



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Fakultät für Informatik und
Wirtschaftsinformatik

Studiengang Informatik Bachelor

Moltkestraße 30
76133 Karlsruhe

Dokumentation zur

Projektarbeit **o/ZB**

von

Patrick Hillert und Aleksej Lednev

Realisierung einer Web-Applikation mit Ruby on Rails

im

Sommersemester 2012

Betreuer Hochschule: Prof. Dr. Frank Schaefer
Auftraggeber: Tassilo Kienle, o/ZB Stuttgart

Inhaltsverzeichnis

1	Einleitung	5
2	Arbeitsumgebung und verwendete Hilfsmittel	6
2.1	Backend.....	6
2.2	Middleware	6
2.3	Frontend	6
2.4	Dokumente	6
3	Einführung in Ruby on Rails (PH)	7
3.1	Namenskonventionen und Routen	7
3.2	ActiveRecord Validations	9
3.3	ActiveRecord Callbacks	11
3.3.1	<i>Callbacks</i> die zur Verfügung stehen:	11
3.3.2	Methoden, die Callbacks triggern	13
3.3.3	Methoden, die Callbacks überspringen	13
3.4	ActiveRecord Associations.....	16
3.4.1	Associations die zur Verfügung stehen	16
3.4.2	Methoden, die durch Associations zur Verfügung stehen	19
3.5	Leserliche Spaltennamen („Humanized Attributes“)	22
3.6	Leserliche Fehlermeldungen.....	22
3.7	Leserliche Formular-Labels.....	25
4	Beschreibung der Geschäftsvorfälle (PH/AL)	26
4.1	Kontenverwaltung (OZB-Konten: EE-/ZE-Konten) (PH)	27
4.1.1	Meine Konten (EE-/ZE-Konten) (PH).....	34
4.1.2	Bürgschaften	35
4.1.3	Routen	37
4.1.4	OzbKontoController.....	37
4.1.5	OzbKonto Model	40
4.1.6	EeKonto Model.....	44

4.1.7	Bankverbindung Model.....	46
4.1.8	Bank Model	47
4.1.9	ZeKonto Model.....	48
4.1.10	KKL-Verlauf Controller	50
4.1.11	KklVerlauf Model	50
4.2	Verwaltung von Bürgschaften (PH).....	52
4.2.1	Routen	52
4.2.2	Bürgschaften Controller.....	52
4.3	Mitgliederverwaltung (AL)	55
4.3.1	Sonderberechtigungen	55
4.3.2	Eigene Daten verwalten	55
4.3.3	Daten anderer Benutzer verwalten	65
4.3.4	Routen für die Mitgliederverwaltung	66
4.3.5	Verwaltung Controller	66
5	Beschreibung des CSV-Import für Buchungen (PH)	80
5.1	Routen.....	80
5.2	Struktur der CSV-Datei	80
5.3	Web-Import.....	81
5.4	Klasse CSVImporter	83
6	Layout / Twitter Bootstrap (AL)	87
6.1	Bootstrap (Framework) ⁵	87
6.2	Basisdaten.....	87
6.3	Grundgerüst: Grid-System und Responsive Design ⁵	87
6.4	Grundlegendes CSS-Stylesheet ⁵	88
6.5	Wiederverwendbare Komponenten ⁵	88
6.6	JavaScript-Plugins ⁵	88
6.7	Ruby on Rails und Twitter Bootstrap.....	89
6.7.1	Einbindung der Twitter Bootstrap Bibliotheken (v2.1.0)	89
6.7.2	Hauptnavigationsleiste	89
6.7.3	Sub Navigationsleiste (Tabs) (AL/PH)	92

6.7.4	will_paginate	93
6.7.5	jQuery Tablesorter	95
6.7.6	Modaler Dialog und jQuery Autovervollständigung	97
6.7.7	jQuery Datepicker	102
7	Fazit (PH/AL).....	103
8	Abbildungsverzeichnis.....	104

1 Einleitung

Die Ohne-Zins-Bewegung in Stuttgart (kurz „o/ZB Stuttgart“) ist ein regionales Finanzierungsinstrument, die Ihren Mitgliedern die Möglichkeit gibt Geld ohne Zinsen zu leihen. Geld, das durch andere Mitglieder ohne Zins angespart wird. Damit dieses Leihen und Sparen im Gleichgewicht bleibt, wird ein Punktesystem verwendet. Dieses erlaubt es, nur eine bestimmte Summe zu leihen und sorgt für einen Ausgleich zwischen geliehenem und gespartem Geld.

