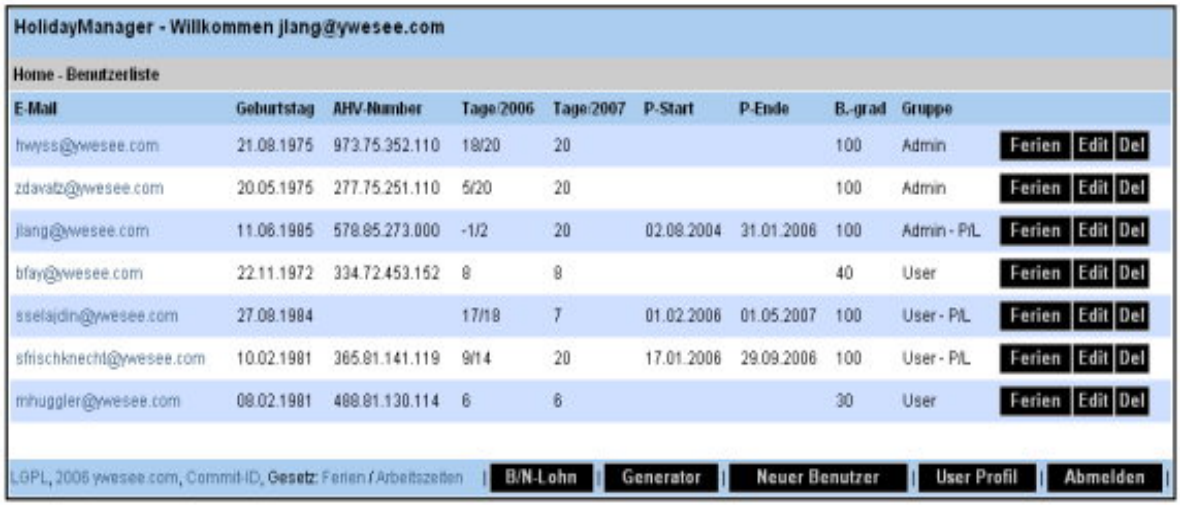
ywesee
intellectual capital connected

# Facharbeit

## Implementation und Integration eines Systems zur Bewilligung von Freitagen im internen Ferien-Management-Tool





**Verfasser:** **John Lang**

**Fachvorgesetzter:** **Hannes Wyss**
**Fachexperte:** **Daniel Boxhammer**


**Erstellt am:** **24. Februar 2006**

## Inhaltsverzeichnis

# 1. Umfeld und Ablauf

## 1.1. Detaillierte Aufgabenstellung

Passe den Ferien-Eingabe-Ablauf so an, dass Ferientage vom Arbeitgeber formell bewilligt werden können.

- Ferien (class Holiday) haben neu einen Bewilligungsstatus
- Ferien müssen neu editiert werden können.
- Benutzerübersicht: ändern die Spalten Tage/<dieses Jahr> und Tage/<nächstes Jahr>. Aktuell: '<unverbrauchte Ferientage>/<total verfügbare Ferientage>'. Neu: wenn alle Ferien bewilligt sind: '<unverbrauchte Ferientage>/<total verfügbare Ferientage>', wenn noch Ferientage zur Bewilligung ausstehen: '<unverbrauchte Ferientage> <ausstehende Ferientage>/<total verfügbare Ferientage>'
- Ferienübersicht: neue Spalte 'Bewilligt' (Ja/Nein)
- Erstellen neue Ferien: nach erfolgreichem Speichern wird automatisch ein Email an alle Administratoren verschickt, mit einem Hinweis auf den neuen Eintrag sowie einem Link. Dieser führt direkt zur Bearbeitungs-Maske für den Ferieneintrag falls der Administrator bereits eingeloggt ist. Andernfalls führt er zur Login-Maske; nach erfolgreichem Login wird der Administrator wiederum direkt zur Bearbeitungs-Maske geführt (Transparentes Login).
- Die Bearbeitungs-Maske für einen Ferieneintrag erlaubt dem Administrator, den Bewilligungsstatus des Eintrags zu verändern. Für den normalen Benutzer darf dies aber selbstverständlich nicht möglich sein.

Evaluiere für das Transparente Login zwei Implementationsansätze (es gibt mindestens drei Möglichkeiten). Beantworte dabei insbesondere für jeden Ansatz folgende Fragen: Wie wird der Zielpunkt (die Bearbeitungs-Maske) über zwei oder mehr HTTP-Requests 'im Auge behalten'? Kann die Bearbeitungsmaske auch nach einer Fehleingabe beim Login noch erreicht werden? Kann der Code für andere geschützte Seiten innerhalb des HolidayManagers verwendet werden?

Implementiere die Erweiterung nach "Test driven Design"-Regeln. Schreibe also vor jedem Programmier-Schritt einen Unit-Test. Überprüfe dass der Test _nicht_ besteht (also einen Fall testet, der noch nicht korrekt implementiert ist), und mache dich dann an die Implementation, bis der Test besteht. Insbesondere stelle sicher, dass die Transparente Login-Funktion für alle denkbaren Situationen (Nicht eingeloggt, Eingeloggt als normaler Benutzer, Eingeloggt als Administrator) korrekt funktioniert. Dokumentiere Integrations- und Acceptance-Tests unabhängig von den Unit-Tests. Die Dokumentation ist bei eXtreme-Programming im Code inhärent - achte deshalb darauf, die Namensvergabe für Klassen, Variablen und Methoden so zu gestalten, dass der Code für (Weiter-)Entwickler selbsterklärend wird, dass also anhand des Namens einer Methode darauf geschlossen werden kann, für was die Methode verantwortlich ist.

## 1.2. Vorkenntnisse

Hauptverantwortlicher für Design und Entwicklung der HolidayManager Applikation. Ruby als Hauptsächliche Programmiersprache während der gesamten Praktikumzeit verwendet.

Das Test Unit Framework habe ich in der HolidayManager Applikation nicht sehr viel gebraucht. Routine war eine Aufgabe in Form von einem Kurzen Text von meinem Arbeitgeber zu bekommen und diese dann schnell und selbständig zu lösen und programmieren.

## 1.3. Vorarbeiten

Die HolidayManager Applikation wurde während der Praktikumzeit entwickelt und wird auf dem Produktionsserver eingesetzt.

- http://holiday.ywesee.com

## 1.4. Firmenstandards

ywesee -- intellectual capital connected

### 1.4.1. Programmierumgebung

Entwickelt wird bei ywesee mit der Gentoo Linux Distribution (gentoo.org).
Als Programmierplattform dient der Konsolenbasierende VIM Editor (vim.org). VIM unterstützt Syntax highliting und syntax folding.

### 1.4.2. Programmiersprache Ruby

Als Programmiersprache dient bei ywesee Ruby. Ruby ist eine interpretierte Programmiersprache, die vor allem in ihrem Ursprungsland Japan verbreitet ist. Ruby besticht durch seine Einfachheit und die klare Syntax, was vor allem der Tatsache zu verdanken ist, dass es eine komplett objektorientierte Sprache ist. Trotz allem ist Ruby sehr mächtig und vielseitig einsetzbar. Von der Konsolenanwendung über graphische Anwendungen mit einer Vielzahl von Toolkits bis hin zu Netzwerk- und Webapplikationen lässt sich vieles sehr einfach realisieren.

### 1.4.3. Filesystem

ywesee orientiert sich bezüglich Filesystems am Ruby-Standard:

| Projektname | Pfad | Bemerkungen |
|---|---|---|
| | /bin/executable-files | (executables können auch ruby-files sein) |
| | /ext/require-pfad/c-extension-files | |
| | /lib/require-pfad/library-files | (alle Ruby-Files enden mit '.rb') |
| | /test/test-files | (alle Test-Files beginnen mit 'test_') |

Zusätzlich möglich:

| Projektname | Pfad | Bemerkungen |
|---|---|---|
| | /src/code-files | (Applikations-Code, der nicht als Library verwendet werden kann) |
| | /src/subsystem-name/code-files | (bei grösseren Projekten) |
| | /test/'test_' + subsystem-name/test-files | (bei grösseren Projekten, Files heissen gleich wie Code-Files) |
| | /doc/ | (Document-Root bei Webprojekten) |
| | resources | (statische Web-Resources, z.B. CSS-Files, Images etc.) |
| | /etc/konfigurations-files | |

Begründung:

- Standard-Compliance macht es z.B. einfacher ein Installations-Script zu erstellen.

