

Bachelor-Projekt: Realisierung von Geschäftsprozessen der o/ZB Stuttgart

Auftraggeber: Tassilo Kienle, o/ZB Stuttgart
Betreut von: Prof. Dr. Frank Schaefer, Hochschule Karlsruhe
Erstellt von: Martin Briewig, Michael Leibel
Erstellt im: Wintersemester 2011/2012

Inhalt

1. Einleitung (ML/MB)	6
1.1 Aufgabe	6
2.1 Die o/ZB Stuttgart (MB)	7
2.2 Ruby (MB)	7
2.3 Ruby on Rails (MB)	8
2.3.1 Model	8
2.3.2 View	8
2.3.3 Control	8
2.3.4 Weitere Module	9
2.3.5 Scaffolding	9
2.4 REST (MB)	9
2.4.1 Adressierbarkeit	9
2.4.2 Unterschiedliche Repräsentationen	10
2.4.3 Zustandslosigkeit	10
2.4.4 Wohldefinierte Operationen	10
2.4.5 Verwendung von Hypermedia	10
2.4.6 Umsetzung	10
2.5 jQuery (ML)	11
2.6 Ajax (ML)	11
2.7 Ruby-Gems (ML)	11
2.7.1 Devise	11
2.7.2 will_paginate	11
2.7.3 Composite Primary Keys	12
2.7.4 SQLite	12
2.7.5 MySQL	12
3. Beschreibung der Web-Applikation	13
3.1 Personenverwaltung (ML)	13
3.1.1 Person Anlegen	13
3.1.2 Person Bearbeiten	19

3.1.3 Person Löschen.....	19
3.2 Kontenverwaltung (MB).....	19
3.2.1 Einlage-/Entnahme-Konto.....	20
3.2.2 Zusatzentnahme-Konto	22
3.3 Bürgschaften (MB)	24
Anlegen	25
Bearbeiten	26
3.4 Kontenklassenverwaltung (MB).....	27
Anlegen	28
Bearbeiten	28
3.5 Reports (MB).....	29
Konten.....	29
Adressen	29
3.6 Aufbau der Web-Applikation (ML)	30
3.6.1 Login	30
3.6.3 Dashboard/Grundlayout.....	31
Grundlayout	31
Dashboard (Meine Konten)	32
3.6.4 Meine Daten	33
3.6.5 Administration.....	34
4. Implementierungsdetails	35
4.1 Datenmodell/Model (ML/MB).....	35
4.2 Controller	37
4.2.1 OZB Person (ML)	37
4.2.1.1 index	37
4.2.1.2 new/create	37
4.2.1.3 edit/update	37
4.2.1.4 delete	37
4.2.2 Application (ML/MB)	37
4.2.2.1 searchKtoNr	38
4.2.2.2 searchOZBPerson.....	38
4.2.3 Index (ML/MB)	38
4.2.4 Bürgschaft (MB).....	38

4.2.4.1 index	39
4.2.4.2 new	39
4.2.4.3 searchKtoNr	39
4.2.4.4 searchOZBPerson.....	39
4.2.4.5 edit.....	40
4.2.4.6 create.....	40
4.2.4.7 update.....	41
4.2.4.8 delete	42
4.2.5 Kontenklasse (MB)	42
4.2.6 OzbKonto (MB).....	42
4.2.6.1 get_konten	43
4.2.6.2 kkl	43
4.2.6.3 changeKKL	44
4.2.6.4 show	44
4.2.6.5 new	45
4.2.6.6 create.....	45
4.2.6.7 update.....	46
4.2.6.8 verlauf	46
4.2.7 Reports (MB)	46
4.3 View - RHTML/JQuery	47
4.3.1 OZB Person (ML)	47
4.3.1.1 Index.....	47
4.3.1.2 New.....	47
4.3.1.3 Edit	47
4.3.2 Allgemeiner Aufbau (ML/MB).....	48
4.3.2.1 HTML-Seiten.....	48
Index	48
new	49
edit	52
4.3.2.2 Javascript.....	52
4.4 Devise (ML).....	53
4.4.1 Model-Änderung und Migration.....	53
4.4.2 Routes.....	53

4.4.3 User Login	54
4.4.4 Lokalisierung	54
4.5 jQuery	54
4.5.1 Datum-Popup (ML)	54
4.5.2 Such-Maske (MB)	55
4.5.2.1 Fancybox	55
4.5.2.2 Funktionalität.....	56
4.5.3 Autocomplete (MB)	58
4.6 Validierung (ML/MB)	59
4.6.1 Pflichtfelder.....	60
4.6.2 Ausblick	61
4.7 Internationalisierung (I18N) (ML).....	62
4.7.1 Lokalisierung (L10N)	62
4.8 Security (Rechteverwaltung) (ML)	62
4.9 Layout (CSS) (MB).....	63
4.10 Konfigurationen (ML).....	64
4.10.1 Routes	64
4.10.2. MySQL	64
5. Fazit (ML/MB)	66

1. Einleitung (ML/MB)

Diese Projektarbeit wurde von Herrn Schäfer ausgestellt und in Zusammenarbeit von Herrn Briewig und Herrn Leibel bearbeitet.

1.1 Aufgabe

Die ursprüngliche Aufgabe im Wortlaut:

Realisierung von Geschäftsvorfällen (o/ZB)

Die OhneZinsBewegung o/ZB in Stuttgart betreibt eine Web-Seite mit verschiedenen Anwendungen für ihre Mitglieder. Für diese Anwendung wurden in den vorigen Semestern bereits die wesentlichen Geschäftsvorfälle beschrieben. In dieser Praxisarbeit geht es jetzt darum, diese Geschäftsvorfälle zu realisieren. Dazu soll die Umsetzbarkeit in PHP und Ruby-on-Rails geprüft und anschließend realisiert werden.

Aufgrund der großen Menge von Geschäftsvorfällen in der o/ZB wurden die Ziele dieser Projektarbeit eingegrenzt.

Es wird evaluiert, ob Ruby on Rails für ein Backend bzw. Frontend Tool, das die Geschäftsvorfälle der o/ZB abbildet, geeignet ist.

Dies soll im ersten Schritt durch einen POC (Proof of Concept) geprüft bzw. bewiesen werden.

Dieser POC soll das von Herrn Okunevych vorgestellte Datenmodell implementieren. Des

Weiteren soll gezeigt werden, dass Operationen wie das Anlegen, Ändern und Löschen von Datensätzen, sowohl visuell als auch effektiv, ansprechend umgesetzt werden können.

Im Anschluss daran wurde der Umfang der zu implementierenden Geschäftsvorfälle festgelegt.

Außerdem soll ein Re-Design des aktuell eingesetzten Backend-Tools umgesetzt werden.

Die nun, für diese Projektarbeit, relevanten Geschäftsvorfälle sind wie folgt:

A. Personen verwalten

1. o/ZB-Person
 - Mitglied
 - Gesellschafter
 - Student
 - Partnermitglied
 - Fördermitglied
 - Administrator
2. Externe Person

B. Konten verwalten

1. Einlage-/Entnahme-Konto
2. Zusatzentnahme-Konto
3. Bürgschaften

Im Folgenden wird, um den Einstieg in die anschließende Vorstellung der umgesetzten Geschäftsprozesse und ihre Implementierung zu erleichtern, die Anwendungsdomäne beschrieben.

2. Grundlagen der Anwendungsdomäne

2.1 Die o/ZB Stuttgart¹ (MB)

Die Solidargemeinschaft o/ZB Stuttgart wurde am 21. Januar 2005 als GbR von 10 GesellschafterInnen gegründet und hatte Mitte des Jahres bereits 100 Mitglieder.

Die o/ZB ist ein regionales Finanzierungsinstrument, das seinen Mitgliedern die Realisierung von Projekten in Selbsthilfe ermöglicht. Anders als bei klassischen Einrichtungen erfolgen alle Einlagen (z.B. Sparen) und Entnahmen (z.B. Leihen) zinslos und kostenfrei, weil alle anfallenden Arbeiten von ihren Mitgliedern in Selbstverwaltung ausgeführt werden können.

Abbildung 1 zeigt das von der o/ZB verwendete Dreiphasenkonzept.

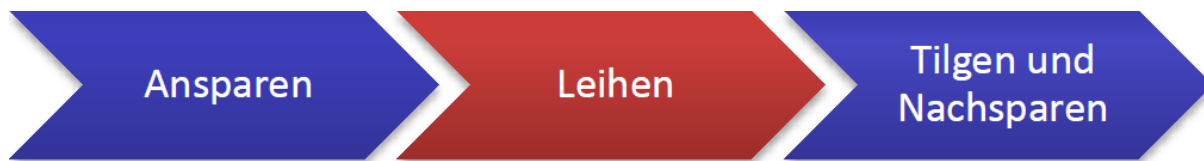


Abbildung 1: Dreiphasenkonzept der o/ZB

2.2 Ruby² (MB)

Ruby ist eine von Yukihiro Matsumoto entworfene und 1995 erschienene höhere Programmiersprache. Der Name Ruby wird vom Edelstein Rubin abgeleitet und ist eine Anspielung auf die Programmiersprache Perl.

Sie ist plattformunabhängig und objektorientiert, bietet dynamische Typisierung, Reflexion (d.h. Ruby erkennt die Struktur des Programms und kann ihn ggf. modifizieren) und eine automatische Bereinigung des Speichers. Außerdem wird sie interpretiert und unterstützt nicht nur das Paradigma der Objektorientierung. Dem Benutzer soll es möglich sein, auch weitere Programmierparadigmen anzuwenden, wie z.B. der prozeduralen/funktionalen Programmierung oder auch der Nebenläufigkeit, man spricht auch von einer "Multiparadigmen-Sprache".

Ruby wurde durch Sprachen wie z.B. Smalltalk, Perl, Python und JavaScript beeinflusst. Dank der GPL Lizenz ist sie frei verfügbar.

¹ Schriftliche Ausarbeitung von Herrn Dmytro Okunevych

² [http://de.wikipedia.org/wiki/Ruby_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Ruby_(Programmiersprache))

2.3 Ruby on Rails³ (MB)

Ruby on Rails wurde von David Heinemeier Hansson in Ruby geschrieben und ist ein Web-Application Framework. Erschienen ist es 2004 und unterliegt der MIT-Lizenz.

Es verfolgt verschiedene Prinzipien, wie z.B. der "Don't Repeat Yourself" welches Redundanzen vermeidet, z.B. durch Verwendung von Templates.

Ein anderes Prinzip ist "Konvention vor Konfiguration", der Entwickler hält sich dabei an den üblichen Konventionen und vereinfacht dadurch die Konfiguration, dies macht sich z.B. bei der Benennung von Klassen bemerkbar.

Dadurch kommen wir zur Architektur: Model-View-Controller. Im Bezug auf die Benennung bedeutet dies, dass bei Rails ein Model auch gleichzeitig einer Tabelle in der Datenbank zugeordnet wird, ohne das sich der Entwickler um die Konfiguration dessen kümmern muss. In Rails wird die MVC Architektur wie folgt umgesetzt:

2.3.1 Model

Das Back-end einer Rails-Applikation ist im Normalfall eine relationale Datenbank. Jede Tabelle repräsentiert eine Klasse, jede Zeile eine Instanz. Das Modul ActiveRecord gewährt den Zugriff auf die Datenbank. Rails unterstützt alle gängigen Datenbanken, wie z.B. MySQL, Oracle, Microsoft SQL Server und noch viele weitere.

2.3.2 View

Um die Ausgabe, bzw. die Präsentation der Applikation, kümmert sich die Klasse ActionView, welche zum ActionPack Modul gehört. Jede Ausgabe wird mit Hilfe eines Templates - einem Grundgerüst - bewerkstelligt. Der Entwickler legt ein Template in seinem gewünschten Ausgabeformat an und Rails füllt diese mit den gewünschten Daten aus. Es werden HTML, XML, JavaScript und Binärdaten als Ausgabe unterstützt. Dazu gehört auch das JSON-Format, welches dem XML-Format ähnelt. Darauf wird im späteren Verlauf kurz eingegangen.

Außerdem ist es erlaubt, den HTTP-Header zu manipulieren, um dem Endanwender auch andere Formate zuzusenden.

2.3.3 Control

In Rails kümmert sich die ActionController Klasse, welche Bestandteil des ActionPack Moduls ist, um die Implementierung der Logik und ist die Schnittstelle zwischen Model und View. Rails erkennt anhand der URL und des eingestellten Routings, welche Aktion ausgeführt werden soll. Im Normalfall besteht ein direkter Zusammenhang zwischen dem Klassennamen und der URL. Es besteht außerdem die Möglichkeit, dass Routing durch die Eigenschaften einer REST-Architektur anzupassen. Somit ist es möglich, dass z.B. zwischen einem GET und einem POST Request unterschieden werden kann.

³ http://de.wikipedia.org/wiki/Ruby_on_Rails

2.3.4 Weitere Module

- **Active Support**
Beinhaltet die Ruby-Erweiterungen
- **Action Mailer**
Stellt E-Mail Versand und Empfang zur Verfügung
- **Active Ressource**
Bietet Web-Service-Programmierung und die Umsetzung für XML-Remote Procedure Calls und der REST (Representational State Transfer) Architektur

Des Weiteren bindet Rails die Java-Script Frameworks Prototype und Scriptaculous ein. Jedoch wird in diesem Projekt das Framework jQuery, welches noch kurz vorgestellt wird, benutzt. Mit Hilfe von JavaScript und den Möglichkeiten von Rails lassen sich auch die Vorzüge von Ajax nutzen.

2.3.5 Scaffolding

Eine Besonderheit an Rails ist das Scaffolding, zu Deutsch Gerüstbau. Mit Hilfe dieser Funktion lassen sich Gerüste von Web-Applikationen bauen, oder Bestandteile eines Gerüsts - wie z.B. ein Model. Dabei legt Rails von selbst die benötigten (Quell-)Dateien und Konfigurationen an. Dank des Scaffolding ist es sehr leicht möglich, die immense Anzahl an freien Erweiterungen in die eigene Rails-Applikation einzubinden. Unterstützt wird dies zudem noch durch ein von Rails zur Verfügung gestelltem Dienst, der sich um die Erweiterungen einer Applikation kümmert. Mit ihm ist es sehr leicht zu bewerkstelligen Erweiterungen zu (de-)installieren.