Für weitere Informationen sei an dieser Stelle auf die Homepage der o/ZB verwiesen. Diese findet sich unter der Internet Adresse <http://www.ozb.eu/>

Die Projektarbeit wurde insgesamt von drei Studenten bearbeitet. Aleksej Lednev, Robert Mrasek und Patrick Hillert. Die Ausarbeitung von Robert Mrasek findet sich in einer separaten Dokumentation, da er sich um alle Angelegenheiten rund um das Datenmodell, der Datenbank und des Datenschemas kümmerte.

Die Dokumentation zum Datenmodell umfasst die folgenden Punkte:

- Erstellung eines konsistenten Datenmodells
- Historisierung des Datenmodells
- Java-Importprogramm zwischen altem und neuem Datenmodell
- Rechteverwaltung

Dieser Teil der Dokumentation zur Anwendungslogik umfasst die folgenden Punkte:

- Umsetzung der Geschäftsprozesse:
 - Kontenverwaltung CRUD (OZB-Konto: EE-Konto, ZE-Konto) (PH¹)
 - Verwaltung von Bürgschaften CRUD (PH)
 - Mitgliederverwaltung CRUD (Fördermitglied, Gesellschafter, ...) (AL²)
- CSV Web-Import von Buchungen aus einer FiBu Software (PH)
- Umsetzung eines einheitlichen Designs mittels Twitter's Bootstrap-Framework (AL)

Das Gesamtziel der Realisierung dieser Web-Anwendung besteht darin, ein System mit konsistentem Datenmodell, lauffähig auf einem produktiven Ruby on Rails Webserver einzurichten.

¹ PH: Text von Patrick Hillert

² AL: Text von Aleksej Lednev

2 Arbeitsumgebung und verwendete Hilfsmittel

2.1 Backend

Für die Entwicklung zwischen Middleware und Backend sowie dem Debuggen und Testen der Daten aus und in die Datenbank wird ein MySQL-Server (Version 5) verwendet. Die darin enthaltene Datenbank der o/ZB verwendet als Engine InnoDB.

Der Zugriff über phpMyAdmin auf die Datenbank gestaltete sich sehr komfortabel über einen zusätzlich installierten Apache Webserver direkt auf dem Entwicklungsrechner. phpMyAdmin erlaubt das detaillierte Analysieren der zugrundeliegenden Daten und bietet sehr viele Möglichkeiten der Manipulationen an.

2.2 Middleware

Als Middleware steht das Ruby on Rails Framework mit Webserver zur Verfügung. Dieser verbindet Backend und Frontend miteinander. Rails bringt gleich mehrere eigene Webserver mit. Wahlweise steht hier WEBRick oder Mongrel zu Verfügung. Der hier verwendete Server ist WEBRick.

Als Entwicklungsumgebung wurde Aptana Studio 3 verwendet, hauptsächlich wegen der einfachen Navigation innerhalb des Projektbaumes und des Syntaxhighlightings.

2.3 Frontend

Im Frontend kamen JavaScript, CSS und HTML als klassische Webvertreter zum Einsatz. Zur einfachen Verwaltung des Designs wurde komplett auf Twitter's quelloffenes CSS-Framework Bootstrap zurückgegriffen. Für JavaScript wurde auf jQuery als ausgezeichneten und einfach zu handhabenden Framework gesetzt.

Zur Analyse und Debugging des Frontend wurden im Browser Mozilla Firefox die Erweiterungen Web Developer und Firebug verwendet.

2.4 Dokumente

Als wichtige Informationsquellen dienten allem voran die Ausarbeitungen der vorherigen Studenten Sergius Dyck, Dmytro Okunevych, Philipp Henschel, Martin Briewig und Michael Leibel. Sie alle lieferten wichtige Grundlagen die unverzichtbar waren für ein fortwährendes und solides Fundament an Wissen, das auch in diese Projektarbeit mit einfluss. Für diesen Teil der Dokumentation ist die Dokumentation von M. Briewig und M. Leibel zentraler Kern.

