IT UNIVERSITY OF COPENHAGEN

BDSA 2014

# Calendar System
# System Design Document

ASSIGNMENT 39

Anders Wind Steffensen - awis@itu.dk
Christopher Blundell - ppma@itu.dk
Pierre Mandas - cnbl@itu.dk

October 2, 2014

# Contents

# Chapter 1
# Introduction

## 1.1 Purpose of the system

The purpose of the Calendar system is to have a standard client-server calendar system like MS Exchange or iCal. The system must provide the user with the ability to save, edit events and be able to retrieve their calendar on any computer with the system installed. The system should furthermore provide the user with multiple kinds of overviews of the events he has planned, and prompt the user with a notification if an event has a notification time.

## 1.2 Design goals

The goal with the systems usability, is to end up with a product which is user-friendly for all types of users. By applying gestalt laws, the goal is to end up with a clean and simple user interface which is logical for all users, even the unexperienced.
Furthermore it is desired to create a reliable program, preventing users from losing all progress related to unexpected crashes. This will be handled by frequently auto-saving data and synchronizing with the data storage. Furthermore every time the user commits anything, for example a new calendar entry, it will also be saved immediately. Force restarting should not be acceptable and most exceptions must be caught and handled runtime. By doing the abovementioned, data loss will be kept to a minimum by limiting it to users current activity, should a failure occur.
The goal performance-wise is to be able to handle a large number of events daily, without setting a noticable strain on the program. Heavy operations must run in the background, and therefore not disturbing the user while operating the program. The trade-off however will be the loading time of the program. This allows us to load all the initially required data, and prevent long waiting times while operating the system.
When rolling out future updates for the system, a full re-installation of the program will be necessary. This is due to resources allocated to other more desired design goals.
Finally the system will be tested before release, but in a limited way. Key components will be tested, but due to the time frame set for this systems development thorough testing is unattainable. The Calendar is a lightweight system, and stores data on the cloud, so there spacewise a maximum of around 1gb should be achievable. Additionally, in it's current form the CalendarSystem is only runnable on Windows OS

## 1.3 Definitions, acronyms and abbreviations

*None at the time.*

## 1.4 References

*None at the time.*

## 1.5 Overview

CHAPTER 2

# Current Software Architecture

CHAPTER 3

# Proposed Software Architecture

## 3.1 Overview

**View subsystem:** This subsystem contains the different types of views that will show, depending on how the user is using the system. Therefore, its assignment of functionality will be that this subsystem has to keep track of the different views.

**Model subsystem:** This subsystem contains the data that the system will be using. When the model changes state, it will notify the controllers, and the controllers will then change the view. Therefore, this subsystems assignment of functionality will be notifying the controllers, when its state is being changed.

**Controller subsystem:** This subsystem is being used to update the views, by receiving the models state. Its assignment of functionality will be that this subsystem will have to update the views with the received models state.

**DataStorage subsystem:** This subsystem contains a database storage. Users gain accesss to this database storage through an interface. Therefore, its assignment of functionality of this subsystem will be to store data, which will be used at a later time.

## 3.2 Subsystem decomposition

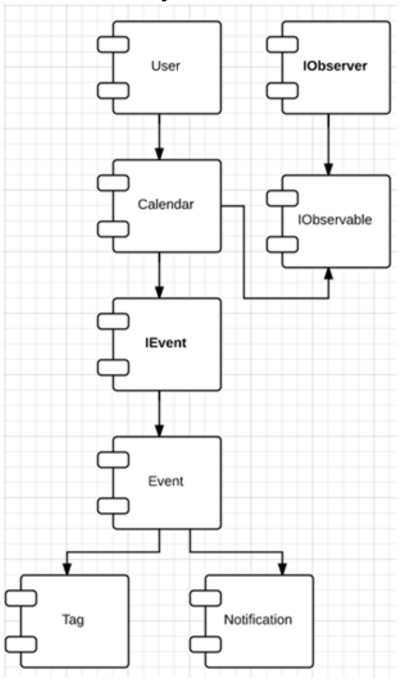
We have chosen to use the MVC design pattern, to make a clear distinction of what functionality should be assigned to which part of our system. The MVC pattern stands for Model-View-Control, and it's commonly used for implementing user interfaces.

We have also chosen to create a database, whereas this database will be used to store persistent data, to when the user will logout or close the program. This will make it possible to receive earlier used data.

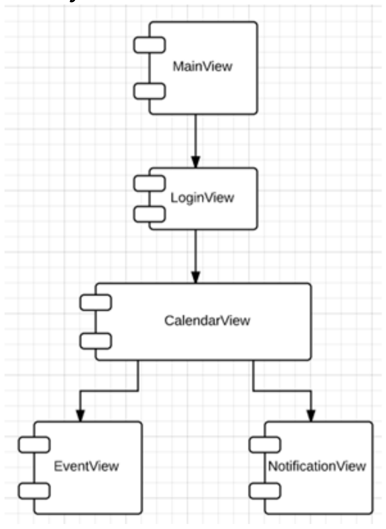
As we have chosen to use the MVC pattern, we will have 3 different subsystems, plus the subsystem of our database storage:

- Model
- View
- Control
- DataStorage

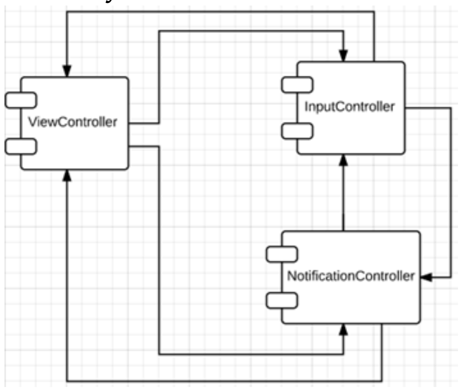
Our **Model subsystem** has the responsibility to keep notifying the controllers to update the views, which the user is using. Here is a picture of our subsystem:



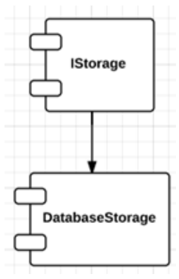
Our **View subsystem** has the responsibility to represent the model's state. Whenever the state of the model is changed, the view will also be changed to represent the new state of the model. Here is a picture of our subsystem:



Our **Control subsystem** has the responsibility to update the view, whenever the model has changed. The controllers will be used to update the views, by being notified by the model about its state. Here is a picture of subsystem:



Our **DataStorage subsystem** has the responsibility to save persistent data, which the system will be using at another time. Persistent data could be a calendar for a specific user. When the user will be login into the system, the controllers will be using the DataStorage subsystem to receive the calendar. Here is a picture of our subsystem:



## 3.3 Hardware/software mapping

## 3.4 Persistent data managemen

## 3.5 Access control and security

## 3.6 Global software control

## 3.7 Boundary conditions

CHAPTER 4

# Subsystem Services

CHAPTER 5

# Glossary