2.4 REST⁴ (MB)

REST steht für Representational State Transfer. Es bezeichnet eine Architektur für Hypermedia-Informationssysteme (Informationen werden durch Verweise auf sog. Informationsknoten dargestellt. Jeder Knoten beinhaltet Informationen in einem beliebigen Format). Hauptanwendung ist hier das World Wide Web.

In der Dissertation von Roy Fielding aus dem Jahre 2000, wird diese Architektur beschrieben. Sie greift auf die Eigenschaften von Mechanismen und Protokollen, wie z.B. dem HTTP Protokoll, zu.

Ziel ist es, Daten über das HTTP zu übertragen, ohne eine weitere Transportschicht zu benutzen.

REST verfolgt verschiedene Prinzipien:

2.4.1 Adressierbarkeit

Jede Ressource, die zur Verfügung gestellt wird, erhält eine eindeutige Adresse - den Uniform Resource Identifier (kurz URI).

⁴ http://de.wikipedia.org/wiki/Representational_State_Transfer

2.4.2 Unterschiedliche Repräsentationen

Informationen können in unterschiedlichen Formaten ausgegeben werden.

2.4.3 Zustandslosigkeit

Hier wird in 2 Kategorien unterschieden:

Der Ressourcenzustand gibt Informationen über die Ressource und deren Zustand wieder. Die Verwaltung liegt hierbei beim Server.

Der Anwendungszustand zeigt die aktuelle Position des Anwenders in der Anwendung auf. Hier liegen die Verwaltungsaufgaben beim Anwender.

Es werden grundsätzlich keine Informationen über den Zustand eines Verhältnisses zwischen Client und Server gespeichert, da alle benötigten Informationen in einer HTTP-Botschaft vorliegen, um die Nachricht zu verstehen. Jede Anfrage und Antwort ist abgeschlossen, diese Eigenschaften lassen es zu, eine REST-konforme Web-Applikation als zustandslos zu bezeichnen.

2.4.4 Wohldefinierte Operationen

Es werden Operationen definiert, die auf alle Informationen bzw Ressourcen angewandt werden können. Diese sind im Falle des HTTP:

GET, POST, PUT und DELETE. Auf diese wird im weiteren Verlauf noch näher eingegangen.

2.4.5 Verwendung von Hypermedia

Zum Austausch von Informationen wird Hypermedia benutzt. Im Regelfall werden die Formate HTML oder XML verwendet. Diese enthalten die benötigten Informationen oder Links zu anderen Ressourcen.

2.4.6 Umsetzung

Im Folgenden werden die oben genannten Operationen kurz erläutert.

GET

Fordert die angegebene Ressource vom Server an. Dies ist eine "sichere" Operation, d.h. sie beschafft nur Informationen und verursacht keine weiteren Effekte.

POST

Fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein. Wird u.a. dazu benutzt Daten zu übermitteln, oder aber auch große Datenmengen, die in einem GET Request nicht hineinpassen.

PUT

Die angegebene Ressource wird angelegt oder geändert.

DELETE

Löscht die angegebene Ressource.

HEAD

Fordert die Metadaten zu einer Ressource vom Server an.

OPTIONS

Prüft, welche Methoden auf einer Ressource zur Verfügung stehen.

2.5 jQuery⁵ (ML)

jQuery ist eine seit 2006 frei erhältliche JavaScript Klassenbibliothek. Sie bietet unter anderem viele Funktionen zur DOM (Document Object Model) -Manipulation und Navigation. Dies ermöglicht eine unkomplizierte Modifizierung eines bestehenden HTML Dokuments. Im Zusammenspiel mit Ajax ist jQuery eine sehr wertvolle Bibliothek.

Ein weiterer Vorteil von jQuery ist, dass es eine sehr große Anzahl an frei verfügbaren Plugins gibt. Somit findet man sehr leicht sehr nützliche Plugins, die die Benutzung einer Web-Applikation verbessern.

2.6 Ajax⁶ (ML)

Ajax - Asynchronus JavaScript and XML.

Dahinter verbirgt sich das Konzept einer asynchronen Datenübertragung zwischen einem Browser und dem Server. Somit können HTTP-Requests durchgeführt werden, während eine HTML-Seite angezeigt wird. Die Besonderheit ist, dass die HTML-Seite modifiziert werden kann, ohne sie komplett neu zu laden, dadurch erfährt der Anwender deutliche Verbesserungen bei der Bedienung der Web-Applikation.

2.7 Ruby-Gems (ML)**2.7.1 Devise**

⁷Devise ist eine Authentifizierungs-Lösung, welche die Controller Logik und die View-Seiten zur Einschränkung der Sichtbarkeit und Rechte steuern kann. Es übernimmt den An- und Abmeldungsprozess, sowie die Session-Verwaltung der angemeldeten Nutzer. Das Passwort des jeweiligen Users wird verschlüsselt in der Datenbank gespeichert.

2.7.2 will_paginate⁸

will_paginate dient zur Darstellung von Daten, welche auf mehrere Seiten aufgeteilt werden. Somit können immer eine genaue Anzahl von Daten innerhalb einer Seite angezeigt und weitere Daten durch Blättern dargestellt werden.

⁵ <http://de.wikipedia.org/wiki/JQuery>

⁶ [http://de.wikipedia.org/wiki/Ajax_\(Programmierung\)](http://de.wikipedia.org/wiki/Ajax_(Programmierung))

⁷ <https://github.com/plataformatec/devise>

⁸ https://github.com/mislav/will_paginate

2.7.3 Composite Primary Keys⁹

Aufgrund der Tatsache, dass Ruby keine zusammengesetzten Primärschlüssel nativ unterstützt, ist dieses Gem erforderlich. Es erweitert Active Record um die geforderten Funktionalitäten.

Die Nutzung ist sehr simpel, um den eben angesprochenen zusammengesetzten Primärschlüssel anzulegen.

```
set_primary_keys :lagerhaus, :regalnummer
```

Auch werden Foreign-Keys benötigt, diese übergibt man mit Hilfe eines Arrays:

```
:foreign_key => [:straße, :hausnummer]
```

Alle für die Web-Applikation wesentlichen Funktionen wurden erläutert, für weitere Informationen steht diese Webseite zur Verfügung: <http://compositekeys.rubyforge.org/>.

2.7.4 SQLite¹⁰

SQLite stellt eine freie Programmbibliothek für ein relationales Datenbanksystem zur Verfügung. Es unterstützt einen Großteil aller im SQL-92-Standard festgelegten SQL-Befehle, sowie Transaktionen, Unterabfragen, Sichten, Trigger und benutzerdefinierten Funktionen.

Eine Besonderheit ist, dass SQLite all seine Daten in einer einzigen Datei abspeichert und somit eine klassische Client/Server Architektur entfällt. Zudem verbraucht diese Datei nur geringen Speicherplatz. Um eine Rechteverwaltung muss sich die Anwendung kümmern, denn diese ist nicht vorhanden, wie z.B. bei einer Client/Server betriebenen Architektur.

SQLite wird heutzutage vermehrt im Embedded Bereich, wie z.B. Smartphones verwendet, aber auch Browser wie Safari und Firefox setzen vermehrt auf SQLite.

In diesem Projekt wird SQLite in der 3ten Version verwendet.

2.7.5 MySQL¹¹

MySQL ist eine relationale Datenbank, welche als Open-Source-Software, sowie als kommerzielle Version verfügbar ist. Die beiden bekanntesten Engines für MySQL sind MyISAM und InnoDB. Es bestehen große Unterschiede zwischen diesen beiden Engines. InnoDB bietet zum einen transaktionssichere Lese- und Schreibzugriffe für Tabellen und die Möglichkeit, Fremdschlüssel-Beziehungen zu überprüfen. MyISAM zeichnet hingegen einen schnellen Zugriff auf Tabellen und Indizes ohne Transaktionssicherung aus, sowie eine sehr effiziente Speicher-Engine.

⁹ <http://compositekeys.rubyforge.org/>

¹⁰ <http://de.wikipedia.org/wiki/SQLite>

¹¹ <http://de.wikipedia.org/wiki/MySQL>

3. Beschreibung der Web-Applikation

3.1 Personenverwaltung (ML)

Die Personenverwaltung kümmert sich um das Anlegen, Bearbeiten und Löschen von Personen. Die Verwaltung selbst zeigt alle angelegten Personen an und die jeweiligen Funktionen die möglich sind.

The screenshot shows the 'Besser ohne / Zins' web application. The header includes the logo and navigation links: 'Meine Konten', 'Meine Daten', 'Meine TAN', 'Administration', 'Protokolle', and 'o/ZBlick'. The user is logged in as 'Mustermann' with an 'Ausloggen' link.

The main content area is titled 'Personen' and contains a table with the following data:

Nr.	Name	Konten	Bearbeiten	Löschen
1	Mustermann, Max	Konten	Bearbeiten	Löschen
2	Mueller, Hermann	Konten	Bearbeiten	Löschen
3	Müller, Lieschen	Konten	Bearbeiten	Löschen
4	Neuer, Manuel	Konten	Bearbeiten	Löschen
5	Kahn, Oliver	Konten	Bearbeiten	Löschen

Below the table, there are navigation links: 'zurück', '1', '2', and 'weiter'. At the bottom of the interface, there are two buttons: 'Zurück' (red) and 'OZB Person hinzufügen' (green).

3.1.1 Person Anlegen

Neben den persönlichen Daten, wie Name, Email-Adresse, Passwort und Anschrift, kann auch eine Rolle ausgewählt werden. Je nach ausgewählter Rolle, werden die nötigen Felder zum ausfüllen angezeigt. Dabei kann es Felder geben, wie beispielsweise "Notar", dabei ist der Notar eine Person im OZB-System, die über ein Suchen-Button ausgewählt werden kann. Somit können keine Fehleingaben entstehen und die Zuordnung ist gesichert. Ebenso werden Pflichtfelder berücksichtigt, sollten diese nicht ausgefüllt worden sein, bekommt der Anwender eine Fehlermeldung und kann die entsprechenden Pflichtfelder setzen.

Mitglied**Person
Hinzufügen**

Name:*	<input type="text"/>
Vorname:	<input type="text"/>
Email:*	<input type="text" value="person1@ozb.de"/>
Passwort:*	<input type="password" value="*****"/>
Geburtsdatum:	<input type="text"/>
Straße:	<input type="text"/>
Hausnummer:	<input type="text"/>
Plz:	<input type="text"/>
Ort:	<input type="text"/>
Telefon:	<input type="text"/>
Fax:	<input type="text"/>
Rolle:	<input type="text" value="Mitglied"/>

Rv datum:	<input type="text"/>
-----------	----------------------

Antragsdatum *:	<input type="text"/>
Aufnahmedatum:	<input type="text"/>

Zurück**Speichern**

Fördermitglied**Person**
Hinzufügen

Name:*	<input type="text"/>
Vorname:	<input type="text"/>
Email:*	<input type="text" value="person1@ozb.de"/>
Passwort:*	<input type="password" value="*****"/>
Geburtsdatum:	<input type="text"/>
Straße:	<input type="text"/>
Hausnummer:	<input type="text"/>
Plz:	<input type="text"/>
Ort:	<input type="text"/>
Telefon:	<input type="text"/>
Fax:	<input type="text"/>
Rolle:	<input type="text" value="Foerdermitglied"/>

Region:*	<input type="text"/>
Förderbeitrag:*	<input type="text"/>

Antragsdatum *:	<input type="text"/>
Aufnahmedatum:	<input type="text"/>

[Zurück](#)[Speichern](#)

Partner**Person**
Hinzufügen

Name:*	<input type="text"/>
Vorname:	<input type="text"/>
Email:*	<input type="text" value="person1@ozb.de"/>
Passwort:*	<input type="password" value="*****"/>
Geburtsdatum:	<input type="text"/>
Straße:	<input type="text"/>
Hausnummer:	<input type="text"/>
Plz:	<input type="text"/>
Ort:	<input type="text"/>
Telefon:	<input type="text"/>
Fax:	<input type="text"/>
Rolle:	<input type="text" value="Partner"/>

Partner:*	<input type="text"/>	<input type="button" value="Suchen..."/>
Berechtigung:*	<input type="text"/>	

Antragsdatum *:	<input type="text"/>
Aufnahmedatum:	<input type="text"/>

Gesellschafter

Person

Hinzufügen

Name:*	<input type="text"/>
Vorname:	<input type="text"/>
Email:*	<input type="text" value="person1@ozb.de"/>
Passwort:*	<input type="password" value="*****"/>
Geburtsdatum:	<input type="text"/>
Straße:	<input type="text"/>
Hausnummer:	<input type="text"/>
Plz:	<input type="text"/>
Ort:	<input type="text"/>
Telefon:	<input type="text"/>
Fax:	<input type="text"/>
Rolle:	<input type="text" value="Gesellschafter"/>

Fa steuernummer:*	<input type="text"/>
Fa lfd. nr.:*	<input type="text"/>
Wohnsitzfinanzamt:*	<input type="text"/>
Notar:	<input type="text"/>
Beurkundungsdatum:	<input type="text"/>

Suchen...

Antragsdatum *:	<input type="text"/>
Aufnahmedatum:	<input type="text"/>

Zurück

Speichern

Student**Person
Hinzufügen**

Name:*	<input type="text"/>
Vorname:	<input type="text"/>
Email:*	<input type="text" value="person1@ozb.de"/>
Passwort:*	<input type="password" value="*****"/>
Geburtsdatum:	<input type="text"/>
Straße:	<input type="text"/>
Hausnummer:	<input type="text"/>
Plz:	<input type="text"/>
Ort:	<input type="text"/>
Telefon:	<input type="text"/>
Fax:	<input type="text"/>
Rolle:	<input type="text" value="Student"/>

Ausbildungsbezeichnung:*	<input type="text"/>
Institut:*	<input type="text"/>
Studienort:*	<input type="text"/>
Studienbeginn:*	<input type="text"/>
Studienende:	<input type="text"/>
Abschluss:*	<input type="text"/>

Antragsdatum *:	<input type="text"/>
Aufnahmedatum:	<input type="text"/>

Zurück**Speichern**

3.1.2 Person Bearbeiten

Alle Daten einer Person können geändert werden, bis auf System-Variablen, wie die Personennummer. Außerdem kann die Rolle der Person nicht geändert werden, dies war so vom Kunden gewünscht. Ansonsten unterscheidet sie sich nicht von der Maske "Person Anlegen".