### 1.4.4. File-Namen

Grundsätzlich gilt:

- 1 File pro implementierter Klasse (mehrere Klassen pro File nur in Ausnahmefällen, z.B. für Subklassen mit geringer Behaviordifferenz)
- File.name = Class.name.downcase + .rb
- TestFile.name = 'test_' + File.name

Ausnahme:
Bei grösseren Projekten, bei denen der Code in Subsystem-Directories abgelegt ist, beginnen Subsystem-Test-Directories mit 'test_'. Dann gilt TestFile.name = File.name

Begründung:

- Ruby erzwingt keinerlei Beschränkungen für die Nomenklatur. Trotzdem ist es für das Verständnis des Code-Lesers sinnvoll, schon im Filesystem eine Verbindung zwischen Code und Tests zu sehen.
- Um die require-Pfade eindeutig zu halten, haben Test-Files oder Subsystem-Test-Directories das Präfix 'test_'

### 1.4.5. Code-Formatierung

Klassen: CamelCase

```
class CodeStandardPolice
```

Konstanten: UPPERCASE, underscores für Worttrennung

```
DOCUMENT_AUTHOR = 'hwyss'
```

Methoden und Variablen: lowercase, underscores für Worttrennung

```
def example_method()
my_var = 10
```

Konditions-Ausdrücke, Methoden-Argumente (Definition und Aufruf) immer in Klammern:

```
if(foo == bar)
def my_method(arg1, arg2)
my_object.my_method('foo', 'bar')
```

Abstand vor und nach Operators:

```
1 + 2
rule_number = 15
```

Abstand nach Komma:

```
my_object.my_method('foo', 'bar', 'baz')
```

Block-Definitionen: Geschweifte Klammern (Abstand vorher), Abstand vor und nach der Block-Argument-Definition:

```
hash.each { |key, value| printf("%i => %s", key, value) }
```

Mehrzeilige Block-Definitionen: Einrückung um ein Tab, Block-Argumente auf der Ursprungszeile, abschliessende geschweifte Klammer auf neuer Zeile und auf ursprünglicher Höhe.

```
hash.each { |key, value|
   printf("%i => %s", key, value)
 }
```

HashTable-Literals: Wie Block-Definitionen, bei mehrzeiligen Hash-Literals Pfeil-Syntax und Werte horizontal aligniert

```
 table = {
   1234      => 'simple',
   385636213 => 'a little more complicated',
 }
```

Lange Code-Zeilen werden umgebrochen: Immer Backslash am Ende der Zeile, auch wenn vom Syntax nicht gefordert. Folgezeilen um ein Tab eingerückt.

Bei Konditions-Ausdrücken werden logische Operatoren auf die neue Zeile genommen.

```
 @a_long_variable_name.and_a_very_long_method_name(with,
many, \
   arguments, to, boot)


 if(my_condition_variable == MY_CONDITION_CONSTANT \
   && my_condition_method(my_condition_variable))
```

Begründung:

- Lesbarkeit in erster Linie eine Frage der Konsistenz und erst in zweiter Linie einer Frage der Formatierung. Die Regeln sind so gewählt, dass das Auge genügend Whitespace zur Orientierung erhält und der Editor keine zusätzlichen Zeilenumbrüche einfügen muss.

### 1.4.6. Namensgebung

Getter-Methoden heissen gleich wie die Instanzvariable auf die sie sich beziehen:

```
class Foo
   def initialize
     @bar = 23
   end
   def bar
     @bar
   end
 end
```

Dies ist ein Beispiel. Im Production Code wird für ein reines Auslesen die 'attr_reader' Klassenmethode verwendet.

Setter-Methoden heissen gleich wie die Instanzvariable, auf die sie sich beziehen, mit dem Suffix '=':

```
class Foo
   def bar=(bar_value)
     @bar = bar_value
   end
 end
```

Dies ist ein Beispiel. Im Production Code wird für ein reines Zuweisen die 'attr_writer' Klassenmethode verwendet.

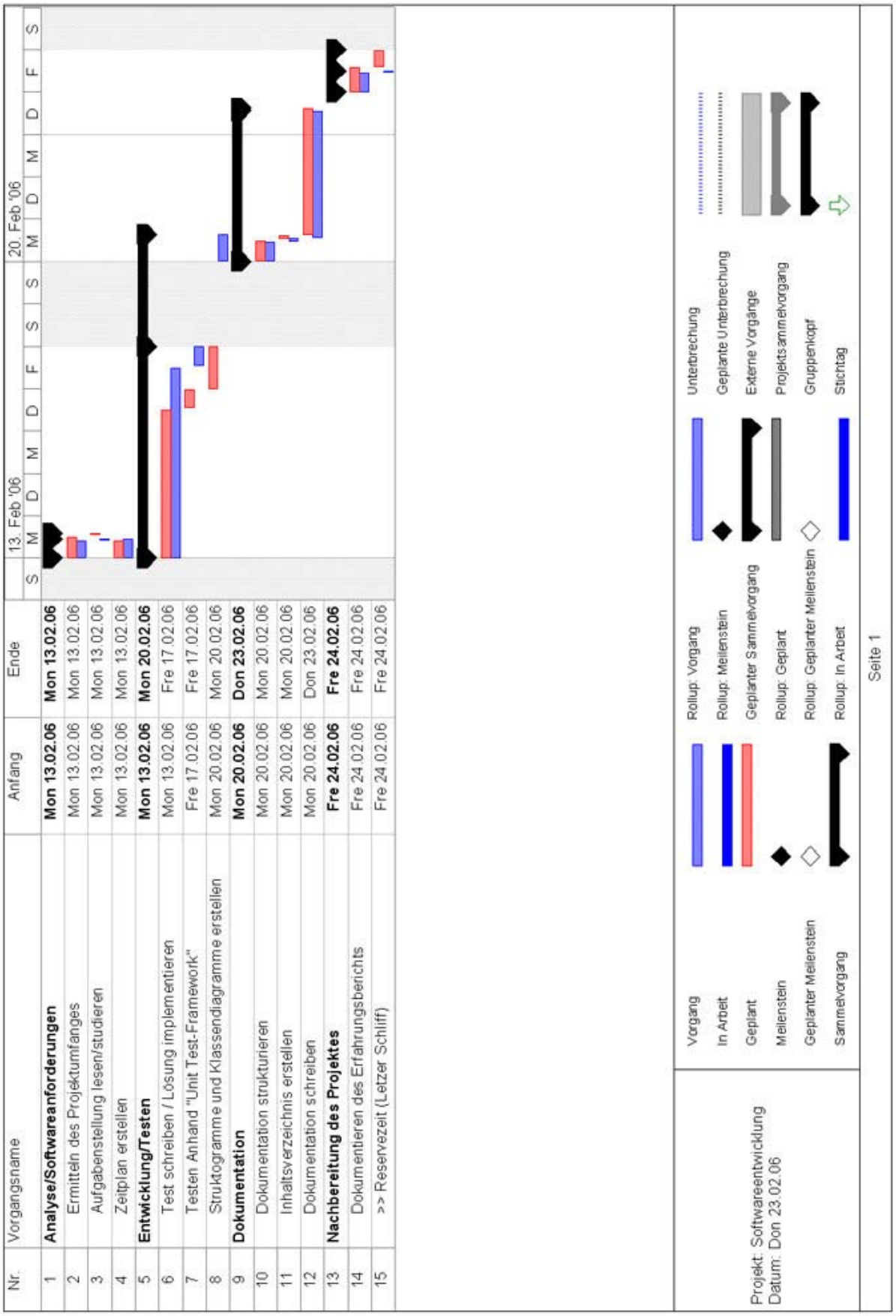
- Keine Namenswiederholung. Eine Klasse 'Person' hat eine Getter-Methode 'name' und nicht 'person_name'.
- Descriptive Names: Namen von Klassen, Methoden und Variablen beschreiben die Funktion/Responsibility des Codes in Englisch.

### 1.4.7. ExtremeProgramming Grundsätze

Ein paar der wichtigsten ExtremeProgramming Grundsätze:

- Für jede neue Funktionalität wird ein Unit-Test (und gegebenenfalls ein Integrations-Test) erstellt, bevor mit der Implementation begonnen wird.
- Do the simplest thing that could possibly work. (Die Antizipation von Kundenbedürfnissen durch den Entwickler und sogar durch den Kunden selbst führt meistens in die falsche Richtung).
- Refactor mercilessly: Code-Duplikation eliminieren, unpräzise Namensgebungen ändern.