3 Einführung in Ruby on Rails (PH)

Die nachfolgenden Abschnitte sind für die generelle Einführung in Ruby on Rails und als kleines Nachschlagewerk niedergeschrieben. Hier werden die wichtigsten Punkte aufgegriffen und kurz erklärt. Anschließend folgen die Erklärungen für die OZB-Anwendung.

Ruby on Rails ist nicht irgendein gewöhnlicher Framework, Rails folgt dem DRY-Prinzip („Don't repeat yourself“) und gibt mit seinem durchdachten Design viele Vorlagen wie Aufgaben zu lösen sind. Allerdings erfordern diese Besonderheiten eine größere Einarbeitungszeit, die sich auf lange Sicht aber mehr als rechnet. Nicht nur die Wartung des erzeugten Codes wird einfacher, auch die Entwicklung und spätere Änderung von einzelnen Komponenten wird dadurch extrem vereinfacht.

Nachfolgend finden sich ergänzend zu den Erklärungen der beiden Studenten M. Briewig und M. Leibel einige besonders wichtige Kernelemente, die bei der Erstellung einer RoR-Anwendung nicht ausgelassen werden sollten.

3.1 Namenskonventionen und Routen

Rails versucht aus einer kleinen Menge an Informationen das Maximum an Wissen zu gewinnen. So werden beispielsweise Namen von Controllern im Plural angegeben. Die zugehörigen Models dagegen im Singular benannt. Daraus lässt sich zu jeder Zeit ableiten wie auf das zugehörige Model zugegriffen werden kann, da der Name bereits bekannt ist. Die Namen von Klassen - egal ob Controller oder Model - werden immer in CamelCase angegeben. CamelCase bedeutet, dass keine Unterstriche in Klassennamen und für jedes Wort zu Beginn ein Großbuchstabe verwendet wird.

So ist beispielsweise der Controller *ClientsController* korrekte Rails Schreibweise, *ClientController* (hier fehlt ein „s“ am Ende), *CLIENTController* (hier folgenden dem ersten Großbuchstaben weitere) oder gar *Client_Controller* (Unterstrich) hingegen nicht. Das ist immer dann wichtig, wenn man die in Rails mitgelieferten Helper-Klassen verwendet - man verwendet sie meistens ohne es zu bemerken. Diese helfen dem Entwickler schneller und effizienter zu entwickeln. Dazu zählen vor allem die URL-Helper, die Pfade verwalten. Damit ist es dann möglich zu einem späteren Zeitpunkt in einem Projekt Pfade zu ändern, ohne dass irgendwelche manuellen Anpassungen in Views gemacht werden müssen. So wird durch die Angabe von

```
<%= link_to "Client XY", client_path(@client) %>
```

innerhalb einer View ein HTML-Link erzeugt, der den Namen *Client XY* trägt und beispielsweise auf */client/23* verknüpft. Vor jeden Großbuchstaben - beginnend ab dem zweiten im Namen - wird ein Unterstrich eingefügt und alle Großbuchstaben durch Kleinbuchstaben ersetzt. So würde in diesem Beispiel der Controller *ClientsController* unter dem Dateinamen *app/controllers/clients_controller.rb* abgespeichert sein.

Bei Model-Klassen sieht die Namensgebung etwas anders aus. Da Models generell als Repräsentant für die zugrundeliegende Tabelle oder Tabellen stehen, in ihrer Instanz aber immer für genau einen Datensatz, so werden sie im Singular angegeben und unter *app/models/* gespeichert. Im Zusammenhang mit der Controller-Klasse *ClientsController* wäre hier die Klasse *Client* unter dem Dateinamen *app/models/client.rb* gespeichert.