Hinzu kommt, dass jede Person nur selbst seine persönlichen Daten ändern kann.

The screenshot shows a web application interface for editing a person's data. The header includes the logo 'Besser ohne / Zins' and navigation links: 'Meine Konten', 'Meine Daten', 'Meine TAN', 'Administration', 'Protokolle', and 'o/ZBlick'. The user is logged in as 'Mustermann' and can click 'Ausloggen'. The main form is titled 'Person Bearbeiten' and contains the following fields:

Person-nr.:	1
Name:	Mustermann
Vorname:	Max
Email:	person1@ozb.de
Passwort:	*****
Geburtsdatum:	1984-12-15
Straße:	Musterstraße
Hausnummer:	42
Plz:	76131
Ort:	Karlsruhe
Telefon:	0213259201
Fax:	021325920101
Rolle:	Mitglied
Rv datum:	2010-12-12
Antragsdatum *:	2010-12-10
Aufnahmedatum:	2010-12-12

At the bottom of the form are two buttons: 'Zurück' (red) and 'Ok' (green).

3.1.3 Person Löschen

Die Person kann man über die Personenverwaltungsseite löschen. Hierbei wird nach dem Klick auf den Button eine Bestätigung ausgelöst, nach der Bestätigung wird die Person gelöscht.

3.2 Kontenverwaltung (MB)

Bei der Kontenverwaltung wird unterschieden zwischen der Bearbeitung und der einfachen Einsicht eines Kontos.

Die Bearbeitung eines Kontos ist ein sensibler Bereich, welcher durch die Rechteverwaltung vor unbefugten Zugriffen geschützt wird.

Um ein spezielles Konto zu bearbeiten, wird zunächst eine Person ausgesucht.

Dies geschieht über die Übersicht aller Personen, die im Menüpunkt "Administrator" zu erreichen sind. Hat man die gewünschte Person gefunden, reicht ein einfacher Klick auf das Konto-Symbol aus, um auf die Übersicht aller der Person zugeordneten Konten zu gelangen.

Im Folgenden kann man bis zu drei E/E-Konten und vier ZE-Konten anlegen, bzw. bearbeiten und löschen. Dafür stehen dem Benutzer die farbig markierten Buttons zur Verfügung.

Die einfache Einsicht eines Kontos geschieht in der Regel über den Menüpunkt "Meine Konten".

Dort werden nun alle Konten zum dazugehörigen eingeloggten User angezeigt, die auch nur von ihm oder einer Person mit erweiterten Rechten eingesehen werden können. Den Kontenklassenverlauf kann man jedoch nur als Administrator einsehen, dafür verwendet der Administrator nicht den Menüpunkt "Meine Konten", sondern navigiert zu der Personenübersicht (über Administration->Personen) und wählt dort ein Konto aus.

Besser ohne / Zins Eingeloggt als Mustermann Ausloggen

Meine Konten Meine Daten Meine TAN Administration Protokolle o/ZBlick

OZB Konten

EE-Konten

Konto-Nr.	Bank-Id	Bank-Konto-Nr.	Bankleitzahl	Bankname	Kreditlimit			
21541	43	21315	54123412	Goalas	123214.0	Editieren	KKLVerlauf	Löschen

Neues EE-Konto hinzufügen

ZE-Konten

KtoNr	EEKtoNr	ZEKtoNr	AbDatum	EndDatum	Betrag	Laufzeit	Tilgungsrate	Ansparrate			
10001	50001	2443623	2011-12-24	2011-12-24	231425.0	12	234.0	435.0	Editieren	KKLVerlauf	Löschen

Zurück Neues ZE-Konto hinzufügen

3.2.1 Einlage-/Entnahme-Konto

Einlage-/Entnahme-Konten, oder auch E/E-Konten sind die Grundlage für jedes Zusatzentnahme-Konto. Darum sollte ein E/E-Konto zuerst angelegt werden. Es kann u.a. auch als ein Treuhandkonto verwendet werden.

Beim Anlegen eines E/E-Kontos werden eine Bankverbindung, die Kontoklasse und ein Kreditlimit verlangt. Der Rest wird vorgegeben, wie z.B. die Pnr.

Eine Kontonummer kann nun frei gewählt werden und wird nicht mehr fest vorgegeben, zur Unterstützung wird jedoch eine vorgeschlagen.

Die Saldo-Werte (WSaldo, PSaldo), sowie das Saldo-Datum wird in Zukunft von einem Skript (bzw. von Ruby) generiert.

Besser ohne / Zins

Eingeloggt als Mustermann Ausloggen

Meine Konten

Meine Daten

Meine TAN

Administration

Protokolle

o/ZBlick

EEKonto

Neues EEKonto

OZB-Konto

Konto-Nr.:

wSaldo:

pSaldo:

SaldoDatum:

KKL:

EE-Konto

Kreditlimit:

Bankverbindung

Bank-Konto-Nr.:

Bankname

Bankleitzahl:

BIC:

IBAN:

Abbrechen

Speichern

Besser ohne / Zins

Eingeloggt als Mustermann Ausloggen

Meine Konten

Meine Daten

Meine TAN

Administration

Protokolle

o/ZBlick

EEKonto

EEKonto editieren

OZB-Konto

Konto-Nr.:

wSaldo:

pSaldo:

SaldoDatum:

EE-Konto

Konto-Nr.:

Kreditlimit:

Bankverbindung

Bank-Konto-Nr.:

Bankname

Bankleitzahl:

BIC:

IBAN:

Abbrechen

Speichern

Die Bearbeitung eines E/E-Kontos erfolgt durch ein weiteres Eingabeformular.

3.2.2 Zusatzentnahme-Konto

Zusatzentnahme-Konten, im Folgenden ZE-Konten, benötigen zwingend ein E/E-Konto. Ein ZE-Konto steht für ein Darlehen.

Beim Anlegen eines ZE-Kontos geschieht die Eingabe aller wichtigen Daten, markiert mit einem Stern (*), wie auch bei einem E/E-Konto, durch ein einfaches Eingabeformular.

Eine Besonderheit ist u.a., dass das E/E-Konto durch eine eigens implementierte Suchmaske (wird in Kapitel 4 erläutert) unterstützt wird.

Besser ohne / Zins Eingeloggt als Mustermann [Ausloggen](#)

Meine Konten Meine Daten Meine TAN Administration Protokolle o/ZBlick

ZEKonto

Neues ZEKonto

OZB-Konto

Konto-Nr.:	<input type="text"/>
wSaldo:	<input type="text"/>
pSaldo:	<input type="text"/>
SaldoDatum:	<input type="text"/>
KKL:	<input type="text"/>

ZE-Konto

ZE-Nr.:	<input type="text"/>	Suchen...
EE-Konto-Nr.:	<input type="text"/>	
AbDatum:	<input type="text"/>	
EndDatum:	<input type="text"/>	
Betrag:	<input type="text"/>	
Laufzeit:	<input type="text"/>	
Zahlmodus:	<input type="text"/>	
Tilgungs-Rate:	<input type="text"/>	
Anspar-Rate:	<input type="text"/>	
Kdu-Rate:	<input type="text"/>	
Rdu-Rate:	<input type="text"/>	
Status:	<input type="text"/>	

[Abbrechen](#) [Speichern](#)

Besser ohne / Zins

Eingeloggt als Mustermann Ausloggen

Meine KontenMeine DatenMeine TANAdministrationProtokolleo/ZBlick

ZEKonto

ZEKonto Editieren

OZB-Konto

Konto-Nr.:	10001
wSaldo:	234234.0
pSaldo:	234234
SaldoDatum:	2011-12-16

ZE-Konto

Konto-Nr.:	10001
ZE-Nr.:	2443623
EE-Konto-Nr.:	50001
AbDatum:	2011-12-24
EndDatum:	<input type="text" value="2011-12-24"/>
Betrag:	<input type="text" value="231425.0"/>
Laufzeit:	<input type="text" value="12"/>
Zahlmodus:	<input type="text" value="1"/>
Tilgungs-Rate:	<input type="text" value="234.0"/>
Anspar-Rate:	<input type="text" value="435.0"/>
Kdu-Rate:	<input type="text" value="21.0"/>
Rdu-Rate:	<input type="text" value="13.0"/>
Status:	AAA

Abbrechen

Speichern

Das Bearbeiten wird wiederum durch ein Eingabeformular ermöglicht.

3.3 Bürgschaften (MB)

Jede Zusatzentnahme wird durch eine Bürgschaft abgesichert.

Um diese zu verwalten, wurde der Menüpunkt "Administration" um einen zusätzlichen Punkt erweitert.

Hier werden alle Bürgschaften aufgelistet und können mit einem Klick angelegt, editiert oder gelöscht werden.

Sowohl das Anlegen, als auch das Editieren einer Bürgschaft wird mit Hilfe eines Eingabeformulars erledigt. Auch hier wird der Benutzer durch Eingabehilfen, wie die der Autovervollständigung und eigens angefertigten Suchmasken, unterstützt.



Besser ohne / Zins Eingelogg als Mustermann Ausloggen

Meine Konten Meine Daten Meine TAN Administration Protokolle o/ZBlick

Bürgschaften

Bürgschafter	Gesellschafter	Konto-Nr.	sichAbDatum	sichEndDatum	sichBetrag	sichKurzBez	
Neuer, Manuel	Kahn, Oliver	50001	2011-12-23	2011-12-31	200.0		Editieren Löschen

[Zurück](#) [Neue Bürgschaft hinzufügen](#)

Anlegen

Besser ohne / Zins

Eingeloggt als Mustermann [Ausloggen](#)

[Meine Konten](#) [Meine Daten](#) [Meine TAN](#) [Administration](#) [Protokolle](#) [o/ZBlick](#)

Bürgschaft

Neu anlegen

Buergschafter *:	<input type="text"/>	Suchen...
Gesellschafter *:	<input type="text"/>	Suchen...
Konto-nr. *:	<input type="text"/>	Suchen...
Sichabdatum:	<input type="text"/>	
Sichenddatum:	<input type="text"/>	
Sichbetrag:	<input type="text"/>	
Sichkurzbez:	<input type="text"/>	

[Abbrechen](#)[Ok](#)

Bearbeiten

Besser ohne / Zins

Eingeloggt als Mustermann [Ausloggen](#)

[Meine Konten](#) [Meine Daten](#) [Meine TAN](#) [Administration](#) [Protokolle](#) [o/ZBlick](#)

Bürgschaft

Editieren

Bürgschafter:	Neuer,Manuel	
Gesellschafter:	Kahn,Oliver	
Sichabdatum:	2011-12-23	
Konto-nr. *:	<input type="text" value="50001"/>	Suchen...
Sichenddatum:	<input type="text" value="2011-12-31"/>	
Sichbetrag:	<input type="text" value="200"/>	
Sichkurzbez:	<input type="text" value="A"/>	

[Abbrechen](#)[Ok](#)

3.4 Kontenklassenverwaltung (MB)

Kontenklassen werden benötigt, um ein Konto für die Berechnung der Punkte einordnen zu können. Ein Konto der Klasse 1 (ehemals A) bekommt bei einer von dem Mitglied getätigten Einzahlung von 100€, dasjenige Mitglied 100 Punkte gutgeschrieben. Das bedeutet, dass 100% der getätigten Einzahlung in Punkte umgewandelt werden. Für ein Konto der zweiten Klasse (ehemals B) bekommt das Mitglied nur noch 75% der Punkte gutgeschrieben, für ein dritte Klasse Konto (ehemals C) nur noch 50%.

Die Kontenklassenverwaltung erreicht man über den Menüpunkt "Administrator". Die Verwaltung wird durch die Rechteverwaltung vor unbefugten Zugriff geschützt. Hier hat man dann die Möglichkeit Kontenklassen anzulegen, bearbeiten oder zu löschen.

Besser ohne / Zins Eingeloggt als Mustermann Ausloggen

Meine Konten Meine Daten Meine TAN Administration Protokolle o/ZBlick

Kontenklassen

Name	Prozent	AbDatum		
1	100.0	2011-12-14	Editieren	Löschen
2	75.0	2011-12-14	Editieren	Löschen
3	50.0	2011-12-14	Editieren	Löschen
4	25.0	2011-12-23	Editieren	Löschen

Zurück Neue Kontenklasse hinzufügen

Anlegen

Besser ohne / Zins

Eingeloggt als Mustermann [Ausloggen](#)

[Meine Konten](#)
[Meine Daten](#)
[Meine TAN](#)
[Administration](#)
[Protokolle](#)
[o/ZBlick](#)

Kontoklasse

Neu anlegen

Kontenklasse:	<input type="text"/>
Prozent:	<input type="text" value="0.0"/>
Abdatum:	<input type="text"/>

Abbrechen

Ok

Bearbeiten

Besser ohne / Zins

Eingeloggt als Mustermann [Ausloggen](#)

[Meine Konten](#)
[Meine Daten](#)
[Meine TAN](#)
[Administration](#)
[Protokolle](#)
[o/ZBlick](#)

Kontoklasse

Editieren

Kontenklasse:	<input type="text" value="4"/>
Prozent:	<input type="text" value="25"/>
Abdatum:	<input type="text" value="2011-12-23"/>

Abbrechen

Ok

3.5 Reports (MB)

Reports dienen einer schnellen Übersicht aller Objekte einer Domäne, wie z.B. Adressen, Konten, Personen usw.. Außerdem können hier schnell viele Informationen ausgedruckt werden, oder z.B. einer Mailingliste/Adressliste dienen. In Zukunft sollen hier auch individuelle Kontoauszüge generiert werden können, des Weiteren sollen Suchmasken oder Sortierfunktionen zum Einsatz kommen.

Alle Reports findet man unter dem Menüpunkt "Administration". Reports sind durch die Rechteverwaltung vor unbefugten Zugriff geschützt. Zuweilen sind folgende Reports implementiert.