## 1.5. Zeitplan


Nr.
Vorgangsname
Anfang
Ende
13. Feb '06
20. Feb '06
S M D M D F S S M D M D F S
1 Analyse/Softwareanforderungen Mon 13.02.06 Mon 13.02.06
2 Ermitteln des Projektumfanges Mon 13.02.06 Mon 13.02.06
3 Aufgabenstellung lesen/studieren Mon 13.02.06 Mon 13.02.06
4 Zeitplan erstellen Mon 13.02.06 Mon 13.02.06
5 Entwicklung/Testen Mon 13.02.06 Mon 20.02.06
6 Test schreiben / Lösung implementieren Mon 13.02.06 Fre 17.02.06
7 Testen Anhand "Unit Test-Framework" Fre 17.02.06 Fre 17.02.06
8 Struktogramme und Klassendiagramme erstellen Mon 20.02.06 Mon 20.02.06
9 Dokumentation Mon 20.02.06 Don 23.02.06
10 Dokumentation strukturieren Mon 20.02.06 Mon 20.02.06
11 Inhaltsverzeichnis erstellen Mon 20.02.06 Mon 20.02.06
12 Dokumentation schreiben Mon 20.02.06 Don 23.02.06
13 Nachbereitung des Projektes Fre 24.02.06 Fre 24.02.06
14 Dokumentieren des Erfahrungsberichts Fre 24.02.06 Fre 24.02.06
15 >> Reservezeit (Letzer Schliff) Fre 24.02.06 Fre 24.02.06
Projekt: Softwareentwicklung
Datum: Don 23.02.06
Vorgang
In Arbeit
Geplant
Meilenstein
Geplanter Meilenstein
Sammelvorgang
Rollup: Vorgang
Rollup: Meilenstein
Geplanter Sammelvorgang
Rollup: Geplant
Rollup: Geplanter Meilenstein
Rollup: In Arbeit
Unterbrechung
Geplante Unterbrechung
Externe Vorgänge
Projektsammelvorgang
Gruppenkopf
Stichtag
Seite 1

## 1.6. Arbeitsjournal

### Montag 13. Februar 2006

**Ausgeführte Arbeiten**
Als erstes habe ich meine Aufgabenstellung genau studiert und mir ein paar Lösungsansätze zurechtgelegt. Nachdem ich mir einen Überblick über meine IPA verschafft habe, machte ich mir einen Zeitplan. Programmiert habe ich heute den Test und der Code für den Bewilligungsstatus der in der Holiday Klasse.

**Erreichte Ziele**
Ich habe einen guten Einstieg in meine Arbeit gefunden und einen Zeitplan erstellt.

**Auftretende Probleme**
Beim Programmieren hatte ich kleinere Probleme, um nicht viel Zeit zu verlieren habe ich mir zwei bis drei Tipps von meinem Fachvorgesetzten eingeholt und konnte somit mein Problem schnell selbstständig lösen.

**Vergleich mit Zeitplan**
Beim Programmieren bin ich weiter gekommen als geplant. Das ist sehr gut so, da ich so etwas Reserve Zeit habe falls ich schwierige und Zeitstehlende Entwicklungssituationen antreffe.

### Dienstag 14. Februar 2006

**Ausgeführte Arbeiten**
Heute habe ich hauptsächlich Programmierarbeit geleistet.

**Erreichte Ziele**
Ferieneinträge können editiert und bewilligt werden. Wenn ein User einen Ferieneintrag macht geht ein Mail an den Administrator mit dem Link zur Edit Maske des neuen Ferieneintrages.

**Aufgetretene Probleme**
Falls der Administrator Ferien editiert, gibt es ein Problem beim abspeichern da der Administrator nicht der Inhaber von diesem Ferieneintrag ist. (Um einen Ferieneintrag zu speichern wird immer der ganze User gespeichert, dies passiert mit user.odba_store)

**Vergleich mit Zeitplan**
Ich liege soweit gut in der Zeit, das Problem mit dem Speichern muss ich einfach noch in den Griff bekommen.

## Mittwoch 15. Februar 2006

**Ausgeführte Arbeiten**
Heute habe ich mich mit dem Speicherungsproblem auseinandergesetzt. Dazu habe ich die get_holiday Methode umgeschrieben, diese liefert mir jetzt den Ferieneintrag wie auch den dazugehörigen User.

**Erreichte Ziele**
Meine Methode liefert mir jetzt den richtigen User um die Holidays abzuspeichern.

**Aufgetretene Probleme**
keine

**Vergleich mit Zeitplan**
Ich liege in der Zeit

## Donnerstag 16. Februar 2006

**Ausgeführte Arbeiten**
Heute habe ich die neue Spaltenformatierung in der Benutzerübersicht implementiert. Ansonsten habe ich am transparenten Login gearbeitet.

**Erreichte Ziele**
Spaltenformatierung wird in jedem Fall korrekt angezeigt (Ferien bewilligt, Ferien noch nicht bewilligt und Total Ferien zur Verfügung).

**Aufgetretene Probleme**
Methode für die Spaltenformatierung hat mir zuerst ein Problem bei Speichern von dem Bewilligungsstatus gemacht. Transparenter Login macht mir zu schaffen.

**Vergleich mit Zeitplan**
Bin nicht mehr ganz in meinem Zeitplan.

## Freitag 17. Februar 2006

**Ausgeführte Arbeiten**
Lösungsansätze für Transparenter Login.

**Erreichte Ziele**
Konnte mein Ziel „Transparenter Login“ heute leider nicht abschliessen.

**Aufgetretene Probleme**
Aus unerklärlichen Gründen konnte ich über die Session nicht auf den request path zugreifen (@session.request_path).

**Vergleich mit Zeitplan**
Ca. Halben Tag Rückstand.

## Montag 20. Februar 2006

### Ausgeführte Arbeiten
Heute habe ich noch den Transparenten Login fertig gemacht. Anschliessend habe ich mich über Klassendiagramme informiert und angefangen meine Klassen darzustellen. Mit der Dokumentation habe ich auch angefangen.

### Erreichte Ziele
Transparenter Login funktioniert. Struktur (Inhaltverzeichnis) meiner Dokumentation.

### Aufgetretene Probleme
Klassendiagramme korrekt zeichnen ist nicht meine Stärke.

### Vergleich mit Zeitplan
Mein Klassendiagramm ist noch nicht fertig. In der Dokumentation bin ich dafür weiter gekommen als geplant.

## Dienstag 21. Februar 2006

### Ausgeführte Arbeiten
An diesem Tag habe ich die Tests nochmals überarbeitet. Sonst habe ich an meiner Dokumentation weiter geschrieben.

### Erreichte Ziele
Ich bin gut vorangekommen in der Dokumentation.

### Aufgetretene Probleme
Keine.

### Vergleich mit Zeitplan
Liege gut in der Zeit

## Mittwoch 22. Februar 2006

### Ausgeführte Arbeiten
Heute habe ich hauptsächlich an meiner Dokumentation gearbeitet. An diesem Tag ist auch mein Experte vorbeigekommen und ich habe Ihm meine Arbeit vorgeführt.

### Erreichte Ziele
Ca. die Hälfte meiner Dokumentation ist erarbeitet.

### Aufgetretene Probleme
Mit dem produzieren von Diagrammen komme ich nicht so gut voran, es fehlt mir noch etwas an Praxis.

### Vergleich mit Zeitplan
Liege gut in der Zeit

**Donnerstag 23. Februar 2006**

**Ausgeführte Arbeiten**
Heute habe ich meine Diagramme fertiggestellt und an Dokumentation weiter gearbeitet.

**Erreichte Ziele**
Dokumentation ist auf gutem Weg.

**Aufgetretene Probleme**
Die Richtigkeit meiner Diagramme stelle ich in Frage, diese gehören aber auch nicht zu meinen gewohnten Methoden und Arbeitstechniken.

**Vergleich mit Zeitplan**
Liege gut in der Zeit

**Freitag 24. Februar 2006**

**Ausgeführte Arbeiten**
Habe an der Dokumentation geschrieben. Da noch etwas Zeit blieb, habe ich zusätzlich noch eine Benachrichtigungs-Methode für den Benutzer Programmiert. Diese Informiert den Benutzer per E-Mail wenn sein Ferieneintrag bewilligt worden ist.

**Erreichte Ziele**
Dokumentation Fertig gestellt und Gebunden.

**Aufgetretene Probleme**
Keine.

**Vergleich mit Zeitplan**
Mein Zeitplan ist sehr gut aufgegangen.

## 1.7. Wissenswertes zum HolidayManager

### 1.7.1. Ausgangslange

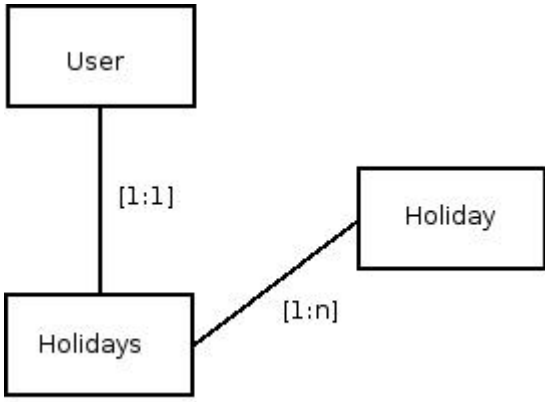
Der HolidayManager (holiday.ywesee.com) ist ein ywesee internes Ferien Management Tool das mit der Objektorientierten Programmiersprache Ruby entwickelt wurde. Das Tool ist eine Webapplikation und wird demzufolge auch via Web (Browser) bedient. In der jetzigen Situation können die Benutzer ihre Ferien einfach ohne Bestätigung eintragen. In Zukunft soll aber der gewünschte Ferieneintrag von der jeweils zuständigen Person bewilligt werden und erst dann in der Ferienübersicht erscheinen.

