * The form is filled out wrong * The database is in use |
| Post-conditions: | The address is added to the database |

|  |  |
| --- | --- |
| Title: | Update Address |
| Actor: | User |
| Description: | The user can update an existing address to the system |
| Flow: | * The user queries an address * The user edits the address form * The system validates the address * The system updates the address in the database |
| Pre-conditions: | * The user is logged in * The address does exist |
| Exceptions: | * The address cannot be found * The form is filled out wrong * The database is in use |
| Post-conditions: | The address is updated in the database |

|  |  |
| --- | --- |
| Title: | Remove Address |
| Actor: | User |
| Description: | The user can remove an address from the system |
| Flow: | * The user queries an address * The system validates the query * The system removes the address from the database |
| Pre-conditions: | * The user is logged in * The address does exist |
| Exceptions: | * The address cannot be found in the database * The query is filled out wrong * The database is in use |
| Post-conditions: | The address is removed from the database |
|  |  |
| User - Appointments | |
| Title: | Add Appointment |
| Actor: | User |
| Description: | The user can add a new appointment to the system |
| Flow: | * The user fills in the appointment form with the appointment details * The system validates the appointment * The system adds the appointment to the database |
| Pre-conditions: | * The user is logged in * The appointment does not exist yet |
| Exceptions: | * The appointment already exists in the database * The form is filled out wrong * The database is in use |
| Post-conditions: | The appointment is added to the database |
|  |  |
| Title: | Update Appointment |
| Actor: | User |
| Description: | The user can update an existing appointment to the system |
| Flow: | * The user queries an appointment * The user edits the appointment form * The system validates the appointment * The system updates the appointment in the database |
| Pre-conditions: | * The user is logged in * The appointment does exist |
| Exceptions: | * The appointment cannot be found * The form is filled out wrong * The database is in use |
| Post-conditions: | The appointment is updated in the database |

|  |  |
| --- | --- |
| Title: | Remove Appointment |
| Actor: | User |
| Description: | The user can remove an appointment from the system |
| Flow: | * The user queries an appointment * The system validates the query * The system removes the appointment from the database |
| Pre-conditions: | * The user is logged in * The appointment does exist |
| Exceptions: | * The appointment cannot be found in the database * The query is filled out wrong * The database is in use |
| Post-conditions: | The appointment is removed from the database |

### Class Diagram - User

|  |
| --- |
| * These tables show the classes that are the fundaments of the OFS website architecture |
|  |

### User - Activity Diagram

|  |
| --- |
| This diagram shows the activity process of the User |
|  |

## Application - Administrator

### Use-Case - Administrator

Administrator: The Administrator can manipulate multiple records in the database

|  |  |  |
| --- | --- | --- |
|  | | |
| Administrator - Invoices | |
| Title: | Remove Invoice |
| Actor: | Administrator |
| Description: | The administrator can remove multiple invoices from the system |
| Flow: | * The administrator selects the invoices * The system validates the query * The system removes the invoices from the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The invoice cannot be found in the database * The query is filled out wrong * The database is in use |
| Post-conditions: | The invoices are removed from the database |

|  |  |
| --- | --- |
| Title: | Summary Invoices |
| Actor: | Administrator |
| Description: | The administrator can see an overview of all recent Invoices in the database |
| Flow: | * The administrator selects a start date and end date * The system queries for all invoices between those dates * The system displays the invoices found in the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The dates in the query are filled out wrong * The database is in use |
| Post-conditions: | The invoices are displayed from the database |

|  |  |
| --- | --- |
| Title: | Remove Address |
| Actor: | Administrator |
| Description: | The administrator can remove multiple addresses from the system |
| Flow: | * The administrator selects the address * The system validates the query * The system removes the address from the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The address cannot be found in the database * The query is filled out wrong * The database is in use |
| Post-conditions: | The addresses are removed from the database |

|  |  |
| --- | --- |
| Title: | Summary Address |
| Actor: | Administrator |
| Description: | The administrator can see an overview of all recent addresses added to the database |
| Flow: | * The administrator selects a start date and end date * The system queries for all addresses between those dates * The system displays the address found in the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The dates in the query are filled out wrong * The database is in use |
| Post -conditions: | The addresses are displayed from the database |

|  |  |
| --- | --- |
| Title: | Remove Appointment |
| Actor: | Administrator |
| Description: | The administrator can remove multiple appointments from the system |
| Flow: | * The administrator selects the appointments * The system validates the query * The system removes the appointments from the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The query is filled out wrong * The database is in use |
| Post-conditions: | The appointments are removed from the database |

|  |  |
| --- | --- |
| Title: | Summary Appointments |
| Actor: | Administrator |
| Description: | The administrator can see an overview of all recent appointments added to the database |
| Flow: | * The administrator selects a start date and end date * The system queries for all appointments between those dates * The system displays the address found in the database |
| Pre-conditions: | * The administrator is logged in |
| Exceptions: | * The dates in the query are filled out wrong * The database is in use |
| Post-conditions: | The appointments are displayed from the database |