Konten

Besser ohne / Zins

Eingeloggt als Mustermann Ausloggen

Meine Konten

Meine Daten

Meine TAN

Administration

Protokolle

o/ZBlick

Alle Konten

Konto-Nr	Mitglieds-Nr	Konto-Einrichtungsdatum	wSaldo	pSaldo	Saldo Datum
10001	1		234234.0	234234	2011-12-16
10002	2		10000.0		
10008	8		2354234.0	234234	2011-12-23
21541	1	2011-12-21	213512.0	213214	2011-12-23
50001	1		520.21		

zurück

1

2

weiter

Zurück

Adressen

Besser ohne / Zins

Eingeloggt als MustermannAusloggen

Meine Konten

Meine Daten

Meine TAN

Administration

Protokolle

o/ZBlick

Alle Adressen

Name	Vorname	Straße	Hausnummer	Postleitzahl	Ort
Mustermann	Max	Musterstraße	42	76131	Karlsruhe
Mueller	Hermann				
Müller	Lieschen				
Neuer	Manuel				
Kahn	Oliver				

zurück

1


2

weiter

Zurück

3.6 Aufbau der Web-Applikation (ML)

3.6.1 Login



The screenshot shows a web application interface for 'Besser ohne / Zins'. The header is dark with the logo 'Besser ohne / Zins' in white and red, and a red 'Einloggen' button in the top right corner. The main content area has a light blue background. In the center, there is a white rounded rectangle containing the login form. The form is titled 'Anmelden' and includes two input fields: 'Mitgliedsnummer' (containing the number '1') and 'Passwort' (containing six asterisks). Below these fields is a checkbox labeled 'Angemeldet bleiben' and a green 'Einloggen' button. At the bottom of the form, there is a blue link that says 'Passwort vergessen?'.

Der Login ist das erste, was der User sieht. Man erkennt ein einfaches Eingabeformular zum einloggen. Es wird die Mitgliedsnummer, sowie das dazugehörige Passwort verlangt. Sollte man sein Passwort vergessen haben, hilft der Link "Passwort vergessen?" weiter. Es besteht des Weiteren die Möglichkeit, automatisch angemeldet zu werden, sobald man die Webseite aufruft.

3.6.3 Dashboard/Grundlayout



Grundlayout

Ziel ist es gewesen, ein schlichtes und leicht verständliches Grundlayout (Application Layout) zu implementieren. Dies ist durch eine simple, horizontale Trennung von Navigation und Content gelungen.

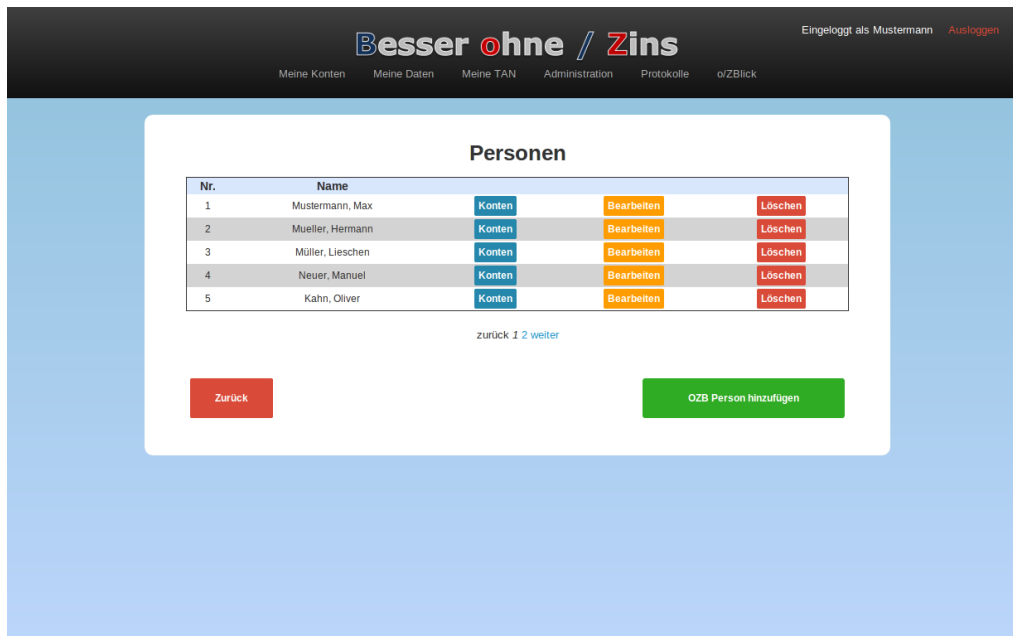
Die Navigation befindet sich im oberen Bereich, sie wird durch einen simplen schwarzen Balken vom Inhalt (Content) getrennt. In diesem befindet sich der Schriftzug der o/ZB und die Menüpunkte:

- Meine Konten
- Meine Daten
- Meine TAN
- Administration
- Protokolle
- o/ZBlick

Oben Rechts wird der Nachname des eingeloggten Users angezeigt, sowie die Möglichkeit zum Ausloggen angeboten.

Der Contentbereich nimmt den restlichen Teil des Bildschirms ein. Dieser wird durch seinen weißen Hintergrund farblich vom blauen Hintergrund hervorgehoben. Über diesem Bereich werden dem User Informationen/Feedback in einem gelben Kasten angezeigt. Dieser Kasten wird angezeigt, sobald eine Information vorhanden ist - z.B. nach einem erfolgreichen Login. Dann wird dies dem User mitgeteilt, oder wenn der User erfolgreich ein Konto angelegt hat. Somit bekommt der User nach jeder Tätigkeit ein passendes Feedback.

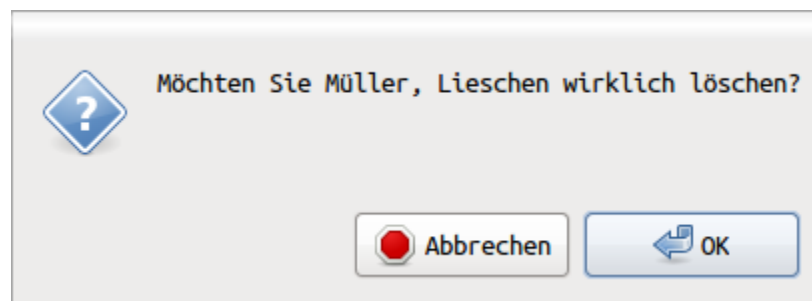
Aufgrund der Zielgruppe (=Anwender) dieser Anwendung, wurde ein Layout gewählt, das die Aktionen farblich stark hervorhebt.



Hier erkennt man die vorhandenen Aktionen sehr deutlich.

Zudem geben die Farben der einzelnen Buttons Aufschluss darüber, ob die sich dahinter verborgenden Aktionen kritisch sind. Dies ist zum Beispiel sehr gut an dem "Löschen"-Button zu erkennen.

Bei jeder kritischen Aktion, wie z.B. Löschen, wird noch einmal nachgefragt, ob man dies wirklich tun möchte.



Hinweis auf einen kritischen Vorgang: Löschen

Dashboard (Meine Konten)

Das Dashboard soll dem User eine grobe Übersicht seiner Konten und Neuigkeiten bieten. In dieser Version wird es nur eine Kontenübersicht geben.

Weitere Möglichkeiten wären administrative Informationen, z.B. User-Anfragen oder einem User anzuzeigen, dass er eine Rate verpasst hat - oder, dass er weniger als noch 3 aktive TAN-Nummern besitzt. Ein Torten-Diagramm, welches ihm seine Schulden und beglichende Schulden anzeigt, wäre auch denkbar.

3.6.4 Meine Daten

Besser ohne / Zins Eingelogg als Mustermann Ausloggen

Meine Konten Meine Daten Meine TAN Administration Protokolle o/ZBlick

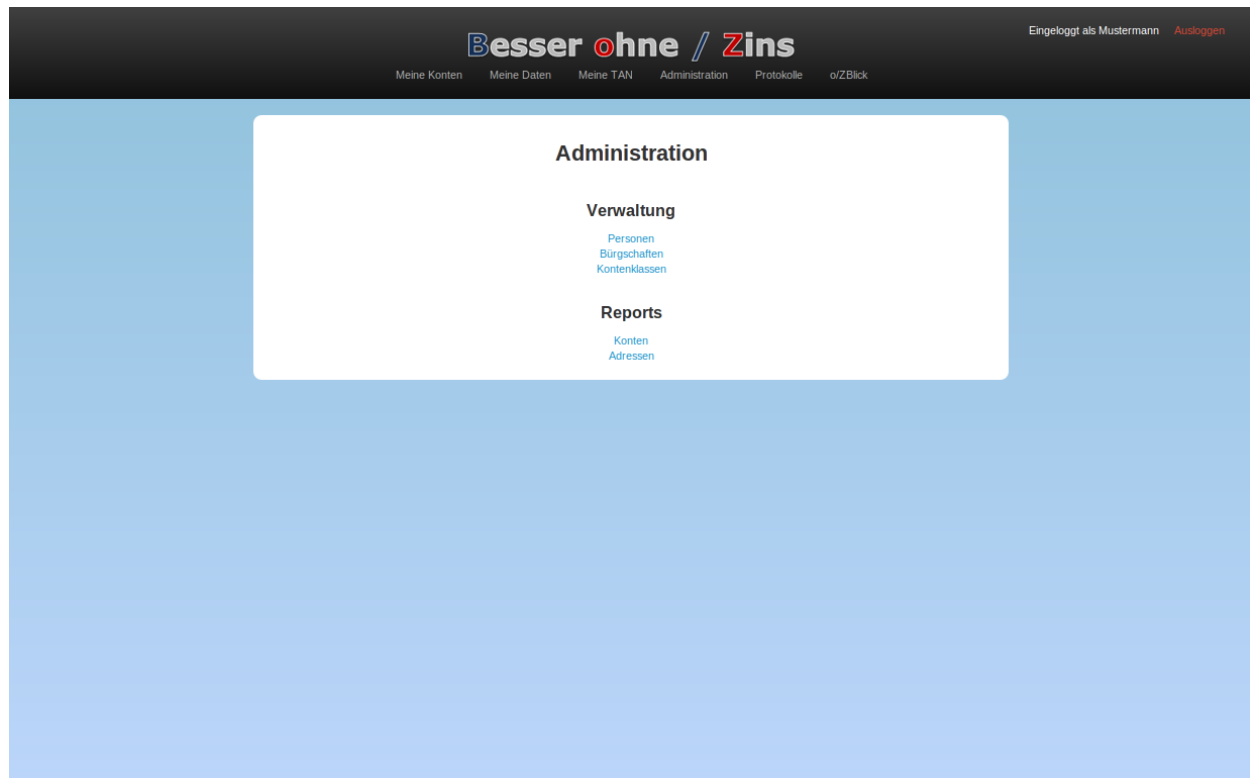
Person Bearbeiten

Person-nr.:	1
Name:	Mustermann
Vorname:	Max
Email:	person1@ozb.de
Passwort:	*****
Geburtsdatum:	1984-12-15
Straße:	Musterstraße
Hausnummer:	42
Plz:	76131
Ort:	Karlsruhe
Telefon:	0213259201
Fax:	021325920101
Rolle:	Mitglied
Rv datum:	2010-12-12
Antragsdatum *:	2010-12-10
Aufnahmedatum:	2010-12-12

Zurück Ok

Jeder User, bzw. o/ZB-Person, kann seine eigenen Stammdaten anpassen. Dies ist zum Beispiel bei einem Umzug der Fall. So werden o/ZB-Facharbeiter entlastet. Eine Seite, die dies ermöglicht, ist somit unbedingt erforderlich. Es wird ein schlichtes Eingabeformular angezeigt, indem die veränderbaren Felder weiß hinterlegt sind.

3.6.5 Administration



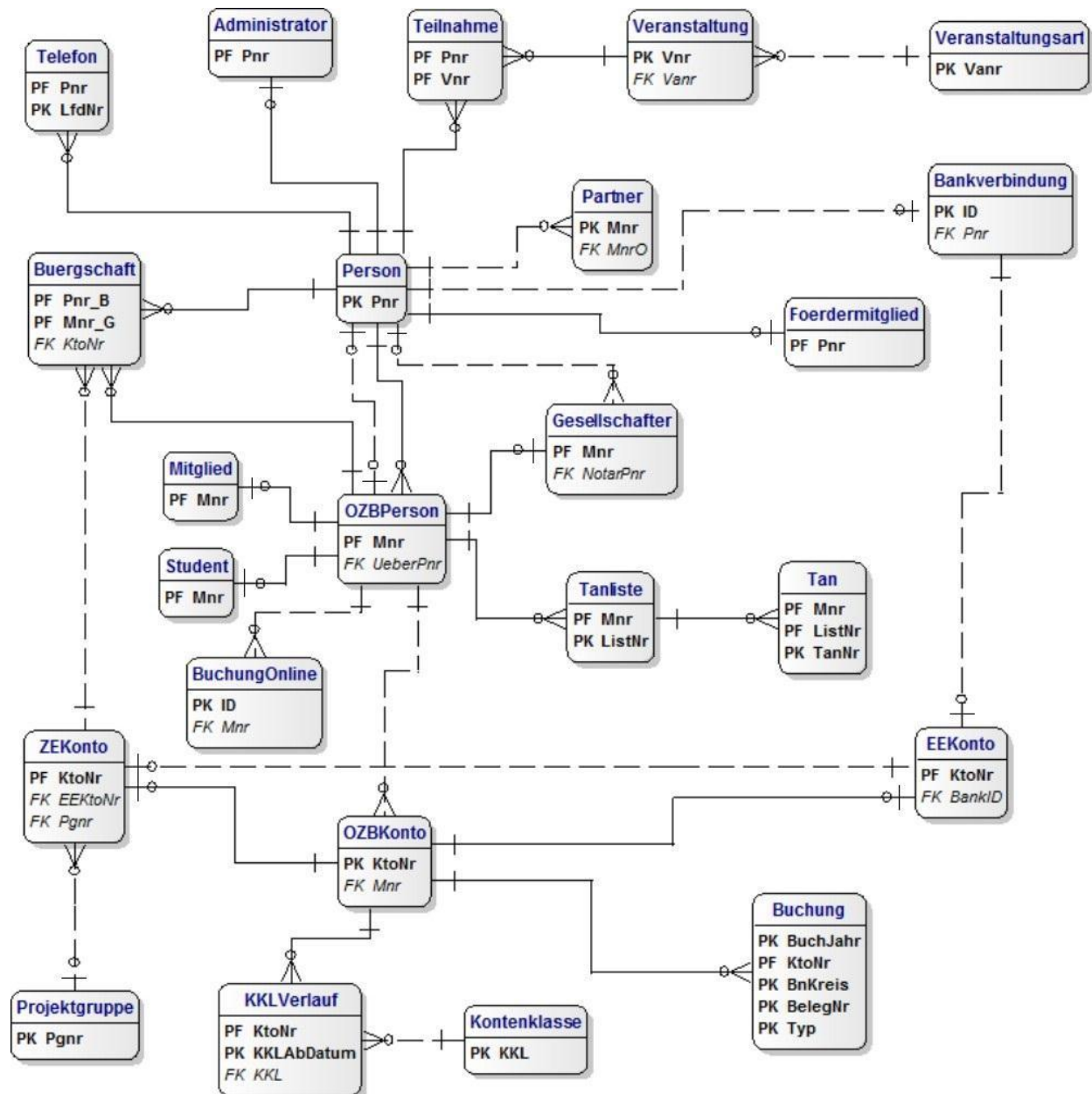
Die Administration entspricht dem Backend-Tool. Hier werden einem berechtigten User fein granular seine Möglichkeiten zur Administration aufgelistet (siehe dazu 4.8 Rechteverwaltung). Was sich hinter jedem einzelnen Link verbirgt, ist in den vorherigen Kapiteln nachzulesen. Hier sollen in Zukunft alle weiteren möglichen Geschäftsprozesse aufgelistet werden, für die erweiterte Rechte erforderlich sind.