**Der HolidayManager wurde nach dem MVC (Model-View-Controller) Prinzip aufgebaut.**

### 1.7.2. Wie werden die Danten gespeichert?

Die Daten werden mit Hilfe der ODBA gespeichert. Die ODBA ist ein Object-Cache für in Ruby programmierte Applikationen.

Dieses ERM soll veranschaulichen wie User, Holidays und Ferieneinträge zueinander stehen.




Wird als zum Beispiel ein Holiday Objekt (Ferieneintrag) editiert, ist es notwendig das man den ganzen Benutzer wieder abspeichert. Dies wird mit @user.odba_store gemacht.

### 1.7.3. Von wo kommt das HTML?

Das HTML wird durch das HtmlGrid Framework produziert. Im View wird ausschliesslich Ruby Code geschrieben dieser wird dann von dem HtmlGrid Framework zu HTML interpretiert.

# 2. Projekt

## 2.1. Verfeinerung des Auftrages

Meine Aufgabenstellung möchte ich in zwei Teile zerlegen:
Beim ersten Teil meiner Aufgabe geht es um zwei Themen, erstens das Editieren der Ferieneinträge und zweitens um die Darstellung verschiedenen Daten.
Um Ferieneinträge zu editieren werde ich mich an dem Code um Ferientage zu erstellen orientieren. Der Unterschied besteht darin, dass beim editieren die vorhandenen Daten eingelesen und wieder abgespeichert werden. Um die Übersicht richtig darzustellen brauche ich no ein paar Methoden, diese sind aber relativ einfach zu realisieren.

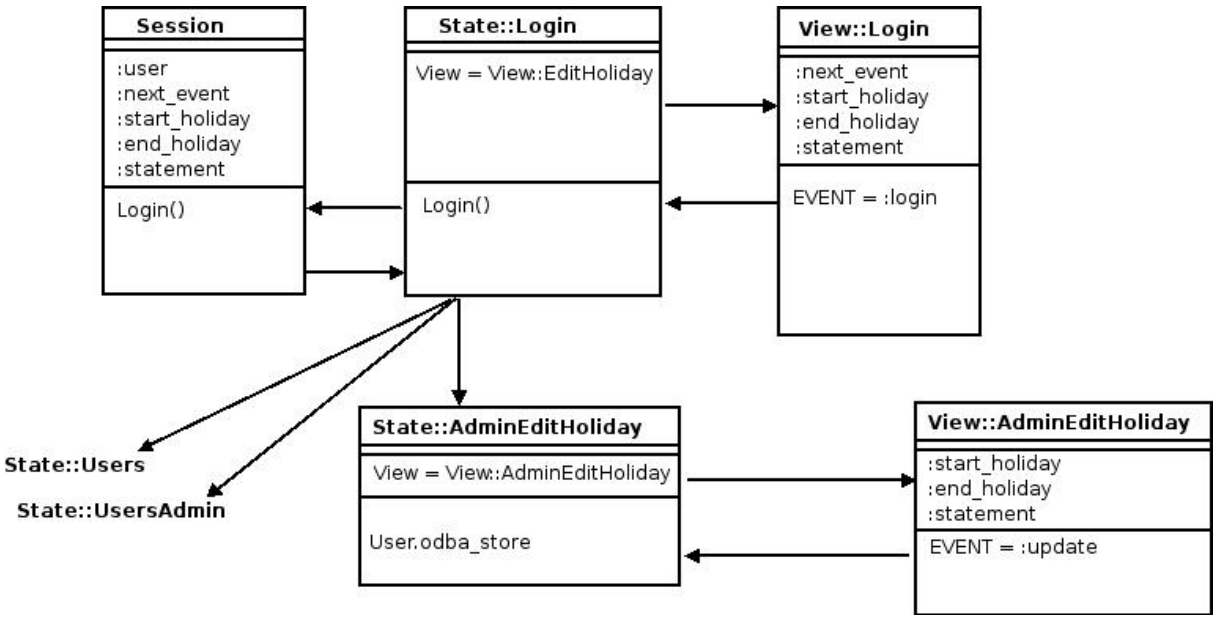
Der zweite, komplexere Teil meines Auftrages ist der Transparente Login. Das ist ein Login System das verschiedene Anfragen entgegennehmen und verarbeiten kann. In meinem Fall ist die Anfrage eine URL der zur entsprechenden Edit Maske von dem Ferieneintrag führt.

Die Verschiedenen Situationen:

| Situation | Nicht eingeloggt | Eingeloggt als normaler User | Eingeloggt als Administrator |
|---|---|---|---|
| Ziel des Links | Link führt zur Login Maske | Link führt zur Benutzerübersicht | Link führt zur Edit Maske |
| Ziel nach erfolgreichem Als Normal User Login | Führt zur Benutzerübersicht | | |
| Ziel nach erfolgreichem Als Administrator Login | Führt zur Edit Maske | | |

## 2.2. Klassendiagram Transparenter Login

Dieses Klassendiagram soll veranschaulichen wie die Klassen zueinander stehen und wie der Login funktioniert.




## 2.3. Planung

Mein Projekt wird nach dem ExtremeProgramming Ansatz durchgeführt. Bei dieser Vorgehensweise schreibt man von Anfang an Code. Die Dokumentation wird auf ein Minimum reduziert. Da ExtremeProgramming meistens mit Test-First kombiniert wird, ist ein Grossteil der Dokumentation eines Systems in den UnitTests enthalten.

## 2.4. Konzept

### 2.4.1. Anforderungen Transparenter Login

In das Konzept auch mit einfliessen sollen die Anforderungen an den Transparenten Login.

- Wie wird der Zielpunkt (die Bearbeitungs-Maske) über zwei oder mehr HTTP-Requests 'im Auge behalten'?
- Kann die Bearbeitungsmaske auch nach einer Fehleingabe beim Login noch erreicht werden?
- Kann der Code für andere geschützte Seiten innerhalb des HolidayManagers verwendet werden?

### 2.4.2. Transparenter Login Cookie Variante

In dieser Variante wird beim ersten Login ein Cookie mit den Login Informationen beim Client gespeichert. Bei einem URL Login auf eine bestimmte Seite kann sich der Client Automatisch via Cookie anmelden.
Diese Variante setzt jedoch voraus, dass sich der Client schon mal angemeldet hat und die Logindaten im Cookie gespeichert wurden.

### 2.4.3. Transparenter Login Hidden Textfield Variante

In dieser Variante wird der Session einen zusätzlichen Variable:next_event mitgeschickt. So ist es möglich in der Login Methode zu erkennen das der Benutzer über der Link der ihm via Email zugeschickt wurde auf die Seite zugreift.

### 2.4.4. Variantenvergleich und Entscheid

Die erste Variante „Cookie“ hat den Nachteil, dass der Benutzer sich schon mal eingeloggt haben muss. Ein weiterer Nachteil ist auch, falls der Benutzer seine Cookies löscht ist ein automatischer Login nicht mehr möglich.

Entschieden habe ich mich für die zweite Variante „Hidden Texfield“. Diese Variante ist auch leicht auszubauen für andere Links. Sprich in Zukunft ist es immer möglich über @session.next_event = <Traget> einen next_event zu setzten und somit den neuen State in der Login Methode zu bestimmen.

File: src/state/login.rb

```
require 'util/server'
require 'view/login'
require 'state/global_predefine'
require 'state/users'

module HolidayManager
 module State
   class Login < SBSM::State
    VIEW = View::Login
    def login
     state = self
     if(usr = @session.login)

      # Hier wird überprüft ob die Variable :next_event mit geschickt
```

```
      # wurde, falls ja wird die Variable state auf den
      # AdminEditHoliday State gesetzt
      if(@session.user_input(:next_event) == "edit_holiday" &&
        usr.group == 'group_admin')
       state = State::AdminEditHoliday.new(@session, usr)
      elsif(usr.group == 'group_admin')
       state = State::UsersAdmin.new(@session, nil)
      else
       if(usr.pass == "81dc9bdb52d04dc20036dbd8313ed055")
        state = State::UserProfile.new(@session, usr)
       else
        state = State::Users.new(@session, nil)
       end
      end
      if(mod = usr.viral_module)
       state.extend(mod)
      end
     end
     state # Hier wird der State (bzw. das nächste Ziel) zurückgegeben
    end

    # Diese Methode wird ausgeführt wenn man über den Link
    # holiday.ywesee.com/de/edit_holiday/… auf die Seite kommt.
    # Falls dies der Fall ist wird hier die Variable next_event
    # und die Userdaten in die @session geschrieben.
    def edit_holiday
     @session.start_holiday = @session.user_input(:start_holiday)
     @session.end_holiday = @session.user_input(:end_holiday)
     @session.statement = @session.user_input(:statement)
     @session.next_event = :edit_holiday
     self
    end
   end
 end
end
```

## 2.5. Realisation

Zuerst habe ich die Aufgabestellung genau studiert und die Prioritäten gesetzt. Ich habe mir mein Projekt in Kleinere Aufgabenbereiche zerlegt und diese dann Stück für Stück realisiert. In den nächsten paar Kapiteln werde ich diese Aufgabenbereiche kurz beschreiben.