### Class Diagram - Administrator

|  |
| --- |
| These tables show the classes that are the fundaments of the OFS programme architecture |
|  |

### Component Diagram - Administrator

|  |
| --- |
| This diagram shows the component tiers of the OFS programme |
|  |

### Administrator - Activity Diagram

|  |
| --- |
| This diagram shows the activity process of the Administrator |
|  |

## Database

### Entity Relationship Diagram

|  |
| --- |
| These tables show the database architecture and its parameter types |
|  |

## Programme Design

The application has 5 sections.

|  |  |  |
| --- | --- | --- |
| Section | Type | Description |
| User Interface | Form | The User Interface will allow the user to manipulate data in the database. |
| Business Object | Class | The Business Objects will hold a set of attributes and will associate them with other business objects. These objects form the business relationships. |
| Business Object Service | Class | The Business Object Services enable much easier communication between the components and the database by allowing the code to be loosely coupled. |
| DAL | Class | The DAL of the programme will allow data access to the database. |
| Main | Class | The Main class is the application entry point. This class will indicate which files of the programme will be executed first. |

### Repository

* GitHub = https://github.com/Naktafoly/OFS.git

### Used technologies

* The desktop programme is made with JPA.
* The website is made with JSP.

### Coding

* The website will be written in html/Java. This will enable Users to complete CRUD operations with ease.
* The programme will be written in Java. This will allow Administrators to maintain and test the database.
* The database will be based on the open source USBWebserver 8.6. This server uses MySQL framework and a PHP-Admin web interface to manage it.

### Testing

Unit tests will test the programmes CRUD operations and will help Administrators maintain the database. This will allow faster debugging of code and help evolve the application in the future.

### Troubleshooting

* USBWebserver
  + Port conflict 3306-3307
  + Foreign keys in USBWebserver are dodgy
* Desktop programme
  + JPA - For some reason netbeans kept removing eclipse Link library
* Website
  + Linking and displaying the right collumns to the database is very tricky, I didn’t get it to work yet before the deadline though I hope still manage to do so.
* Dropbox
  + Dropbox caused many issues removing files from the project or not updating the documentations. This brought a lot of work rewriting the documentation and the code.

### Operations

After being tested the programme will be ready for use. The Users and Administrators will be able to do CRUD operations on the OFS database.

# Solution

## The Programme

The programme itself has a very user-friendly interface. The focus of the design is set on simplicity. Allowing anyone to use it without to much effort and all functions can be used without to much effort.

|  |  |  |
| --- | --- | --- |
| Main menu |  | The main menu contains the main navigation between the 3 programmes allowing easy access to it’s functions. |
| Addressbook |  | The Addressbook allows easy access to SCRUD operations of contacts. |
| Appointments |  | The appointment application allows appointments to be added swiftly. The button menu allows to fast editing of appointments. This programme has a list displayed at the bottom of the page containing upcoming appointments in the near future. |
| Invoices |  | The application for the invoices has the same SCRUD functionality as the previous explained applications. It also features a scan function to allow the forms of the invoices to be saved in the database as BLOB objects. |

## The website

The website is similar to the desktop app. This will allow users to access the programmes from anywhere.

|  |  |  |
| --- | --- | --- |
| Login screen |  | The users logs in to the website to access the addressbook, invoice and appointment manager. Users are registred by the java desktop applications. This feature has been omitted because of time constraints. |
| Main menu |  | Once the user is logged in the user can make a selection to start one of the 3 applications. |
| Addressbook |  | The address book the user can use CRUD operations to manage the contacts. |
| Appointments |  | The user can register new appointments in the database. Only contacts that are in the address book can be assigned. The bottom list box gives a list of the upcoming appointments. |
| Invoices |  | The invoices application is also depended on the address book and has the same CRUD functionality as the other applications. The list box below shows the unpaid invoices. |

## Achieved features

* Java application to manage records from the database.
* Website application to quickly apply new information.

## Omitted features

* The login screen has been removed due time constraints.
* The scan function has been removed due time constraints.
* The CSS of the website is not done due time constraints.

## Summary

Working on the OFS programme was an interesting experience, from debugging to many encounters with Murphy’s law. Developing a programme that looks very simple can still be a very time consuming endeavour. The biggest time sinks are the debugging and the CSS. The harvest of developing an applications is to stay on track with the building plan of the code. It’s very easy to get distracted do a bit of code here and a bit of code there with as result in spaghetti code. The worst experience of this code project was the data loss that happened several times. I started the project with hibernate but after encountering many debugging so I switched to JPA. The project made me realise that even knowing languages like C# and asp.net, working with JAVA brings a lot of its own problems and tricks on to solve things. This project thought me more about constructive programming and time management.