4. Implementierungsdetails

Hierbei wird gezeigt wie einzelne Fälle von Geschäftsprozessen implementiert worden sind, sowie Besonderheiten z.B. von Ruby-Gems oder auch jQuery Erweiterungen.

4.1 Datenmodell/Model (ML/MB)

In diesem Projekt wurde das von Herrn Okunevych vorgestellte Datenmodell implementiert.



ER-Diagramm des nun aktuellen Datenmodells

In Ruby werden so genannte Migrationen geschrieben, um ein vorgegebenes Schema zu implementieren. Diese Migrationen findet man unter *db/migrate*. Jede einzelne Tabelle wird hier angelegt.

```
class CreateTeilnahme < ActiveRecord::Migration
  def self.up
    create_table(:Teilnahme, :id => false) do |t|
      # Pnr PS1, FS
      t.integer :pnr, :null => false, :limit => 10
      # Vnr PS2, FS
      t.integer :vnr, :null => false, :limit => 5
      # Teilnahmeart
      t.column "teilnArt", :enum, :limit => [:a, :e, :u]
      # a: anwesend, e: entschuldigt, u: unentschuldigt
    end
  end

  def self.down
    drop_table :Teilnahme
  end
end
```

Ausschnitt aus der Migration einer Teilnahme (20111014093338_create_teilnahme.rb)

Anschließend werden die dazugehörigen Klassen geschrieben, die die Datenbank-Tabellen abstrahieren. Diese findet man unter *app/models*.

Ruby verfolgt das Prinzip der Namenskonventionen, da diese nicht immer eingehalten werden konnten, ist folgende Zeile notwendig.

```
set_table_name "Foerdermitglied"
```

Mit Hilfe dieser Zeile teilt man mit, hier der Klasse Foerdermitglied, welche Tabelle aus der Datenbank mit dieser Klasse in Verbindung steht.

In der Klassendatei werden auch die verschiedensten Beziehungstypen definiert. Da auch hier nicht die Namenskonventionen von Ruby eingehalten werden können, sind auch hier extra Anpassungen nötig gewesen, um die Verbindung der einzelnen Klassen untereinander herzustellen. Also ist es nötig, den Fremdschlüssel extra anzugeben. Dies ermöglicht folgende Zeile.

```
has_many :teilnahme, :foreign_key => :vnr
```

In diesem Beispiel ist das Attribut "vnr" der Fremdschlüssel in der Tabelle "Teilnahme", um eine Teilname einer Veranstaltung zuordnen zu können.

So sind im Laufe der Zeit, für jede einzelne Tabelle, jeweils eine Migration und eine Klasse entstanden.

4.2 Controller

Die Controller befinden sich in dem Verzeichniss *app/controllers*.

4.2.1 OZB Person (ML)

Im OZB Person Controller wird das Rechteflag A benutzt. Dieses schränkt ein, ob ein User Daten einsehen, bearbeiten oder löschen kann.

4.2.1.1 index

Mit dem Index-Aufruf werden alle OZB Personen aufgelistet, mit Unterstützung des Gems *will_paginate*.

4.2.1.2 new/create

Um eine neue Person anzulegen, wird im Abschnitt "new" die nötigen Rollen und die Personen-Suche geladen. Im "Create"-Bereich wird die Person, OZB Person, Telefone (Festnetz/Mobil/Fax) und die jeweilige Rolle (Mitglied, Fördermitglied, Partner, Gesellschafter, Student) erstellt. Ist während dem Erstellen der einzelnen Datensätze ein Fehler aufgetreten, wird dieser angezeigt und der Speichervorgang abgebrochen, ansonsten wird die Person angelegt und in der Übersicht angezeigt.

4.2.1.3 edit/update

Zum Editieren einer Person wird mittels des GET-Parameters *id* die Mitgliedsnummer der Person übergeben, dadurch können alle Informationen der Person geladen werden, sprich Person, OZB Person, Telefone und Rolle. Ebenso wird die Personen-Suche geladen. Es gibt noch eine Besonderheit, die Person selbst kann sich selbst editieren, jedoch nur die persönlichen Daten. Dies wird überprüft, indem die angemeldete Person mit der zu bearbeitenden Person verglichen wird. Beim speichern einer Änderung wird der "Update"-Bereich aufgerufen. Hier werden alle Felder per POST übergeben und in den jeweiligen Datensatz gespeichert.

4.2.1.4 delete

Um eine Person zu löschen ist es nur nötig, die jeweilige Person per ID zu bestimmen und diese zu "destroyen". Im Gegensatz zu einem delete, welches nur den einzelnen Datensatz löscht, werden bei einem destroy zusätzlich alle Beziehungen gelöscht.

4.2.2 Application (ML/MB)

Der Application Controller fungiert in diesem Projekt als Basis-Klasse für Methoden, die öfter benötigt werden. Von dieser Klasse erben alle weiteren Controller (Klassen). Somit verhindert man unnötige Redundanzen. Bisher werden zwei Methoden, die sich um den Inhalt der beiden Suchmasken (Kontonummer und OZB Person) kümmern, implementiert. In dieser Klasse werden folgenden Methoden programmiert:

4.2.2.1 searchKtoNr

Diese Methode durchsucht die Tabelle OZBKonto nach einem Konto.
Die möglichen Suchparameter werden via POST-Request übergeben:

- ktoNr
- ktoEinrDatum
- mnr
- name
- vorname

Alle Konten, die den Suchparametern entsprechen, werden mit Hilfe von will_paginate gruppiert (= in gleich große Gruppen/Seiten geteilt) zurückgeliefert.

4.2.2.2 searchOZBPerson

Diese Methode durchsucht die Tabelle OZBPerson nach einer o\ZB Person.
Die möglichen Suchparameter sind:

- mnr
- pnr
- rolle
- name
- ktoNr

Auch hier wird will_paginate benutzt, um die zutreffenden Datenenelemente gruppiert wiederzugeben.

4.2.3 Index (ML/MB)

Der Index-Controller stellt das Dashboard bzw. den Menüpunkt "Meine Konten" dar.
Der Funktionsumfang besteht zur Zeit aus der Initialisierung der View *index/dashboard.html*.

```
def dashboard
  @current_person = Person.find(current_OZBPerson.mnr)
end
```

Die Methode "admin" wird benötigt, um den Menüpunkt "Administrator" zu rendern.
Die letzte Methode "error_404" dient dazu dem User einen Hinweis zu geben, dass die noch nicht implementierte Seiten:

- Meine TAN
- Protokolle
- o/ZBlick

nicht vorhanden bzw. verfügbar sind.

4.2.4 Bürgschaft (MB)

Der Bürgschaft Controller behandelt alle Geschäftsvorfälle die mit dem anlegen, bearbeiten und löschen von Bürgschaften zu tun haben. Konkret wird hier B.3 (s. 1.1) umgesetzt.
Um Bürgschaften verwalten zu können, ist das Rechteflag "canEditB" erforderlich.
Im Folgenden werden alle Methoden aufgelistet und genauer beschrieben. Aufgrund der Tatsache, dass sich die nachfolgenden Controller von der Struktur und Funktionsweise sehr stark ähneln und um Redundanz zu vermeiden, wird darauf verzichtet, diese nochmals detailliert zu beschreiben. Es werden daher nur Besonderheiten erwähnt.

4.2.4.1 index

Hier werden alle Bürgschaften aufgelistet. Damit diese von der korrespondierenden View auch dargestellt werden können, werden alle vorhandenen Bürgschaften aus der Datenbank geladen.

```
@buergschaften = Buergschaft.paginate(:page => params[:page], :per_page => 5)
```

4.2.4.2 new

Diese Methode bereitet eine neue, leere Instanz einer Bürgschaft für die dazugehörige View (*views/buergschaft/index*) vor. Außerdem werden die Methoden “searchKtoNr” und “searchOZBPerson” aufgerufen, um die beiden dazugehörigen Suchmasken zu initialisieren.

```
@buergschaft = Buergschaft.new
searchKtoNr()
searchOZBPerson()

@tempNames = Array.new
@tempNames.push("")
@tempNames.push("")
```

4.2.4.3 searchKtoNr

Für den Fall, dass eine Bürgschaft editiert wird, muss diese zuerst geladen werden. Danach wird ganz normal die Suchmaske für ein Konto initialisiert.

```
if( !params[:pnrB].nil? && params[:pnrB].to_i > 0 && !params[:mnrG].nil? &&
    params[:mnrG].to_i > 0) then
  @buergschaft = Buergschaft.find([params[:pnrB], params[:mnrG]] )
end
```

4.2.4.4 searchOZBPerson

Diese Methode beinhaltet lediglich den Aufruf “super”, der, wie in Java, die Methode der “Mutterklasse” aufruft. Sie dient dazu, die Suchmaske für eine o/ZB Person zu initialisieren.

4.2.4.5 edit

Hier wird die, mit dieser Methode verknüpften, View initialisiert.

```
searchKtoNr()

@tempNames = Array.new
@tempNames.push("")
@tempNames.push("")
```

4.2.4.6 create

Um eine Bürgschaft neu anzulegen, wird diese Methode aufgerufen. Der Vorgang ist wie folgt: Zuerst wird eine neue Bürgschaft mit den vorhandenen Parametern angelegt.

```
@buergschaft = Buergschaft.new(params[:buergschaft])
```

Anschließend wird die Validierung ausgeführt.

```
@errors = @buergschaft.validate(params[:pnrBName], params[:mnrGName])
```

Falls es Eingabefelder gibt, die nicht zur Bürgschaft gehören, aber unbedingt zwischengespeichert werden müssen (um den Verlust der Daten bei einer fehlgeschlagenen Validierung zu vermeiden), wird ein Feld angelegt um diese zwischenzuspeichern.

```
@tempNames = Array.new
@tempNames.push(params[:pnrBName])
@tempNames.push(params[:mnrGName])
```

Nun wird entschieden, welche View dargestellt wird. Falls die Validierung fehlgeschlagen ist, wird nochmals die View *new.html.erb* aufgerufen. Im Falle einer erfolgreichen Validierung, wird die Bürgschaft in die Datenbank gespeichert und es wird zur Bürgschaft-Übersicht (index) weitergeleitet. Diese wird auch initialisiert.

```
if !@errors.nil? && @errors.any? then
  searchKtoNr()
  searchOZBPerson()
  render "new"
else
  @buergschaft.save
  @buergschaften = Buergschaft.all
  redirect_to :action => "index", :notice => "Bürgschaft erfolgreich angelegt."
end
```


4.2.4.7 update

Die "update" Methode wird aufgerufen, um eine bereits vorhandene Bürgschaft zu aktualisieren. Sie wird aus der View *view/buergschaft/edit.html.erb* aufgerufen.

Der Aktualisierungs-Prozess läuft wie folgt ab:

Die zu aktualisierende Bürgschaft wird anhand der Parameter pnrB (Personalnummer Bürger) und mnrG (Mitgliedsnummer Gesellschafter) herausgesucht.

```
@buergschaft = Buergschaft.find([params[:pnrB], params[:mnrG]])
```

Im Anschluss werden die neuen Werte der Attribute mit Hilfe der Parameter übergeben.

```
@buergschaft.attributes = params[:buergschaft]
```

Danach wird die Validierung durchgeführt.

```
@errors = @buergschaft.validate(params[:pnrBName], params[:mnrGName])
```

Auch hier gilt, falls es Eingabefelder gibt, die nicht zur Bürgschaft gehören, aber unbedingt zwischengespeichert werden müssen, wird ein Feld angelegt um diese zwischenzuspeichern.

```
@tempNames = Array.new
@tempNames.push(params[:pnrBName])
@tempNames.push(params[:mnrGName])
```

Am Ende wird entschieden, welche View gerendert werden soll.

Falls die Validierung fehlgeschlagen ist, wird nochmals die View *edit.html.erb* angezeigt. Im Falle einer erfolgreichen Validierung, wird die aktualisierte Bürgschaft in die Datenbank gespeichert und es wird zur Bürgschaft-Übersicht (index) weitergeleitet. Diese wird auch initialisiert.

```
if !@errors.nil? && @errors.any? then
  searchKtoNr()
  searchOZBPerson()
  render "edit"
else
  @buergschaft.update_attributes(params[:buergschaft])
  redirect_to :action => "index", :notice => "Bürgschaft erfolgreich aktualisiert."
end
```

4.2.4.8 delete

Hat man sich dazu entschieden, eine Bürgschaft zu löschen, wird diese Methode aufgerufen. Zuerst wird die zu löschende Bürgschaft mit Hilfe der Parameter ausgemacht und mittels der delete Methode aus der Datenbank entfernt. Hier wäre die Methode "destroy" unangebracht, da diese sonst auch die mit dieser Bürgschaft in Beziehung stehenden Objekte aus der Datenbank entfernen würde. Da die Konten und Personen weiterhin bestehen bleiben sollen, wird die Methode "delete" verwendet.

```
@buergschaft = Buergschaft.find([params[:pnrB], params[:mnrG]])
@buergschaft.delete
```

Anschließend wird nach dem erfolgreichen Entfernen des Datensatzes auf die Übersichtsseite weitergeleitet.

```
@buergschaften = Buergschaft.all
redirect_to :action => "index"
```

4.2.5 Kontenklasse (MB)

Dieser Controller kümmert sich um alle Geschäftsvorfälle die direkt mit der Bearbeitung und Erstellung von Kontenklassen zu tun haben. Dieser Controller erbt auch von der Klasse ApplicationController, jedoch kommen hier bislang noch keine Methoden aus dieser Basisklasse zum Einsatz. Der Funktionsumfang ist der gleiche wie beim zuvor vorgestellten Controller der Bürgschaft. Es ist möglich eine Kontenklasse anzulegen, zu editieren und zu löschen. Zudem gleicht, wie bereits angesprochen, die Struktur und Funktionsweise der des Bürgschaft-Controllers. Die Methoden

- index
- new
- edit
- create
- update
- delete

sind auch hier enthalten und funktionieren so, wie bereits beim Bürgschafts-Controller beschrieben.