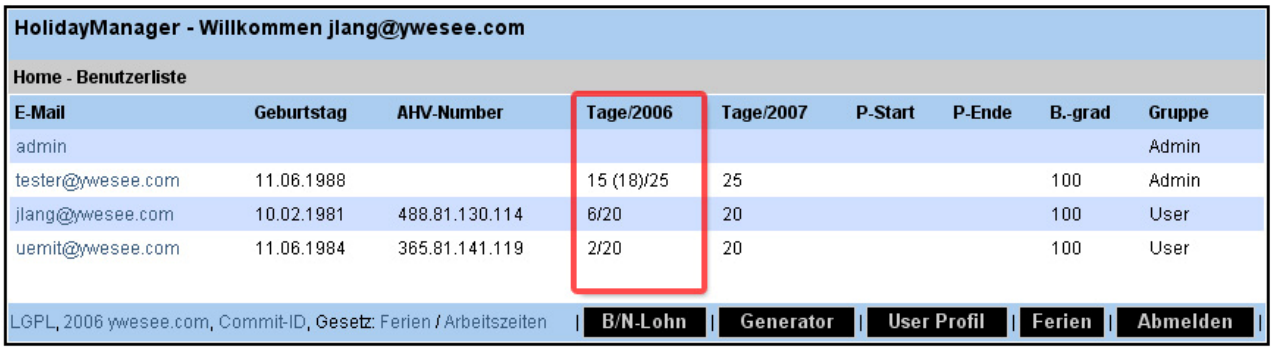
### 2.5.1. Bewilligungsstatus und Ferien Editieren

In der Holiday Klasse habe ich einen zusätzlichen attr_accessor allocation (Bewilligung) definiert. Dieser wird bei einem neuen Ferieneintrag automatisch auf ‚allocation_no' gesetzt.
Als nächstes habe ich den State und View für das Editieren von Ferieneinträgen erstellt. Ein Teil diese Prozesses konnte ich von dem State und View der zuständig ist um Ferieneinträge zu erstellen ableiten

### 2.5.2. Benutzerübersicht

In der Benutzerübersicht soll neu bewilligte und noch nicht bewilligte Ferientage ersichtlich sein. Um dies zu erreichen habe ich in dem zuständigen State zusätzlich

eine neue Methode gemacht, die mir alle Tage der noch nicht bewilligte Ferieneinträge zusammenzählt.

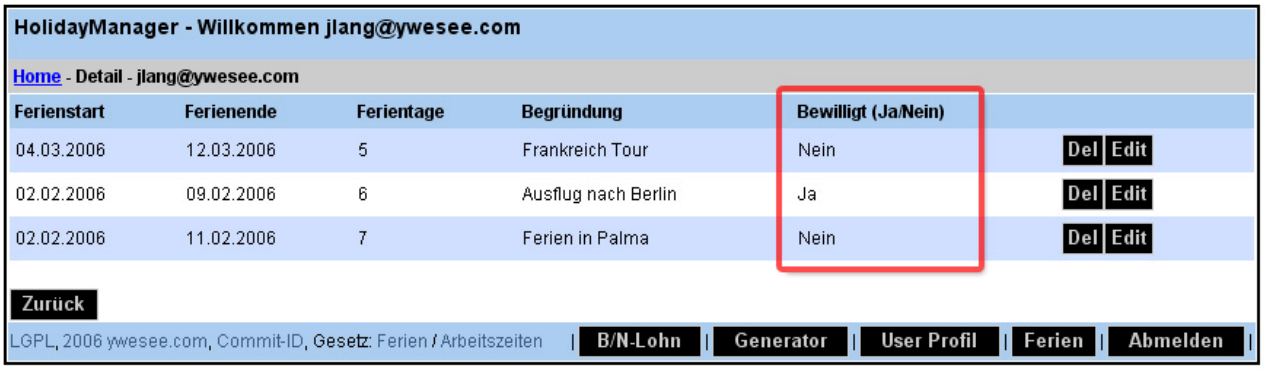



Bedeutung:

| Bewilligte Ferientage | Noch nicht bewilligte Ferientage | Von | Total Ferientage |
|---|---|---|---|
| 15 | (18) | / | 25 |

### 2.5.3. Ferienübersicht

Um die Ferienübersicht anzupassen musste ich hier nur den View bearbeiten und ein paar Wörter im lookandfeel File hinzufügen.




### 2.5.4. Automatisches E-Mail

Die Überlegung hier ist einfach: Nach erfolgreichem speichern des neuen Ferieneintrages soll ein E-Mail verschickt werden. Um dies einfach zu lösen habe ich eine neue Methode holiday_report mit den Argumenten email, start_holiday, end_holiday und statement gemacht.
Der Link der ihm Mail mitgeschickt werden soll habe ich wie folgt erstellt:

```
Lookanfeel._event_url(:edit_holiday, args)
```

Dieser Aufruf liefert mir den kompletten Link mit dem richtigen Event und den Argumenten dazu (args). Der Link führt direkt zur Bearbeitungsmaske des jeweiligen Ferieneintrages.

### 2.5.5. Transparenter Login

Den Transparenten Login habe ich nach dem oben genannten Konzept realisiert. Der springende Punkt war das man im View nur einen EVENT setzen konnte somit hatte ich das Problem wieder zu meiner Ausgangs Methode zurückzukommen. Schlussendlich habe ich das Problem so gelöst, indem ich eine Hidden Textfield im View eingebaut habe. So hatte ich die Möglichkeit dort eine Variable mitzuschicken, die mir dann in der Login Methode behilflich war den richtigen State zu finden.

### 2.5.6. Projektzusatz

Da ich noch Zeit hatte, habe ich noch eine Methode geschrieben die ein E-Mail an den User verschickt falls dessen Ferien bewilligt wurden. Die Methode wird nur dann ausgeführt wenn der Bewilligungsstatus nein ist und neu auf ja gesetzt wird.

```
If(input[:allocation] == ‚allocation_yes’ &&
  @model.allocation == ‚allocation_no’)
 holiday_allowed_report(@user.email, @model.start_holiday,
  @model.end_holiday)
end
```

## 2.6. Tests

Bei der ExtremeProgramming Technik schriebt man in den meisten Fällen zuerst einen Test. Dieser Test Testet die Erwartungen an den Code, der aber in diesem Moment noch nicht geschrieben ist. Dann wird der der Code Schritt für Schritt geschrieben, bis der Test besteht.

Hinweis: Wird zum Beispiel ein neues Feature eingebaut das an verschiedenen Orten Veränderungen mit sich zieht, kann man alle Tests laufen lausen und sieht somit wo die Tests fehlschlagen. So ist das Problem schnell lokalisiert und kann behoben werden.

### 2.6.1. Acceptancetests

Erfolgreiche Testfälle:

| Testfall | Ausgeführte Tätigkeit | Resultat |
|---|---|---|
| Login als Administrator über Link der direkt zur Edit Maske des Ferieneintrages führt. | Öffne den Link mit dem Browser der mich zur Login Maske führt. Dann Logge ich mich als Admin ein. | Wie erwartet komme ich direkt zur Edit Maske des gewünschten Ferieneintrages |
| Login als Normaler Benutzer über Link der direkt zur Edit Maske des Ferieneintrages führt. | Öffne den Link mit dem Browser der mich zur Login Maske führt. Dann Logge ich mich als User ein. | Ich habe keine Admin Rechte und komme daher einfach zur Benutzerübersicht. |

| | | |
|---|---|---|
| Kann die Bearbeitungsmaske auch nach einer Fehleingabe beim Login noch erreicht werden? | Loge mich mit falschem PW oder Benutzername ein. Beim zweiten Versuch melde ich mich mit korrekten Daten an. | Trotz des Fehllogins bin ich zum richtigen Ort gekommen, nämlich zur Edit Maske des Ferieneintrages |
| Eingeloggt als normaler Benutzer möchte ich Ferien eintragen. | Klicke auf den Ferienbutton und Trage meine Daten ein. | Administrator erhält ein Mail mit Link zum neuen Ferieneintrag. |
| Ich möchte als normaler User meine Ferien editieren. | Klicke auf den Edit Button bei dem gewünschten Ferieneintrag. | Wie erwartet soll hier kein DropDown Menu für den Bewilligungsstatus erscheinen, dieser kann nur der Admin sehen und ändern. |
| Administrator möchte Ferien Bewilligen. | Klicke auf Link, setze den Bewilligungsstatus auf ‚Ja'. Und speichere ab. | Der Benutzer wird via E-Mail benachrichtigt, dass seine Ferien bewilligt worden sind. |

## 2.7. Anwendung

Diese Anleitung setzt ein entsprechend ausgerüstetes System voraus.