Und genau so arbeiten auch die Routen. Routen sind in Rails nichts anderes als Angaben darüber, wie URLs zu Controllern und deren Action gemappt werden. Durch die Angabe einer Resource innerhalb der Datei *config/routes.rb* werden vollständige CRUD-Pfade erzeugt. Im Beispiel *Clients* stehen mit

```
:resources :clients
```

in den Views die folgenden HTTP-Verben (RESTful) zur Verfügung:

HTTP-Verb	Pfad	Action	
GET	/clients	index	eine Liste aller Kunden
GET	/clients/new	new	ein Formular zum Kunden anlegen
POST	/clients	create	erzeugt einen neuen Kunden
GET	/clients/:id	show	zeigt einen bestimmten Kunden an
GET	/clients/:id/edit	edit	ein Formular zum Kunden editieren
PUT	/clients/:id	update	aktualisiert einen bestimmten Kunden
DELETE	/clients/:id	destroy	löscht einen bestimmten Kunden

Hält man sich an diese Vorgaben, so macht die Entwicklung unter Rails weniger Probleme bei der Verwendung von vielen weiteren Features.

3.2 ActiveRecord Validations

Validierung von Daten ist der zentrale Punkt bei einer Anwendung, um fehlerhafte Daten erst gar nicht in irgendeiner Form in die Anwendung aufzunehmen, und so von Anfang an Fehler zur Laufzeit zu vermeiden. Dabei bietet Rails auch an dieser Stelle sehr viele Möglichkeiten der (sauberen) Validierung von Daten. Und zwar genau an der Stelle, an der Daten verwaltet werden - im Model.

Geht man von einem Controller und einem zugehörigen Model aus, so lässt sich eine Validierung generell immer auch im Controller verwalten. Das mag auch passend sein und in der Funktionsweise keinen Unterschied zur Validierung innerhalb eines Models darstellen. Wird allerdings im Verlauf des Projektes oder aufgrund von Erweiterungen an anderen Stellen nach dem Erzeugen von Datensätzen gefragt, so wird spätestens dann eine ausgelagerte Funktion benötigt, um die gleiche Validierung zu implementieren oder - als sehr unsauberen Stil - sogar die Implementierung 1:1 an die neue Stelle im Quellcode kopiert. Spätestens jetzt sollte man sich überlegen, ob die Validierung nicht auch „geschickter“ platziert werden könnte. Und genau hier fasst Rails das Konzept der Validations auf und bringt von Haus aus die wichtigsten Validierungen mit ist aber offen für komplexere Überprüfungen.

ActiveRecord bietet einige sehr wichtige ValidationHelper an:

- *acceptance*
 - ❖ validiert ein Attribut, das im Formular eine Checkbox darstellt
- *validates_associated*
 - ❖ überprüft ein mit diesem Model verknüpftes Model
- *confirmation*
 - ❖ überprüft ein Attribut mit dem zugehörigen Attribut <attribut>_confirmation
- *exclusion*
 - ❖ überprüft ein Attribut, ob es nicht in einer Menge von Werten vorkommt
- *format*
 - ❖ überprüft ein Attribut auf ein bestimmtes Format mittels RegEx
- *inclusion*
 - ❖ überprüft ein Attribut auf ein Vorkommen innerhalb einer Menge von Werten
- *length*
 - ❖ überprüft ein Attribut auf eine bestimmte Länge
- *numericality*
 - ❖ überprüft ein Attribut auf einen gültigen Zahlenwert
- *presence*
 - ❖ überprüft ein Attribut auf Existenz

- *uniqueness*
 - ❖ überprüft ein Attribut auf Eindeutigkeit innerhalb der Datenbankspalte
- *validates_with*
 - ❖ gibt eine separate Klasse zur Validierung des Models an
- *validates_each*
 - ❖ überprüft eine Menge von Attributen auf individuelle Weise

Diese *Validations* können direkt innerhalb der Model-Klasse verwendet werden.

An dieser Stelle sei für weitere Informationen auf die Ruby on Rails Guide-Seite http://guides.rubyonrails.org/active_record_validations_callbacks.html verwiesen. Diese bietet sehr viele praktische Beispiele, die für jeden Rails-Entwickler hilfreich als Nachschlagewerk dienen.

Die konkrete Validierung innerhalb der o/ZB-Anwendung findet (bezogen auf die OZB-Konten), innerhalb aller abhängigen Klassen statt: OzbKonto, KklVerlauf, EeKonto, Bankverbindung, Bank und ZeKonto.