4.2.6 OzbKonto (MB)

Der OzbKonto-Controller kümmert sich konkret um die Geschäftsvorfälle B.1 und B.2 (s. Kapitel 1.1). Darüber hinaus lässt sich auch die Klasse eines Kontos verändern.

Die Struktur und Funktionsweise ähnelt auch hier, dem bereits ausführlich vorgestellten, Bürgschaft-Controller:

- index
- get_konten
- new
- create
- update
- edit
- show
- delete

- searchKtoNr
- kkl
- changeKKL

Es gibt jedoch ein paar Besonderheiten, die im Folgenden vorgestellt werden. Die Herausforderung war, sauber zwischen E/E-Konten und ZE-Konten zu trennen. Dies zieht sich durch den gesamten Controller.

4.2.6.1 get_konten

In dieser Methode werden alle Konten einer o/ZB Person ausgelesen und nach ihrer Zugehörigkeit (E/E, ZE) in 2 verschiedenen Arrays sortiert, um eine fehlerlose Ausgabe zu gewährleisten.

```
def get_konten
  @ozb_konten = OZBKonto.where( :mnr => params[:id] )
  @ee_konten = Array.new
  @ze_konten = Array.new

  @ee_count = 0
  @ze_count = 0
  @ozb_konten.each do |konto|
    if konto.EEKonto.count > 0 then
      @ee_konten.push(konto.EEKonto.first)
      @ee_count += 1
    end
    if konto.ZEKonto.count > 0 then
      @ze_konten.push(konto.ZEKonto.first)
      @ze_count += 1
    end
  end
end
```

Da dies sehr häufig geschehen muss, wurde diese Funktionalität in einer Methode ausgegliedert.

4.2.6.2 kkl

Hier wird die View *ozb_konto/kkl.html.erb* initialisiert, die sich um die Änderung von Zugehörigkeiten eines Kontos mit einer Kontenklasse kümmert.

```
def kkl
  if current_OZBPerson.canEditB then
    @konten = OZBKonto.paginate(:page => params[:page], :per_page => 5)
  else
    redirect_to "/"
  end
end
```

4.2.6.3 changeKKL

Diese Methode geht jedes, per Parameter übergebene, Konto durch und legt einen neuen Kontenverlauf (KKLVerlauf) mit aktualisierter Kontenklasse an.

```
def changeKKL
  if current_OZBPerson.canEditB then
    i = 0
    params[:kk1].each do |kk1|
      @verlauf = KKLVerlauf.create( :ktoNr => params[:ktoNr][i],
                                   :kk1AbDatum => Time.now, :kk1 => kk1 )
      i += 1
    end
    redirect_to :action => "kk1"
  else
    redirect_to "/"
  end
end
```

4.2.6.4 show

Mit Hilfe dieser Methode soll ein einzelnes Konto angezeigt werden. Da in E/E-Konten und ZE-Konten unterschieden wird, ist eine entsprechende if-Abfrage notwendig.

```
def show
  @ozb_konto = OZBKonto.where( :ktoNr => params[:ktoNr] ).first
  if params[:typ] == "EE" then
    @konto = @ozb_konto.EEKonto.first
  end

  if params[:typ] == "ZE" then
    @konto = @ozb_konto.ZEKonto.first
  end

  render "show.html.erb"
end
```

4.2.6.5 new

Zusätzlich wird für je ein E/E-Konto bzw. ein ZE-Konto eine Kontonummer generiert, um dem User eine Kontonummer vorzuschlagen.

Am Beispiel eines E/E-Kontos sieht dies wie folgt aus.

```
if params[:typ] == "EE" then
  count = 0
  @ozb_konten.each do |konto|
    if konto.EEKonto.count > 0 then
      count += 1
    end
  end

  # Neue Kontonummer erzeugen
  @newKtoNr = (count + 5).to_s
  (1..(4-@person.pnr.to_s.length)).each do |i|
    @newKtoNr += "0"
  end
  @newKtoNr += @person.pnr.to_s

  render "new_ee.html.erb"
end
```

4.2.6.6 create

Wenn ein Konto angelegt wird, wird nicht nur ein weiterer Datensatz in der Tabelle OZBKonto angelegt, sondern noch weitere. Im Falle eines E/E-Kontos wird zudem eine neue Bankverbindung, ein neuer Kontenverlauf und dazu ein E/E-Konto angelegt.

```
# Neuen Kontenverlauf hinzufügen
@verlauf = KKLVerlauf.new( :ktoNr => params[:ozb_konto][:ktoNr],
                          :kklAbDatum => Time.now, :kkl => params[:kkl] )
@errors.push(@verlauf.validate!)

# EE Konto
if params[:typ] == "EE" then
  # Bankverbindung: id, :pnr, :bankKtoNr, :blz, :bic, :iban, :bankName
  @bankverbindung = Bankverbindung.new( :pnr => params[:id],
                                         :bankKtoNr => params[:bankKtoNr], :blz => params[:blz],
                                         :bic => params[:bic], :iban => params[:iban],
                                         :bankName => params[:bankName] )
  @errors.push(@bankverbindung.validate!)

  # EE-Konto: :ktoNr, :bankId, :kreditlimit
  @ee_konto = EEKonto.new( :ktoNr => params[:ozb_konto][:ktoNr], :bankId => 1,
                           :kreditlimit => params[:kreditlimit] )
  @errors.push(@ee_konto.validate!)
end
```

Auch hier werden alle angelegten Objekte validiert und am Ende wird die Validierung ausgewertet.

4.2.6.7 update

Die Besonderheit besteht darin, dass hier zwischen einem E/E-Konto und einem ZE-Konto unterschieden werden muss. Je nachdem müssen die verschiedensten Attribute neu eingelesen, validiert und in der Datenbank aktualisiert werden.

```
if params[:typ] == "EE" then
  @ee_konto = @ozb_konto.EEKonto.first
  @bankverbindung = Bankverbindung.where( :id => @ee_konto.bankId ).first

  @ee_konto.kreditlimit = params[:kreditlimit]
  @errors.push(@ee_konto.validate!)

  @bankverbindung.bankKtoNr = params[:bankKtoNr]
  @bankverbindung.bankName = params[:bankName]
  @bankverbindung.blz = params[:blz]
  @bankverbindung.bic = params[:bic]
  @bankverbindung.iban = params[:iban]

  @errors.push(@bankverbindung.validate!)
end
```

Im Falle eines E/E-Kontos wird auch die Bankverbindung aktualisiert.

4.2.6.8 verlauf

Diese Methode zeigt den Kontenklassenverlauf eines Kontos in einer simplen Tabelle an.

```
# Zeigt den KKLVerlauf an
def verlauf
  @ozb_konto = OZBKonto.where( :ktoNr => params[:ktoNr] ).first
  @verlauf = @ozb_konto.KKLVerlauf
end
```

4.2.7 Reports (MB)

Diese Klasse bereitet alle bisher implementierten Reports auf.

```
def ozbKonten
  @ozbKonten = OZBKonto.paginate(:per_page => 5, :page => params[:page])
end
```

Lädt alle verfügbaren o/ZB-Konten für die View.

```
def adressen
  @personen = Person.paginate(:per_page => 5, :page => params[:page])
end
```

Lädt alle verfügbaren Adressen für die View.

4.3 View - RHTML/JQuery

Alle Views sind unter *app/views* zu finden.

4.3.1 OZB Person (ML)

4.3.1.1 Index

Hier gibt es zwei nennenswerte Eigenschaften, zum einen wird die Mnr per GET-Request an Edit und Delete übergeben und zum anderen nach dem Klick auf Löschen eine Bestätigung angefordert. Mittels der Option “:confirm” kann man eine Bestätigungsnachricht definieren, welche dann als JavaScript Popup erscheint.

```
<%= link_to "Löschen", delete_link, :class => "delete_link", :confirm => "Möchten Sie " +  
current_person.name + ", " + current_person.vorname + " wirklich löschen?" %>
```

Ausschnitt aus *app/views/ozb_person/index.html.erb*

4.3.1.2 New

Um nicht alle Rollen anzuzeigen, wurden die einzelnen Rollen als <div> definiert, damit diese ausgeblendet werden können. Über die JavaScript-Funktion *switch_role()* wird die gewählte Rolle über ein Dropdown-Feld eingeblendet.

4.3.1.3 Edit

Die Detail- und Bearbeitungsansicht werden innerhalb der selben Datei abgebildet. Da die selben Felder angezeigt werden sollen, war dies eine einfache Möglichkeit beide Ansichten darzustellen. Die Felder, die zum Bearbeiten freigegeben werden sollen, werden durch die Variable *@disabled* freigeschaltet.

4.3.1.4 searchOZBPerson

Die new- und edit-Seite beinhalten die Suche nach Personen. Die Suche wird für die Bestimmung des Notars und Partners eingesetzt. Dazu sind folgende Scripts nötig.

```
var insertId = "#notarPnr";

$("#searchNotarFrame").click(function(){
    insertId = "#notarPnr";
});

$("#searchPartnerFrame").click(function() {
    insertId = "#partner";
});

$(document).ready(function() {
    $("#searchNotarFrame").fancybox();
    $("#searchPartnerFrame").fancybox();
});

$(function() {
    var availableTagsB = [
        <% tags = String.new %>
        <% Person.all.each do |person| %>
        <% tags += "'" + person.pnr.to_s + "'" + "," %>
        <% end %>
        <%= tags.chop %>
    ];
    $( "#notarPnr" ).autocomplete({
        source: availableTagsB
    });
});
```

Ausschnitt aus app/views/ozb_person/new.html.erb

4.3.2 Allgemeiner Aufbau (ML/MB)

Anhand der Bürgschaft wird der allgemeine Aufbau einer View-Struktur erläutert. Da bei der Bürgschaft alle Elemente der verwendeten Programmier Techniken und Plugins vorkommen, ist dies ein repräsentatives Beispiel.

Wie bereits in dem Abschnitt über die Controller erwähnt, sind auch die verschiedenen Views isomorph. Es werden zudem einfache HTML-Seiten und Javascript-Dateien voneinander unterschieden.

4.3.2.1 HTML-Seiten

Index

Hier werden alle Bürgschaften mit Hilfe einer einfachen Tabelle aufgelistet. Zu beachten ist der Zusatz für das Gem "will_paginate".

```
<!-- Seitennavigation -->
<div class="paginateBox" >
  <%= will_paginate @buergschaften %>
</div>
```


Dazu wird für jede Bürgschaft jeweils ein Editier- und Löschen-Link erzeugt.

```
<% edit_link = "/buergschaften/" + buergschaft.pnrB.to_s + "/" + buergschaft.mnrG.to_s %>
<% delete_link = "/buergschaften/" + buergschaft.pnrB.to_s + "/" + buergschaft.mnrG.to_s +
"/delete" %>

<td><%= link_to "Editieren", edit_link, :class => "edit_link" %></td>
<td><%= link_to "Löschen", delete_link, :class => "delete_link", :confirm => 'Sind Sie
sicher?' %></td>
```

Am unteren Ende wird in den meisten Fällen eine weitere Tabelle mit verfügbaren Aktionen, wie einem Zurück-Button und einem Button der ein neues Element anlegt, dargestellt.

```
<table class="actions">
  <tr>
    <td><%= link_to "Zurück", "/Admin", :class => "cancel_link" %></td>
    <td><%= link_to "Neue Bürgschaft hinzufügen", "/buergschaften/new", :class => "submit_ok"
%></td>
  </tr>
</table>
```

new

Es wird in der Regel ein einfaches, in tabellarischer Form, dargestelltes Eingabeformular präsentiert. Zu beachten ist die Art und Weise, wie eine so genannte Form definiert wird.

```
<%= form_for @buergschaft, :url => { :action => "create" }, :html => {:id => "newForm"} do
|f| %>
```

Der Form wird die dazugehörige Controller-Action mitgeteilt. Außerdem wird auch hierbei der Objekt-Kontext festgelegt (@buergschaft). Des Weiteren wird der Form noch eine ID zugewiesen, die ein eindeutiges jQuery-Statement zulässt.

```
<td>
<%= link_to_function "Ok", "this.form.submit()", :onclick => "$('#newForm').submit();
return false;", :class => "submit_ok" %>
</td>
```

Ein einzelnes Feld wird wie folgt implementiert.

```
<tr>
  <td><b><%= label_tag "Buergschafter *:" %></b></td>
  <td><%= text_field_tag :pnrBName, @tempNames[0] %><%= f.hidden_field :pnrB, :id =>
"pnrB" %></td>
  <td class="searchlink_td"><%= link_to "Suchen...", "#searchOZBPersonFrame", :id =>
"searchBFrame", :class => "searchlink" %></td>
</tr>
```

Da aufgrund der Lesbarkeit der ausgeschriebene Name einer Person, anstatt der Personalnummer, zu favorisieren ist, wird das für die Datenbank relevante Feld "pnrB" als hidden-Field angelegt.

Neben den einfachen Eingabefeldern gibt es auch die angesprochen Suchmasken, die dem User bei der Eingabe von Daten helfen sollen. Diese werden wie folgt initialisiert.

```
<div style="display:none">
  <div id="searchKtoNrFrame">
    <div id="konten"><%= render 'application/suche_konten' %></div>
  </div>
</div>

<div style="display:none">
  <div id="searchOZBPersonFrame">
    <div id="personen"><%= render 'application/suche_ozb_personen' %></div>
  </div>
</div>
```

Somit werden diese in die HTML-Seite eingebunden.