### 2.7.1. Serverseitig

Um die HolidayManager Applikation zu starten muss man legendlich den Apache Webserver starten, Postgrey SQL starten und der HolidayManager starten. Dies kann man in der Konsole mit folgenden Befehlen tun.

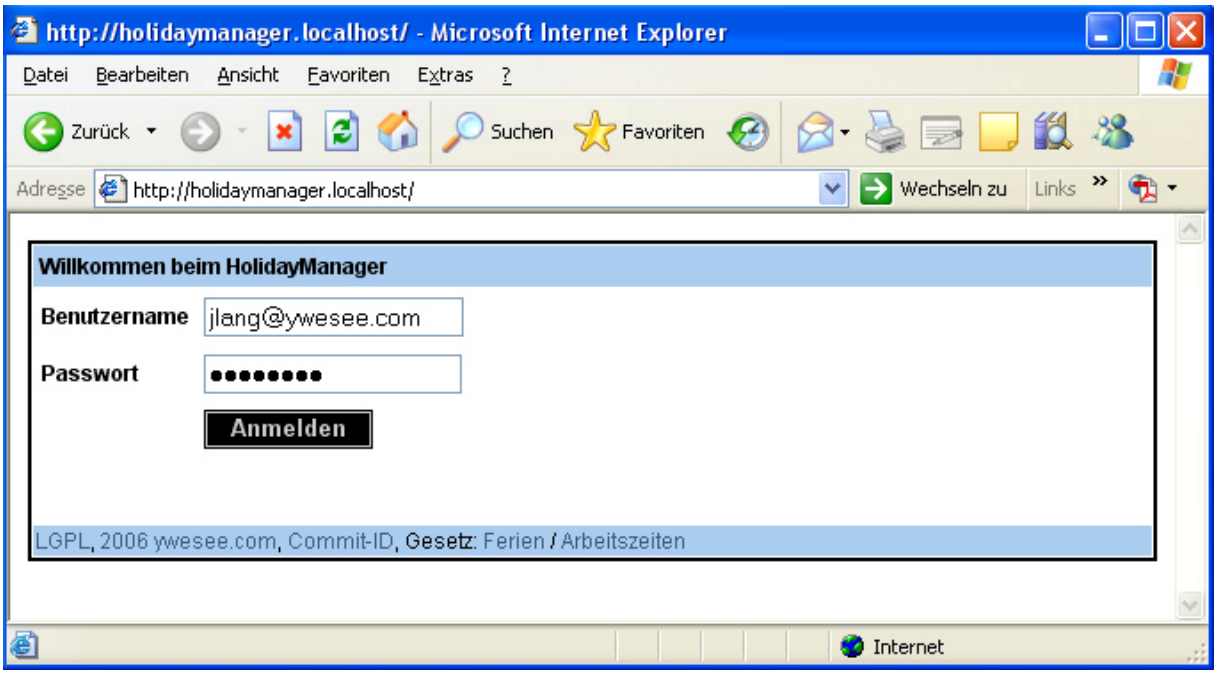
Befehle:

**Apache starten:** sudo /etc/init.d/apache start
**Postgrey SQL starten:** sudo /etc/init.d/postgrey start
**HM starten:** bin/serverd

Apache und Postgrey SQL müssen als Superuser gestartet werden.

### 2.7.2. Clientseitig

Der Benutzer kann via http://holidaymanager.localhost auf die Applikation zugreifen. Als erstes erscheint die Login Maske, der Benutzer kann sich einloggen. Ansonsten ist das der HolidayManager Clientseitig selbst erklärend.




## 2.8. Schlusswort

Dieses Projekt hat mir Spass gemacht und war eine gute Erfahrung für mich. Ich konnte einiges dazulernen in den Bereichen Projekt Management und Programmierverständnis für die objektorientierte Programmierung. Diese Aufgabe stellte interessante Herausforderungen, die mit meinem bisher erworbenen Wissen und den erlernten Instrumenten weitgehend selbständig zu lösen waren.

Ich möchte mich an dieser Stelle bei meinen beiden Begleitern durch diese Arbeit, Herrn Hannes Wyss (Fachvorgesetzter) und Herrn Daniel Boxhammer (Fachexperte) ganz herzlich bedanken für die kompetente und umsichtige Betreuung wären diesen zwei Wochen. Besten Dank.

# 3. Anhang

## 3.1. Glossar

**HiddenTextfield**
Verstecktes Textfeld

**Event**
Nächstes Ziel oder Methode die ausgeführt werden soll.

**ExtremeProgramming**
ExtremeProgramming (XP) ist ein agiler Softwareentwicklungsprozess für kleine Teams. Der Prozess ermöglicht, langlebige Software zu erstellen und während der Entwicklung auf vage und sich rasch ändernde Anforderungen zu reagieren.

**ODBA**
Framework, speichert Objekte persistent ab.

**State**
Der State verwaltet die Sichten, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und hat schreibenden Zugriff auf das Modell. Er enthält die Intelligenz und steuert den Ablauf der Anwendung.

**Model**
Das Datenmodell enthält die dauerhaften (persistenten) Daten der Anwendung. Das Model hat lesenden Zugriff auf diverse Backend-Speicher wie zum Beispiel Datenbanken.

**MVC**
Model-View-Controller bezeichnet ein Architekturmuster zur Trennung von Softwaresystemen in die drei Einheiten Datenmodell (engl. Model), Präsentation (engl. View) und Programmsteuerung (engl. Controller). In diesem Projekt wird der Controller State genannt.

**UnitTests**
Teil eines Softwareprozess-Vorgehensmodells, dienen zur Validierung der Korrektheit von Modulen einer Software, z.B. von einzelnen Klassen.

**Refactoring**
Code zusammenfassen.

**Ruby**
Objektorientierte Programmiersprache.

**View**
Die Darstellungsschicht präsentiert in der Regel die Daten. Die Programmlogik sollte aus dem View entfernt werden. Der View kennt das Model und ist dort registriert, um sich selbständig aktualisieren zu können.

## 3.2. Literaturverzeichnis

**Informationen zu ExtremeProgramming**
http://de.wikipedia.org/wiki/Extreme_Programming

**Ruby Referenz und weiterführende Informationen**
http://www.ruby-doc.org

**Deutsche Ruby Dokumentation**
http://home.vr-web.de/juergen.katins/ruby/buch

**ExtremeProgramming**
http://de.wikipedia.org/wiki/Extreme_Programming
http://www.frankwestphal.de/ExtremeProgramming.html

## 3.3. Code Anhang

Alles was Fett gedruckt ist habe ich geschrieben.

### 3.3.1. Bewilligungsstatus und Ferien Editieren

File: src/model/holiday.rb

```
require 'date'

module HolidayManager
	module Model
		class Holiday
			attr_accessor  :start_holiday, :end_holiday,
				:holiday, :statement, :user, :allocation
			def days
				days = []
				dates = (@start_holiday..@end_holiday).to_a
				dates.each { |date|
					if(date.wday != 0 && date.wday != 6)
						days.push(date)
					end
				}
				days.size
			end
			def days_in_year(year)
				days = []
				dates = (@start_holiday..@end_holiday).to_a
				dates.each { |date|
					if(date.wday != 0 && date.wday != 6 &&
					   date.year == year)
						days.push(date)
					end
				}
				days.size
			end
		end
	end
end
```