Zum Beispiel muss eine gültige Kontonummer nur aus Zahlen bestehen, mindestens jedoch aus einer. Eine solche Validierung wird innerhalb der Klasse OzbKonto verwendet und wird wie folgt definiert

```
validates :KtoNr, :presence => { :format => { :with => /[0-9]+/ }, :message => "Bitte geben Sie eine gültige Kontonummer an." }
```

Wird beim Anlegen oder Ändern eines Datensatzes keine Ziffer für die Kontonummer angegeben, so wird die angegebene Fehlermeldung zusammen mit dem Symbol *:KtoNr* (und dessen Übersetzung) als Fehlermeldung zurückgeliefert und dem Benutzer angezeigt. Der Datensatz wird nicht gespeichert.

Aber nicht nur einfache Überprüfungen sind möglich, auch komplexere Abfragen in eigenen Funktionen können mit Ruby on Rails definiert und durchgeführt werden. Die meisten Validierungen benötigen aber keine gesonderte Behandlung und können wie im Beispiel oben verwendet werden.

3.3 ActiveRecord Callbacks

Mindestens genauso hilfreich aber etwas weniger wichtig wie die Validierung sind die sogenannten *ActiveRecord Callbacks*. Diese helfen dem Entwickler bei unterschiedlichen Aufgaben innerhalb des Models. Ein Callback ist nichts anderes als eine Art Trigger, der vor, während oder nach einer bestimmten Aktion (erstellen, ändern, löschen) in einer bestimmten Reihenfolge aufgerufen wird. Dadurch ist es zum Beispiel möglich ein zugehöriges Model noch nach dem Speichern zu Manipulieren ohne über eine Klasse erneut auf dieses Objekt zugreifen zu müssen. *Callbacks* können generell auch nur unter bestimmten Umständen ausgeführt werden, diese nennt man dann *Conditional Callbacks*.

3.3.1 Callbacks die zur Verfügung stehen:

Die Nachfolgende Auflistung ist in der Reihenfolge niedergeschrieben, wie sie auch zur Laufzeit auftritt.

3.3.1.1 Erstellen eines Objektes

1. *before_validation*
 - ❖ Aufruf vor der eigentlichen Validierung, um evtl. Attributwerte zu korrigieren oder hinzuzufügen bzw. abhängige Models zu erzeugen oder ähnliches.
2. *after_validation*
 - ❖ Aufruf nach der Validierung, um evtl. Abhängigkeiten aufzulösen oder bestimmte Aktionen erst nach der Validierung durchzuführen.
3. *before_save*
 - ❖ Aufruf vor dem Speichern des Datensatzes, nachdem die Methode *save* auf diesem Objekt aufgerufen wurde.
4. *around_save*
 - ❖ Aufruf vor dem Speichern, kann mittels *yield* den Speichervorgang dann innerhalb dieses *Callbacks* ausführen und anschließend weitere Aktionen ausführen.
5. *before_create*
 - ❖ Vor dem (erstmaligen!) Erstellen des Datensatzes in der Datenbank.
6. *around_create*
 - ❖ Analog zu (erstmaligen!) *around_save* beim Speichern in die Datenbank.
7. *after_create*
 - ❖ Nach der (erstmaligen!) Eintragung in die Datenbank.
8. *after_save*

- ❖ Nachdem der Datensatz gespeichert wurde. Wird sowohl beim Ändern, wie auch beim erstmaligen Speichern aufgerufen.

3.3.1.2 Ändern eines Objektes

1. `before_validation`
 - ❖ Analog zu 3.3.1.1
2. `after_validation`
 - ❖ Analog zu 3.3.1.1
3. `before_save`
 - ❖ Analog zu 3.3.1.1
4. `around_save`
 - ❖ Analog zu 3.3.1.1
5. `before_update`
 - ❖ Vor der Änderung der Daten.
6. `around_update`
 - ❖ Aufruf vor dem Ändern, kann mittels *yield* den Speichervorgang dann innerhalb dieses *Callbacks* ausführen und anschließend weitere Aktionen ausführen.
7. `after_update`
 - ❖ Aufruf nachdem der Datensatz geändert wurde.
8. `after_save`
 - ❖ Analog zu 3.3.1.1

3.3.1.3 Löschen eines Objektes

1. `before_destroy`
 - ❖ Aufruf vor dem Löschen des Objekts mittels der *delete* Methode.
2. `around_destroy`
 - ❖ Aufruf vor dem Löschen, kann mittels *yield* den Löschvorgang dann innerhalb dieses *Callbacks* ausführen und anschließend weitere Aktionen ausführen.
3. `after_destroy`
 - ❖ Aufruf nachdem der Datensatz gelöscht wurde.