Der Link zu einer Suchmaske ist ein einfacher Link, der eine bestimmte ID erhält.

```
<td class="searchlink_td"><%= link_to "Suchen...", "#searchOZBPersonFrame", :id =>
"searchGFrame", :class => "searchlink" %></td>
```

Damit die Suchmasken auch sichtbar werden, ist folgender jQuery-Code nötig.

```
$(document).ready(function() {
  $("#searchKtoNr").fancybox();
  $("#searchBFrame").fancybox();
  $("#searchGFrame").fancybox();
});
```

Dabei wird das jQuery-Plugin Fancybox genutzt, um durch einen Klick auf den dazugehörigen Button den Inhalt darzustellen. Fancybox wird in einem späteren Kapitel vorgestellt.

Aufgrund der Tatsache, dass es hier 2 Suchmasken vom Typ OZB-Personensuche gibt, muss man mit einem weiteren jQuery Statement festlegen, in welches DOM-Objekt das vom User angeklickte Ergebnis eingefügt werden soll.

```
$("#searchBFrame").click(function(){
    insertId = "#pnrB";
});

$("#searchGFrame").click(function(){
    insertId = "#mnrG";
});
```

Des Weiteren erfährt dabei die Autovervollständigung das jQuery Plugin autocomplete seine Verwendung. Zuerst werden die möglichen Tags initialisiert, dies geschieht in einem, von Ruby unterstütztem, jQuery Statement.

```
var availableTagsB = [
  <% tags = String.new %>
  <% Person.all.each do |person| %>
    <% tags += "'" + person.name.to_s + ", " + person.vorname.to_s + "'", " %>
  <% end %>
  <%= tags.chop %>
];

$( "#pnrBName" ).autocomplete({
  source: availableTagsB
});
```

Neben der Autovervollständigung und den Suchmasken gibt es noch eine weitere Eingabehilfe, speziell um ein Datum eintragen zu können, der Datepicker. Der Datepicker wird ebenfalls mit einem jQuery-Statement initialisiert.

```
$(function() {
  $( "#sichEndDatum" ).datepicker({
    dateFormat: 'yy-mm-dd'
  });
});
```

Darüber hinaus ist die Ausgabe der Validierung implementiert, die dem User die Fehler bei der Eingabe anzeigt.

```
<% if !@errors.nil? && @errors.any? %>
  <div id="error_explanation">
    <h2><%= @errors.count %> Fehler <br /> Konnte keine Bürgschaft anlegen, weil:</h2>
    <ul>
      <% @errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
    </ul>
  </div>
<% end %>
```

edit

Die Edit-View ist im Prinzip die gleiche, wie die New-View.

Unterschiede bestehen z.B. darin, dass das zu bearbeitende Objekt aus der Datenbank geladen und das Eingabeformular mit diesen Daten vorbelegt wird.

```
<% buergschafter = Person.where(:pnr => @buergschaft.pnrB).first %>
<% gesellschaft = Person.where( :pnr => OZBPerson.where(:mnr =>
@buergschaft.mnrG).first.ueberPnr ).first %>
```

Außerdem wird die ID, die Controller-Methode und die Methode der Übertragung geändert.

```
<%= form_for @buergschaft, :url => { :action => "update" }, :html => {:id => "editForm",
:method => "post"} do |f| %>
```

4.3.2.2 Javascript

In jeder Javascript Datei, zum Beispiel der *new.js.erb*, werden jQuery Statements zur Ersetzung von HTML-Content beschrieben. Diese werden benötigt, um die Suchmaske nach jeder Sucheingabe zu aktualisieren. Es wird dabei das zu ersetzende DOM-Element gewählt und mit Hilfe des ruby-Befehles "render" der neue Content berechnet.

Zu beachten ist, dass man den neuen Inhalt "escaped", sodass Anführungszeichen und andere kritische Zeichenketten korrekt dargestellt werden. Ansonsten kann entweder das Ergebnis aufgrund des String-Parsings beeinflusst oder verfälscht werden und eventuell zu einem Programmabbruch führen.

```
$('#personen').html('<%= escape_javascript(render("application/suche_ozb_personen")) %>');
```

Ausschnitt aus *buergschaft/new.js.erb*

4.4 Devise (ML)

Devise wird für die User-Authentifizierung und Session-Verwaltung eingesetzt. Hilfreich für das Einrichten und Verwenden von Devise war das Video-Tutorial auf railscasts.com¹².

4.4.1 Model-Änderung und Migration

Als User-Model wurde das bestehende Model `ozb_person.rb` verwendet. Hier mussten folgende Zeilen hinzugefügt werden.

```
# Include default devise modules. Others available are:
# :token_authenticatable, :encryptable, :confirmable, :lockable, :timeoutable
# and :omniauthable

devise :database_authenticatable, :registerable,
      :recoverable, :rememberable, :trackable, :validatable
```

Ebenfalls werden neue Felder für das Passwort und die E-Mail-Adresse benötigt.

```
attr_accessible :email, :password, :password_confirmation
```

Zusätzlich wird ein Migrations-File erstellt. Zu finden ist es unter `db/migrate/20111019110025_devise_create_o_z_b_person.rb`

4.4.2 Routes

In den Routes musste die Login-Tabelle Devise zugeordnet werden.

```
# Devise - Login Management
devise_for :OZBPerson
```

Dadurch werden die Routen fürs Login/Logout gesetzt. Zu sehen sind sie mit dem Befehl `rake routes`.

```
new_OZBPerson_session GET    /OZBPerson/sign_in(.:format)          {:action=>"new",
:controller=>"devise/sessions"}

OZBPerson_session POST   /OZBPerson/sign_in(.:format)
      {:action=>"create", :controller=>"devise/sessions"}

destroy_OZBPerson_session DELETE /OZBPerson/sign_out(.:format)
      {:action=>"destroy", :controller=>"devise/sessions"}
```

¹² <http://railscasts.com/episodes/209-introducing-devise>

4.4.3 User Login

Unter *app/views/devise/sessions/new.html.erb* ist die Login-Maske zu finden. Zur Authentifizierung war gewünscht, dass sich eine Person mit der Mitgliedsnummer und Passwort einloggen kann.

4.4.4 Lokalisierung

Um Devise auf Deutsch zu übersetzen, wurde eine weitere Lokalisierungsdatei erstellt. Zu finden ist diese unter *config/locales/devise.de.yml*.

Weitere Informationen zur Lokalisierung stehen unter 4.7.1.

4.5 jQuery

4.5.1 Datum-Popup (ML)

Um das Eintragen eines Datum zu vereinfachen wurde das jQuery-Feature *datepicker*¹³ benutzt. *datepicker* wird in fast jedem Datumsfeld eingesetzt.

Ein Beispiel.

```
$(function() {  
    $( "#studienbeginn" ).datepicker({  
        dateFormat: 'dd.mm.yy'  
    });  
});
```

¹³ <http://jqueryui.com/demos/datepicker/>

4.5.2 Such-Maske (MB)

Die Suchmaske soll den User bei der Eingabe von Kontonummern oder Personen/Mitglieds-Nummern unterstützen. Sie erlaubt es dem User mit Hilfe von Suchparametern sein gewünschtes Objekt in der Datenbank zu finden, um es dann anschließend in das Eingabeformular einzufügen.

Um dies zu bewerkstelligen, werden Techniken wie Ajax und Plugins wie Fancybox verwendet. Die Suchmaske sieht wie folgt aus.



4.5.2.1 Fancybox

Vorab wird im Folgendem das jQuery-Plugin Fancybox kurz vorgestellt.

Fancybox wird verwendet, um jede Art von Web-Content, abgesondert vom Hintergrund, darzustellen. Es wird als eine Art Lightbox beschrieben, der Hintergrund wird abgedunkelt, sodass der nun wichtige Content im Vordergrund steht. Es gibt sehr viele Anwendungsmöglichkeiten, die wohl bekannteste wird die Darstellung von Fotos sein.

Fancybox ist leicht zu handhaben, sodass der Einbau keine große Hürde darstellt. In einem bereits vorher geschriebenen Kapitel, wird die Verwendung anhand eines Beispiels demonstriert (4.3.1.4 searchOZBPerson).

Um weitere Informationen über Fancybox zu erhalten, wird die Webseite <http://fancybox.net/> empfohlen.

4.5.2.2 Funktionalität

Die Darstellung, bzw. der HTML-Code, befindet sich im Verzeichnis *app/views/application*. Dieses Verzeichnis enthält unter anderem *_suche_konten.html.erb*, in dem die Suchmaske der Kontosuche behandelt wird, und die Datei *_suche_ozb_personen.html.erb*, welche sich um die Personensuche kümmert. Der Aufbau ist in nur wenige Bereiche unterteilt. Die Suchform.

```
<!-- #Suchkriterien: mnr, pnr, rolle, name, vorname, ktoNr -->
<%= form_tag({:action => "searchOZBPerson"}, :id=>"searchOZBPersonForm", :method => 'get') do
%>
  <td><%= text_field_tag "sop_mnr", "", :id => "ktoNr", :style => "width: 80px;" %></td>
  <td><%= text_field_tag "sop_rolle", "", :style => "width: 150px;" %></td>
  <td>
    <table style="border:none">
      <tr>
        <td><%= text_field_tag "sop_name", "", :style => "width: 80px;" %></td>
        <td><%= text_field_tag "sop_vorname", "", :style => "width: 80px;" %></td>
      </tr>
    </table>
  </td>
  <td><%= text_field_tag "sop_ktoNr", "", :style => "width: 80px;" %></td>
  <td><%= link_to_function "Suchen", "", :class => "submit_search", :onclick =>
"$('#searchOZBPersonForm').submit(); return false;", :id => "searchButton" %></td>
<% end %>
```

Und den Bereich, in dem tabellarisch die Ergebnisse dargestellt werden.

```
<% i = 0 %>
<% @personen.each do |person| %>
  <% if !person.OZBPerson.first.nil? %>
    <% if i%2 == 0 then bg_color="white" else bg_color="lightgrey" end %>
    <tr style="background:<%= bg_color %>;">
      <td><%= person.OZBPerson.first.mnr %></td>
      <td><%= person.rolle %></td>
      <td><%= person.name.to_s + ", " + person.vorname.to_s %></td>
      <% if !person.OZBPerson.first.nil? %>
        <% if !person.OZBPerson.first.OZBKonto.first.nil? then %>
          <td><%= person.OZBPerson.first.OZBKonto.first.ktoNr.to_s %></td>
        <% else %>
          <td />
        <% end %>
      <% end %>
    <% end %>
    <% fullname = ""+person.name.to_s+", "+person.vorname.to_s+""" %>
    <td class="insert_td"><%= link_to_function "einfügen", "insert("+person.pnr.to_s+", "+
fullname +")", :class => "insert" %></td>
  </tr>
  <% end %>
  <% i += 1 %>
<% end %>
```


In jeder Ergebniszeile bekommt der User die Möglichkeit, den gewünschten Datensatz in das Eingabeformular einzufügen. Dies geschieht durch den Folgenden Javascript-Aufruf.

```
<%= link_to_function "einfügen", "insert("+person.pnr.to_s+","+ fullname +")", :class =>
"insert" %>
```

Die Methode insert ist wie folgt definiert.

```
function insert(value, value2) {
  $(insertId).val(value);
  $(insertId+"Name").val(value2);
  $.fancybox.close() ;
}
```

Die übergebenen Wertepaare werden mit Hilfe von jQuery in das vorher gewählte DOM-Element eingefügt. Anschließend wird die Fancybox, d.h. die Suchmaske, geschlossen.

Entscheidend ist das Versenden einer Suchanfrage. Es soll vermieden werden, dass sich die Seite komplett neu aufbaut, somit kommt die Ajax-Technologie zum Tragen. Sie wird wie folgt angewandt.

```
// Search form.
$('#searchOZBPersonForm').submit(function () {
  $.get("/application/suche_ozb_personen.js", $(this).serialize(), null, 'script');
  return false;
});
```

Sobald auf den Suchen-Button geklickt wird, wird die Form abgeschickt, d.h. ihre Daten werden versandt. Nun wird noch eine weitere Funktionalität getriggert: Mit Hilfe von "get" werden die in der Form enthaltenen Daten an die angegebene Javascript-Datei weitergeleitet, in dieser Datei passiert nun folgendes.

```
$('#personen').html('<%= escape_javascript(render("application/suche_ozb_personen")) %>');
```

Mit Hilfe von Javascript wird nun eine Ruby-Methode ausgelöst, die das Ergebnis der Suchoperation berechnet und den neuen HTML-Code liefert. Dieser wird nun mit dem in der Fancybox vorhandenen Content getauscht, ohne die Seite vollständig neuzuladen.

Ein weiteres Problem stellt `will_paginate` dar, sobald versucht wird eine andere Ergebnis-Seite aufzurufen, wird dies nicht automatisch mit Ajax geschehen. Darum kümmert sich folgendes jQuery-Statement.

```
$(function () {  
  $('.pagination a').live('click',  
    function () {  
      $.getScript(this.href, {search:"person"});  
      return false;  
    }  
  );  
});
```

Hier wird der Klick auf einen der Links “zurück” oder “weiter” abgefangen und ein weiterer Javascript-Aufruf wird ausgelöst. Dabei handelt es sich um einen weiteren Ajax-Aufruf, welcher die Ruby-Applikation anweist, die nächste Seite zu laden. Nun ruft er den von `will_paginate` erzeugten Link auf, und tauscht dabei lediglich die Daten der neuen Seite mit der, inzwischen alten, aus.

4.5.3 Autocomplete (MB)

Autocomplete ist ein weiteres jQuery Plugin. Es verhilft einem normalen Textfeld zu neuen Funktionalitäten. Autovervollständigungen sollen dem User eine Hilfe sein, längere Texte einzugeben. Oft jedoch fallen dem User nur Bruchstücke eines gesuchten Namens ein, in diesem Fall kann Autocomplete helfen. Wie es verwendet wurde, wird in Kapitel 3.3.2.1 gezeigt. Für weitere Informationen wird die Webseite <http://jqueryui.com/demos/autocomplete/> empfohlen.

4.6 Validierung (ML/MB)

Die Validierung passiert in der Applikationsschicht, genauer im jeweiligen Model. Hier wird die Methode “validate” überschrieben in der anschließend die Validierungslogik implementiert wird.

```
def validate!  
...  
end
```

Falls Fehler auftreten, werden diese in einem roten Kasten aufgezählt.