File: src/state/edit_holiday.rb

```
require 'view/edit_holiday'
require 'state/global_predefine'

module HolidayManager
	module State
		class EditHoliday < State::Global
			VIEW = View::EditHoliday
			def init
				start_holiday = @session.user_input(:start_holiday)
				end_holiday = @session.user_input(:end_holiday)
				statement = @session.user_input(:statement)
				@model,   @user   =   @session.get_holiday(start_holiday,   end_holiday,
statement)
			end
			def update
				mandatory = [:start_holiday, :end_holiday ]
				keys = mandatory + [ :statement ]
				input = user_input(keys, mandatory)
				if(input[:start_holiday] == nil)
					error = create_error('e_missing_date',
						:start_holiday, nil)
					@errors.store(:start_holiday, error)
				end
				if(input[:end_holiday] == nil)
					error = create_error('e_missing_date',
					:end_holiday, nil)
					@errors.store(:end_holiday, error)
				end
				if(input[:statement].size >= 500)
					error = create_error('e_wrong_string',
						:statement, nil)
```

```
                                @errors.store(:statement, error)
                        end
                        unless(error?)
                                @model.start_holiday = input[:start_holiday]
                                @model.end_holiday = input[:end_holiday]
                                @model.statement = input[:statement]
                                @user.odba_store
                                State::DetailUser.new(@session, @user)
                        end
                end
        end
        class AdminEditHoliday < EditHoliday
                VIEW = View::AdminEditHoliday
                def holiday_allowed_report(email, start_holiday, end_holiday)
                        lookandfeel = @session.lookandfeel
                        mail = TMail::Mail.new
                        mail.set_content_type('text', 'plain', 'charset'=>'ISO-8859-1')
                        mail.to = email
                        mail.from = "noreplay@ywesee.com"
                        mail.subject = "Ferieneintrag Bewilligt - HolidayManager"
                        mail.date = Time.now
                        mail.body = [
                                "#{lookandfeel.lookup(:hello)} #{email}",
                                nil,
                                "Ihr Ferieneintrag vom #{start_holiday} bis
                                #{end_holiday} wurde bewilligt.",
                                nil,
                        ].join("\n")
                        Net::SMTP.start('mail.ywesee.com') { |smtp|
                                smtp.sendmail(mail.encoded, 'jlang@ywesee.com',
                                [email])
                        }
                end
                def update
                        mandatory = [:start_holiday, :end_holiday, :allocation ]
                        keys = mandatory + [ :statement ]
                        input = user_input(keys, mandatory)
                        if(input[:start_holiday] == nil)
                                error = create_error('e_missing_date',
                                        :start_holiday, nil)
                                @errors.store(:start_holiday, error)
                        end
                        if(input[:end_holiday] == nil)
                                error = create_error('e_missing_date',
                                :end_holiday, nil)
                                @errors.store(:end_holiday, error)
                        end
                        if(input[:statement].size >= 500)
                                error = create_error('e_wrong_string',
                                        :statement, nil)
                                @errors.store(:statement, error)
                        end
                        unless(error?)
                                allocation_now = @model.allocation
                                @model.start_holiday = input[:start_holiday]
                                @model.end_holiday = input[:end_holiday]
                                @model.statement = input[:statement]
                                @model.allocation = input[:allocation]
                                @user.odba_store
                                if(input[:allocation] == "allocation_yes" &&
                                        allocation_now == 'allocation_no')
                                        holiday_allowed_report(@user.email,
                                          @model.start_holiday, @model.end_holiday)
                                end
                                State::DetailUser.new(@session, @user)
                        end
                end
        end
    end
end
```

File: src/view/edit_holiday.rb

```
require 'view/template'
require 'htmlgrid/list'
require 'htmlgrid/select'
require 'htmlgrid/form'
require 'htmlgrid/inputtext'
require 'htmlgrid/inputcheckbox'
require 'htmlgrid/inputdate'
require 'htmlgrid/errormessage'
require 'custom/lookandfeel'

module HolidayManager
	module View
		class EditHolidayForm < HtmlGrid::Form
			include HtmlGrid::ErrorMessage
			EVENT = :update
			COMPONENTS = {
				[0,0]  =>     :start_holiday,
				[0,1]  =>     :end_holiday,
				[0,2]  =>     :statement,
				[0,3,0]=>     :submit,
				[0,3,1]=>     :cancel,
				[0,4]  =>     'newline',
			}
			SYMBOL_MAP = {
				:start_holiday =>  HtmlGrid::InputDate,
				:end_holiday          =>  HtmlGrid::InputDate,
			}
			CSS_MAP = {
				[0,2]  =>     'top',
			}
			LABELS = true
			CSS_CLASS = 'composite'
			LEGACY_INTERFACE = false
			COMPONENT_CSS_MAP = {
				[0,3]  =>     'button',
			}
			def init
				super
				error_message
			end
			def cancel(model)
				button = HtmlGrid::Button.new(:cancel, model, @session, self)
				button.set_attribute("class", "button")
				url = @lookandfeel.event_url(:cancel)
				button.set_attribute('onclick', "location.href='#{url}'")
				button
			end
			def statement(model)
				input = HtmlGrid::Textarea.new(:statement, model, @session, self)
				input.set_attribute('cols', 43)
				input.set_attribute('rows', 4)
				input.set_attribute('wrap', true)
				js = "if(this.value.length > 500) {
				  (this.value = this.value.substr(0,500))}"
				input.set_attribute('onKeypress', js)
				input.label = true
				input
			end
		end
		class AdminEditHolidayForm < EditHolidayForm
			include HtmlGrid::ErrorMessage
			EVENT = :update
			COMPONENTS = {
				[0,0]  =>     :start_holiday,
				[0,1]  =>     :end_holiday,
				[0,2]  =>     :statement,
				[0,3]  =>     :allocation,
				[0,4,0]=>     :submit,
				[0,4,1]=>     :cancel,
				[0,5]  =>     'newline',
			}
			SYMBOL_MAP = {
				:allocation    =>  HtmlGrid::Select,
				:start_holiday =>  HtmlGrid::InputDate,
				:end_holiday          =>  HtmlGrid::InputDate,
			}
			CSS_MAP = {
				[0,3]  =>     'top',
```

```
                        }
                        LABELS = true
                        CSS_CLASS = 'composite'
                        LEGACY_INTERFACE = false
                        COMPONENT_CSS_MAP = {
                                [0,4]  =>     'button',
                        }
                end
                class EditHolidayComposite < HtmlGrid::Composite
                        COMPONENTS = {
                                [0,0]  =>     EditHolidayForm,
                        }
                        CSS_CLASS = 'composite'
                        LEGACY_INTERFACE = false
                end
                class AdminEditHolidayComposite < HtmlGrid::Composite
                        COMPONENTS = {
                                [0,0]  =>     AdminEditHolidayForm,
                        }
                        CSS_CLASS = 'composite'
                        LEGACY_INTERFACE = false
                end
                class EditHoliday < PrivateTemplate
                        STATE_TITLE = :edit_holiday
                        CONTENT = EditHolidayComposite
                end
                class AdminEditHoliday < PrivateTemplate
                        STATE_TITLE = :edit_holiday
                        CONTENT = AdminEditHolidayComposite
                end
        end
end
```

### 3.3.2. Benutzerübersicht

File: src/view/detail_user.rb

```
…

require 'custom/lookandfeel'
require 'htmlgrid/list'
require 'htmlgrid/formlist'
require 'view/template'

module HolidayManager
        module View
                class DetailUserList < HtmlGrid::List
                        COMPONENTS = {
                                [0,0]  =>     :start_holiday,
                                [1,0]  =>     :end_holiday,
                                [2,0]  =>     :days,
                                [3,0]  =>     :statement,
                                [4,0]  =>     :allocation,
                                [5,0,0]=>     :del_holiday,
                                [5,0,1]=>     :edit_holiday,
                        }
                        DEFAULT_CLASS = HtmlGrid::Value
                        CSS_CLASS = 'composite'
                        CSS_MAP = {
                                [0,0]  =>     'list bg',
                                [1,0]  =>     'list bg',
                                [2,0]  =>     'list bg',
                                [3,0]  =>     'list bg',
                                [4,0]  =>     'list bg',
                                [5,0]  =>     'list bg',
                                [6,0]  =>     'list bg',
                        }

                        LEGACY_INTERFACE = false
                        SORT_DEFAULT = :start_holiday
                        SORT_REVERSE = true
                        def allocation(model)
                                if(model.allocation == "allocation_yes")
                                        "Ja"
                                elsif(model.allocation == "allocation_no")
```

```
                                        "Nein"
                                end
                        end
                        def del_holiday(model)
                                if(@session.user.takes?(model))
                                        button = HtmlGrid::Button.new(:delete, model, @session, self)
                                        button.set_attribute("class", "button")
                                        args = {
                                                :email                => @session.user_input(:email),
                                                :start_holiday => model.start_holiday,
                                                :end_holiday    => model.end_holiday,
                                        }
                                        url = @lookandfeel.event_url(:delete, args)
                                        button.set_attribute('onclick', "location.href='#{url}'")
                                        button
                                end
                        end
                        def edit_holiday(model)
                                if(@session.user.takes?(model) || @session.user.group == 'group_admin')
                                        button  =  HtmlGrid::Button.new(:edit_holiday,  model,  @session,
self)
                                        button.set_attribute("class", "button")
                                        args = {
                                                :start_holiday => model.start_holiday,
                                                :end_holiday    => model.end_holiday,
                                                :statement      => model.statement,
                                        }
                                        url = @lookandfeel.event_url(:edit_holiday, args)
                                        button.set_attribute('onclick', "location.href='#{url}'")
                                        button
                                end
                        end
                        def start_holiday(model)
                                if(date = model.start_holiday)
                                        date.strftime("%d.%m.%Y")
                                end
                        end
                        def end_holiday(model)
                                if(date = model.end_holiday)
                                        date.strftime("%d.%m.%Y")
                                end
                        end
                end
                class DetailUserComposite < HtmlGrid::Composite
                        COMPONENTS = {
                                [0,0]          =>     :detail_user_list,
                                [0,1]          =>     'newline',
                                [0,2,1] =>     :back,
                        }
                        CSS_CLASS = 'composite'
                        LEGACY_INTERFACE = false
                        COMPONENT_CSS_MAP = {
                                [0,2]  =>     'button',
                        }
                        def back(model)
                                button = HtmlGrid::Button.new(:back, model, @session, self)
                                button.set_attribute("class", "button")
                                url = @lookandfeel.event_url(:back)
                                button.set_attribute('onclick', "location.href='#{url}'")
                                button
                        end
                        def detail_user_list(model)
                                DetailUserList.new(model.holidays, @session, self)
                        end
                end
                class DetailUser < PrivateTemplate
                        STATE_TITLE = :detail_user
                        CONTENT = DetailUserComposite
                end
        end
end
…
```