3.3.2 Methoden, die Callbacks triggern

- create
- create! (wirft Exception)
- decrement!
- destroy
- destroy_all
- increment!
- save
- save! (wirft Exception)
- save(:validate => false)
- toggle!
- update
- update_attribute
- update_attributes
- update_attributes! (wirft Exception)
- valid?

3.3.3 Methoden, die Callbacks überspringen

- decrement
- decrement_counter
- delete
- delete_all
- find_by_sql
- increment
- increment_counter
- toggle
- touch
- update_column
- update_all
- update_counters

Als Beispiel aus der o/ZB-Anwendung sind in der Model Klasse *OzbKonto* mehrere *Callbacks* notwendig.

```
before_save :set_assoc_attributes, :set_wsaldo_psaldo_to_zero, :set_saldo_datum
```

before_save besitzt gleich drei Methoden, die vor dem Speichern des Datensatzes ausgeführt werden.

```
# bound to callback
def set_assoc_attributes
  if (!self.ee_konto.nil?)
    self.ee_konto.SachPNR = self.SachPNR
  end
end
```

Zum einen die *set_assoc_attributes*, die den gesetzten Sachbearbeiter an seine abhängigen Model Klassen weitergibt. Dadurch muss der Controller, der die Daten für das *OzbKonto* übergibt, nicht wissen, welche Abhängigkeiten sich hinter einem *OzbKonto* befinden. Abhängigkeiten werden damit entkoppelt.

```
def set_wsaldo_psaldo_to_zero
  if (self.PSaldo.nil?)
    self.PSaldo = 0
  end

  if (self.WSaldo.nil?)
    self.WSaldo = 0.0
  end
end
```

set_wsaldo_psaldo_to_zero setzt die Salden des Kontos auf die Zahl Null.

```
# bound to callback
def set_saldo_datum
  if (self.SaldoDatum.nil?)
    self.SaldoDatum = self.KtoEinrDatum
  end
end
```

set_saldo_datum setzt das Datum (falls nicht bereits geschehen) auf das Einrichtungsdatum des Kontos.

```
before_create :set_valid_time

# bound to callback
def set_valid_time
  unless(self.GueltigBis || self.GueltigVon)
    self.GueltigVon = Time.now
    self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
  end
end
```

Der Callback *before_create* setzt in der Methode *set_valid_time* den Gültigkeitszeitraum dieses Datensatzes, wenn der Datensatz zum ersten Mal angelegt wird.

```
after_destroy :destroy_historic_records
```

Und der Callback *after_destroy* löscht mit der Methode *destroy_historic_records* alle abhängigen Datensätze, die vom Zeitpunkt des aufrufenden Datensatzes in die Vergangenheit reichen. Dadurch wird sichergestellt, dass beim Löschen eines Datensatzes auch dessen historisierte Daten mit gelöscht werden und keine verwaisten Datensätze mehr existieren.

```
# bound to callback
def destroy_historic_records
  # find all historic records that belongs to this record and destroy(!) them
  # note: destroy should always destroy all the corresponding association objects
  # if the association option :dependent => :destroy is set correctly
  recs = self.class.find(:all, :conditions => ["KtoNr = ? AND GueltigBis < ?",
    self.KtoNr, self.GueltigBis])

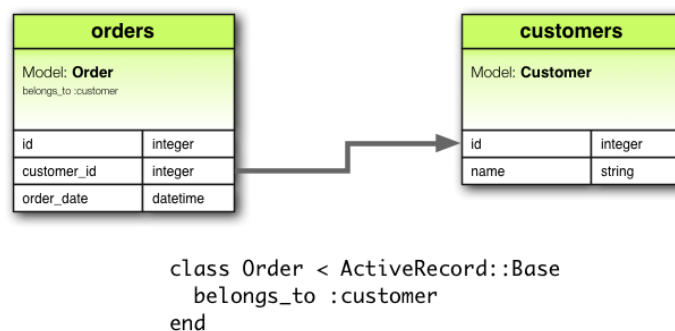
  recs.each do |r|
    r.destroy
  end
end
```