Bürgschaft Neu anlegen

2 Fehler
Konnte keine Bürgschaft anlegen, weil:

- Die Kontonummer darf nicht leer sein.
- Bitte geben Sie einen Sicherheitsbetrag größer 0 an.

Bürge *:	<input type="text" value="Mustermann, Max"/>	<input type="button" value="Suchen..."/>
Gesellschafter *:	<input type="text" value="Kahn, Oliver"/>	<input type="button" value="Suchen..."/>
Konto-nr. *:	<input type="text"/>	<input type="button" value="Suchen..."/>
Sichabdatum:	<input type="text"/>	
Sichenddatum:	<input type="text"/>	
Sichbetrag:	<input type="text"/>	
Sichkurzbez:	<input type="text"/>	

4.6.1 Pflichtfelder

Die Pflichtfeldüberprüfung wird im jeweiligen Model eingestellt mit `validates_presence_of`¹⁴ und den jeweiligen Feldern. Die Fehlermeldungen, welche im jeweiligen Controller abgefangen werden, werden im Anschluss in einem Array gesammelt.

```
#Person erstellen
@new_Person = Person.create( :rolle => params[:rolle],
  :name => params[:name], :vorname => params[:vorname],
  :geburtsdatum => params[:gebDatum], :strasse => params[:strasse],
  :hausnr => params[:hausnr], :plz => params[:plz], :ort => params[:ort],
  :antragsdatum => params[:antragsdatum],
  :aufnahmedatum => params[:aufnahmedatum] )

#Fehler aufgetreten?
if !@new_Person.errors.empty? then
  @errors.push(@new_Person.errors)
end
```

Später wird das Array innerhalb der View aufgerufen und die Fehlermeldungen angezeigt.

```
<% if !@errors.nil? && @errors.any? %>
  <div id="error_explanation">
    <h2>Konnte keine Person anlegen:</h2>
    <ul>
      <% @errors.each do |error| %>
        <% error.full_messages.each do |msg| %>
          <li><%= msg %></li>
        <% end %>
      <% end %>
    </ul>
  </div>
<% end %>
```

¹⁴ <http://ar.rubyonrails.org/classes/ActiveRecord/Validations/ClassMethods.html#M000083>

4.6.2 Ausblick

Eine Validierung kann ein großes Ausmaß annehmen, deren Umfang für eine weitere Projektarbeit genügen würde. Somit wurde die Validierung in diesem Projekt eingeschränkt, jedoch wurden Ausblicke auf die Möglichkeiten gegeben.

Ein Beispiel ist die Validierung eines o/ZB-Kontos.

Hier wird überprüft, ob es sich bei der eingegeben Nummer auch wirklich um eine Nummer handelt und ob eine Kontonummer eingegeben worden ist. Dies geschieht mit einem regulären Ausdruck, mit dem sich sehr viele Muster darstellen lassen, um z.B. auch Datumsformate zu überprüfen.

```
# Kontonummer
if self.ktoNr.nil? then
  errors.add("", "Bitte geben sie eine Kontonummer an.")
else
  if self.ktoNr.to_s.match(/[0-9]+/).nil? then
    errors.add("", "Die Kontonummer muss eine Zahl sein.")
  end
end
end
```

Ausschnitt aus *models/ozb_konto.erb*

Eine einfache Variante ist zu prüfen, ob zumindest ein Feld aus einer Gruppe korrekt ausgefüllt worden ist.

```
if (self.blz.nil? or self.blz == "") and (self.bic.nil? or self.bic == "") and (self.iban.nil?
or self.iban == "") then
  errors.add("", "Bitte geben Sie mindestens eine BLZ, BIC oder IBAN an.")
end
```

Ausschnitt aus *models/bankverbindung.erb*

Eine etwas anspruchsvollere Variante enthält eine Datenbankabfrage, z.B. um zu prüfen, ob eine angegebene Personalnummer der Datenbank bekannt ist.

```
# Bürgschafter
if self.pnrB.nil? then
  names = bName.split(",")
  self.pnrB = find_by_name(names[0], names[-1])
  person = Person.where("pnr = ?", self.pnrB).first

  if self.pnrB == 0 then
    self.pnrB = nil
    errors.add("", "Personalnummer oder Name des Bürgschafter konnte nicht gefunden
werden.")
  end
else
  person = Person.where("pnr = ?", self.pnrB).first
end
end
```

Ausschnitt aus *models/buergschaft.erb*

4.7 Internationalisierung (I18N) (ML)

Seit Ruby on Rails 2.2 ist eine Internationalisierung in Form einer API integriert. Dieses Thema soll nur kurz angerissen werden, da eine Internationalisierung nicht direkt benötigt wird. Sie stellt eine Erleichterung dar, die bestehenden Meldungen von Gems und Formatierungen für die gewünschte deutsche Lokalisierung zu übersetzen.

Um den Standartwert für die Lokalisierung zu setzen, muss in der *config/environment.rb* folgende Zeile geändert werden.

```
config.i18n.default_locale = :de
```

4.7.1 Lokalisierung (L10N)

Die Lokalisierungsdateien befinden sich im Ordner *config/locales/*. Hier werden die einzelnen Gems übersetzt, Datum- und Zeitformatierungen definiert.

4.8 Security (Rechteverwaltung) (ML)

Die o/ZB benötigt für Ihre unterschiedlichen Sachbearbeiter entsprechende Rechte, damit nicht jeder die Rolle des Administrator einnimmt. Deshalb wurde für jeden administrativen Punkt im Admin-Dashboard ein Rechteflag in der Tabelle OZBPerson angelegt. Somit kann man jeden User für den jeweiligen Punkt freischalten, um Daten einsehen, editieren oder löschen zu können.

Diese lauten wie folgt.

```
:canEditA, :canEditB, :canEditC, :canEditD
# A: OZBPersonenverwaltung
# B: Kontenverwaltung
# C: Veranstaltung und Teilnahmeverwaltung
# D:Tabellen (Veranstaltung, Kontoklassen, Projektgruppen)
```

Ausschnitt aus *models/ozb_person.rb*

Nun wird für jede Aktion, die in Verbindung mit diesen Rechten ist, eine weitere Abfrage nötig:

```
if current_OZBPerson.canEditD then
  ...
else
  redirect_to "/"
end
```

Ausschnitt aus *controllers/kontenklasse_controller.rb*

Besitzt der eingeloggte User nicht das geforderte Rechteflag, wird er auf die Startseite (Dashboard) weitergeleitet.

4.9 Layout (CSS) (MB)

Das Layout wird durch Cascading Style Sheets unterstützt.

Zur Zeit benötigt diese Web-Applikation nur eine CSS-Datei. Diese findet man hier:

public/stylesheets/global.css.

Zu Beginn werden grundlegende Elemente definiert, im Anschluss daran werden eigene Klassen eingeführt. Die Web-Applikation wird in 2 Bereiche gegliedert, Navigation und Content. Die Navigation wird mit Hilfe der Klasse "topbar", der Content mit der Klasse "content" definiert. Zu erwähnen sind auch die Klassen "Submit", von der sich weitere Unterklassen ableiten und die Klasse "searchTable". Erstere kümmert sich um alle relevanten Buttons, wie z.B. "zurück", "löschen", "suchen" etc., sprich alle Aktionen die einen Einfluss auf Datensätze oder die Navigation einnehmen. Letztere behandelt die Darstellung der Suchergebnisse in jeder Suchmaske (s. Kapitel 4.5.2).

Des Weiteren ergibt sich im Zusammenhang mit dem, eigens für den User generierten, Feedback eine weitere ID "notice". Diese behandelt die Darstellung des Feedbacks.

Zuletzt findet die Darstellung der Validierung in dieser CSS-Datei einen Platz. Die Klasse "field_with_errors" beinhaltet wesentliche Elemente der Darstellung, daneben gibt es noch die ID "error_explanation", die sich um die Darstellung der einzelnen Validierungsfehlerpunkte kümmert.

4.10 Konfigurationen (ML)

4.10.1 Routes

Die Routes, zu finden unter *config/routes.rb*, ist ein sehr wichtiger Bestandteil der Web-Applikation. Hier werden die aufgerufenen Adressen mit den Funktionen der einzelnen Controllern verknüpft.

Die Einbindung von Devise, wurde bereits erläutert.

Daneben ergibt sich eine Struktur, die von der Controller-Struktur geprägt ist.

Besonderheiten hier ergaben sich nur durch die Angabe, ob dieser Link durch einen GET oder POST Request aktiv wird. GET Aufrufe werden in der Regel bei darstellenden Views veranlasst.

Sobald es jedoch um Funktionalitäten, wie bei dem Abspeichern oder Aktualisieren von Datensätzen, handelt, wird der Parameter *:via* entsprechend mit POST gesetzt.

```
match '/OZBPerson/:id' => 'OZBPerson#edit', :via => :GET
match '/OZBPerson/:id' => 'OZBPerson#update', :via => :POST
```

Ausschnitt aus *config/routes.rb*

Die letzte Besonderheit ist, dass zwingend der *root* Parameter gesetzt sein muss, damit Devise korrekt funktioniert. Dieser wurde mit der Anzeige des Dashboards verknüpft.

```
# Root
root :to => "index#dashboard"
```

Ausschnitt aus *config/routes.rb*

4.10.2. MySQL

Da MySQL das gewünschte Datenbanksystem ist und die Entwicklung bisher nur auf *sqlite* stattfand, wurde ein kurzer Test mit einer MySQL Datenbank gemacht. Dazu waren folgende Schritte nötig.

Im Gemfile musste statt *sqlite*, *mysql* hinzugefügt werden und in *config/database.yml* die Verbindungsdaten geändert werden.

```
#Datenbank
gem 'mysql'
```

Ausschnitt aus Gemfile

```
development:
  adapter: mysql
  database: ozb_project
  encoding: utf8
  username: ozb
  password: ozbpaswd
  host: localhost
```

Ausschnitt aus *config/database.yml*

Außerdem musste die alte Enumeration-Definition in den Models geändert werden. Ein Beispiel wäre.

```
#Alt
t.column "status", :enum, :limit => [:n, :d, :a]

#Neu
t.column :status, "ENUM('n', 'd', 'a')"
```

Ausschnitt aus db/migrate/20111014085624_create_tanliste.rb

5. Fazit (ML/MB)

Im Folgenden werden drei Themen behandelt:

- Aufgetretene Probleme und Konsequenzen
- Anforderungsbeschreibung an weiterführenden Studenten
- Ausblick auf die Fortführung dieser Projektarbeit

Bei der Bearbeitung dieser Projektarbeit sind keine fachlichen Probleme aufgetaucht, die mit der Umsetzung dieser Web-Anwendung im Zusammenhang stehen. Die hohe Anzahl der Geschäftsprozesse hat den kritischen Punkt überschritten, sodass die Anforderung "Realisierung der Geschäftsprozesse" angepasst werden musste. Auch im Hinblick auf die weitere Anforderung "Realisierung einer Web-Anwendung", die von Grund auf neu geplant und umgesetzt werden sollte, ist diese Entscheidung nötig gewesen.

Die Ziele dieser Projektarbeit wurden neu aufgefasst. Ein wichtiger Punkt war die Prüfung der Realisierung aller Anforderungen mit Hilfe von "Ruby on Rails" (RoR) in Verbindung mit MySQL, welche durch einen Proof of Concept erfolgreich gezeigt und beendet werden konnte. Daraufhin wurde die Anforderung "Realisierung der Geschäftsprozesse" auf eine überschaubare Menge reduziert, sodass mehr Zeit für den Grundstein dieser Web-Anwendung gelegt werden konnte. Dazu zählt ein neues Erscheinungsbild der Web-Anwendung, Umsetzung des neuen Datenbankschemas, Einbau eines Rechtemanagements, erste Geschäftsprozess-Realisierungen, Einbindung eines User-Session-Management und eines Leitfadens für die Validierung. Darüber hinaus wurde die Anwendung von jQuery Plugins demonstriert, sowie Realisierungen von Eingabehilfen, wie z.B. den verschiedenen Suchmasken, die mit Hilfe einer Lightbox und Ajax-Technologie einfach und komfortabel zu bedienen sind.

Daraus lässt sich ableiten, dass die Projektarbeit einem zukünftigen Studenten als Basis dienen soll, sodass er sich im nächsten Schritt auf die Realisierung aller weiteren Geschäftsprozesse konzentrieren kann.

Die letzte wichtige Anforderung an die Projektarbeit ist, dem zuvor angesprochenen Studenten, einen detaillierten Einblick in die Web-Anwendung zu liefern. Der Student soll die komplette Architektur und Funktionsweise, mit Hilfe dieses Dokumentes, nachvollziehen können. Dadurch wird sich eine Reduzierung der Einarbeitungszeit erhofft.

Damit diese Hoffnung sich erfüllen kann, muss dieser Student einige Anforderungen erfüllen. Der Student sollte unbedingt weitreichende Kenntnisse in RoR und Web-Anwendungen mit sich bringen. Darüber hinaus sind Java-Script und Ajax nicht zu vergessen, denn die Web-Anwendung implementiert u.a. diese Technologien. Ausreichende Kenntnisse sind für die Datenbankumsetzung nötig.

Der Ausblick beinhaltet die Fertigstellung aller Geschäftsprozesse, darüber hinaus sollte man sich Gedanken über Validierung und eine Konvertierung der Daten des alten Datenmodells in

dem nun umgesetzten Datenmodell machen. Zur Umsetzung der Konvertierung der Daten bieten sich automatisierte generierte Seeds an. Ein weiteres Thema stellt das Deployment dar. Es muss sichergestellt sein, dass auf dem Server eine Ruby-Server-Anwendung laufen kann. Darüber hinaus sollte die Erstellung eines Deployment-Skripts in Betracht gezogen werden.

Zum Abschluss einige persönliche Eindrücke. Die Bearbeitung dieser Projektarbeit war durchaus anspruchsvoll und zeitintensiv. Jedoch ist die Motivation eine solide Web-Anwendung anzubieten jederzeit da gewesen. Die Konzeption einer Web-Anwendung und die anschließende Umsetzung hat uns Freude bereitet. Zudem ist der Kontakt mit den Verantwortlichen sehr angenehm gewesen und stellt eine sehr gute Vorbereitung auf die spätere Realität dar.

Wir möchten uns auch bei der tatkräftigen Unterstützung durch Herrn Kienle, Herrn Kleinert und Herrn Schäfer bedanken.