### 3.3.3. Ferienübersicht

File: src/view/users.rb

```
…
                def this_year(model)
                        not_allotted_holidays = model.not_allotted_days(Date.today.year)
                        holidays = model.total_days(Date.today.year) -  not_allotted_holidays
                        if(holidays != 0 && not_allotted_holidays != 0)
                                "#{model.this_year - holidays}
                                      (#{not_allotted_holidays})/#{model.this_year}"
                        elsif(holidays == 0 && not_allotted_holidays != 0)
                                      "(#{not_allotted_holidays})/#{model.this_year}"
                        elsif(holidays != 0 && not_allotted_holidays == 0)
                                      "#{model.this_year - holidays}/#{model.this_year}"
                        else
                                      model.this_year
                end
…
```

File: src/model/user.rb

```
…
                def not_allotted_days(year)
                        not_allotted_holidays = []
                        @holidays.each { |holiday|
                                if(holiday.allocation == "allocation_no")
                                        not_allotted_holidays.push(holiday)
                                end
                        }
                        not_allotted_holidays.inject(0) { |sum, holiday|
                                sum + holiday.days_in_year(year)
                }
                end
…
```

### 3.3.4. Automatisches E-Mail

File: src/state/enter_holiday.rb

```
require 'view/enter_holiday'
require 'state/global_predefine'

module HolidayManager
        module State
                class EnterHoliday < State::Global
                        VIEW = View::EnterHoliday
                        def holiday_report(email, start_holiday, end_holiday, statement)
                                lookandfeel = @session.lookandfeel
                                mail = TMail::Mail.new
                                mail.set_content_type('text', 'plain', 'charset'=>'ISO-8859-1')
                                mail.to = 'jlang@ywesee.com'
                                mail.from = "noreplay@ywesee.com"
                                mail.subject = "Neuer Ferieneintrag - HolidayManager"
                                mail.date = Time.now
                                args = {
                                        :start_holiday => start_holiday,
                                        :end_holiday => end_holiday,
                                        :statement => statement,
                                        :email => email,
                                }
                                link = lookandfeel._event_url(:edit_holiday, args)
                                mail.body = [
                                        "#{lookandfeel.lookup(:hello)}Administrator",
                                        nil,
                                        "#{email} #{lookandfeel.lookup(:holiday_report)}",
                                        link,
                                        nil,
                                ].join("\n")
                                Net::SMTP.start('mail.ywesee.com') { |smtp|
                                        smtp.sendmail(mail.encoded,                  'jlang@ywesee.com',
['jlang@ywesee.com'])
                                }
                        end
```

```
                        def enter
                                @holiday = HolidayManager::Model::Holiday.new
                                mandatory = [:start_holiday, :end_holiday ]
                                keys = mandatory + [ :statement ]
                                input = user_input(keys, mandatory)
                                if(input[:start_holiday] == nil)
                                        error = create_error('e_missing_date',
                                                :start_holiday, nil)
                                        @errors.store(:start_holiday, error)
                                end
                                if(input[:end_holiday] == nil)
                                        error = create_error('e_missing_date',
                                        :end_holiday, nil)
                                        @errors.store(:end_holiday, error)
                                end
                                if(input[:statement].size >= 500)
                                        error = create_error('e_wrong_string',
                                                :statement, nil)
                                        @errors.store(:statement, error)
                                end
                                unless(error?)
                                        @holiday.start_holiday = input[:start_holiday]
                                        @holiday.end_holiday = input[:end_holiday]
                                        @holiday.statement = input[:statement]
                                        @holiday.allocation = "allocation_no"
                                        @model.add_holiday(@holiday)
                                        @model.odba_store
                                        holiday_report(@model.email, @holiday.start_holiday,
                                         @holiday.end_holiday, @holiday.statement)
                                        State::DetailUser.new(@session, @model)
                                end
                        end
                end
        end
end
```

### 3.3.5. Transparenter Login

File: src/model/login.rb

```
require 'util/server'
require 'view/login'
require 'state/global_predefine'
require 'state/users'

module HolidayManager
        module State
                class Login < SBSM::State
                        VIEW = View::Login
                        def login
                                state = self
                                if(usr = @session.login)
                                        if(@session.user_input(:next_event) == "edit_holiday" &&
                                          usr.group == 'group_admin')
                                                state = State::AdminEditHoliday.new(@session, usr)
                                        elsif(usr.group == 'group_admin')
                                                state = State::UsersAdmin.new(@session, nil)
                                        else
                                                if(usr.pass == "81dc9bdb52d04dc20036dbd8313ed055")
                                                        state = State::UserProfile.new(@session, usr)
                                                else
                                                        state = State::Users.new(@session, nil)
                                                end
                                        end
                                        if(mod = usr.viral_module)
                                                state.extend(mod)
                                        end
                                end
                                state
                        end
                        def user_profile
                                user = @session.get_user(@session.user_input(:email))
                                if(user && user.incomplete?)
                                        State::IncompleteUserProfile.new(@session, user)
                                end
```

```
                        end
                        def edit_holiday
                                @session.start_holiday = @session.user_input(:start_holiday)
                                @session.end_holiday = @session.user_input(:end_holiday)
                                @session.statement = @session.user_input(:statement)
                                @session.next_event = :edit_holiday
                                self
                        end
                end
        end
end
```

File: src/view/login.rb

```
require 'view/template'
require 'htmlgrid/pass'
require 'htmlgrid/form'

module HolidayManager
        module View
                class LoginForm < HtmlGrid::Form
                        EVENT = :login
                        COMPONENTS = {
                                [0,0]  =>     :loginname,
                                [0,1]  =>     :pass,
                                [1,2]  =>     :submit,
                                [1,3,0] =>    :next_event,
                                [1,3,1] =>    :start_holiday,
                                [1,3,2] =>    :end_holiday,
                                [1,3,3] =>    :statement,
                        }
                        SYMBOL_MAP = {
                                :loginname  =>  HtmlGrid::InputText,
                                :pass                   =>  HtmlGrid::Pass,
                        }
                        COMPONENT_CSS_MAP = {
                                [1,2]  =>     'button',
                        }
                        CSS_CLASS = 'component'
                        LABELS = true
                        def next_event(model, session)
                                hidden = HtmlGrid::Input.new("next_event",
                                   model, session, self)
                    hidden.set_attribute('type', 'hidden')
                                hidden.value = session.next_event
                                hidden
                        end
                        def start_holiday(model, session)
                                hidden = HtmlGrid::Input.new("start_holiday",
                                   model, session, self)
                    hidden.set_attribute('type', 'hidden')
                                hidden.value = session.start_holiday
                                hidden
                        end
                        def end_holiday(model, session)
                                hidden = HtmlGrid::Input.new("end_holiday",
                                   model, session, self)
                    hidden.set_attribute('type', 'hidden')
                                hidden.value = session.end_holiday
                                hidden
                        end
                        def statement(model, session)
                                hidden = HtmlGrid::Input.new("statement",
                                   model, session, self)
                    hidden.set_attribute('type', 'hidden')
                                hidden.value = session.statement
                                hidden
                        end
                end
                class LoginComposite < HtmlGrid::Composite
                        COMPONENTS = {
                                [0,0]  =>     "login_welcome",
                                [0,1]  =>     LoginForm,
                        }
                        CSS_CLASS = 'composite'
                        CSS_MAP = {
                                [0,0]  =>     'th big',
```

```
                        }
                end
                class Login < PublicTemplate
                        CONTENT = LoginComposite
                end
        end
end
```