3.4 ActiveRecord Associations

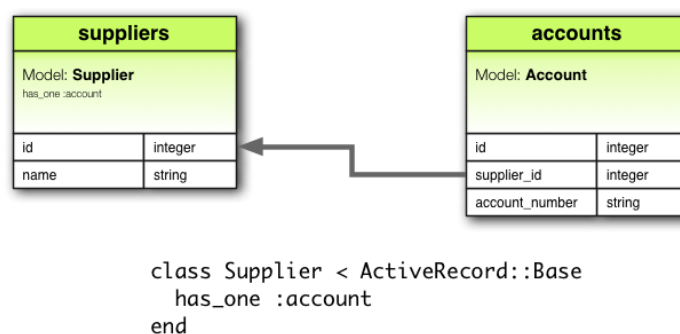
Für Abhängigkeiten in einem komplexen Datenmodell sind *Associations* die wichtigste Grundlage, um diese Abhängigkeiten stabil innerhalb der Anwendung abbilden zu können. *Rails* bietet hier sehr viele nützliche Möglichkeiten, die ein stabiles, schnelles und sehr übersichtliches Arbeiten mit komplexen Daten ermöglichen.

3.4.1 Associations die zur Verfügung stehen³

- *belongs_to*
 - ❖ Erzeugt eine 1:1 Verbindung mit einem anderen Model und zwar so, dass jede Instanz des Models, das diese *Association* verwendet zu einer Instanz des anderen Models gehört. Diese Klasse enthält den Fremdschlüssel zur anderen Klasse.



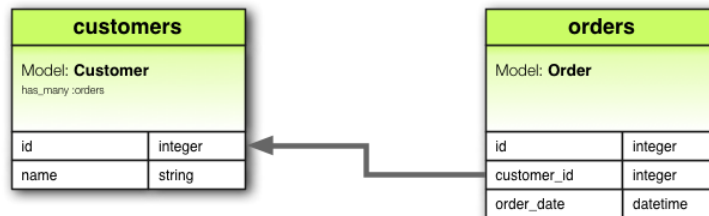
- *has_one*
 - ❖ Erzeugt eine 1:1 Verbindung mit einem anderen Model und zwar so, dass jede Instanz des Models, das diese *Association* verwendet eine Instanz des anderen Models besitzt. Dies ist eine andere Form von *belongs_to*: diese Klasse beinhaltet im Gegensatz zu *belongs_to* keinen Fremdschlüssel sondern die Abhängige Klasse besitzt diesen.



³ Bilddaten verwendet von http://guides.rubyonrails.org/association_basics.html

- *has_many*

- ❖ Erzeugt eine 1:n Verbindung mit einem anderen Model und zwar so, dass jede Instanz des Models, das diese *Association* verwendet mehrere abhängige Instanzen in der abhängigen Klasse besitzt. Die andere Klasse besitzt dann meistens eine *belongs_to Association*.

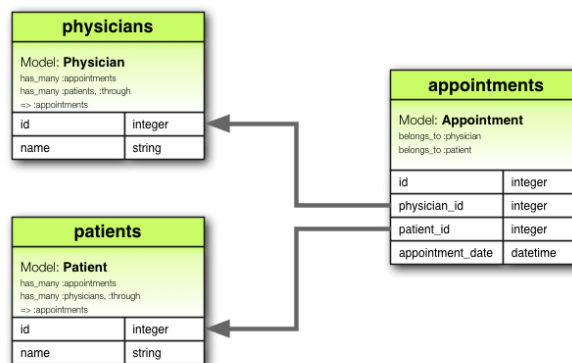


```

class Customer < ActiveRecord::Base
  has_many :orders
end
  
```

- *has_many :through*

- ❖ Erzeugt eine n:n Verbindung mit einem anderen Model und zwar so, dass jede Instanz des Models, das diese *Association* verwendet mehrere abhängige Instanzen in der abhängigen Klasse zugeordnet ist, aber zusätzlich auch jede abhängige Instanzen mehrere dieser Instanzen zugeordnet sind. Die Zuweisung erfolgt über eine dritte Klasse, die die Fremdschlüssel beider Klassen und evtl. weitere Spalten mit Informationen beinhaltet.



```

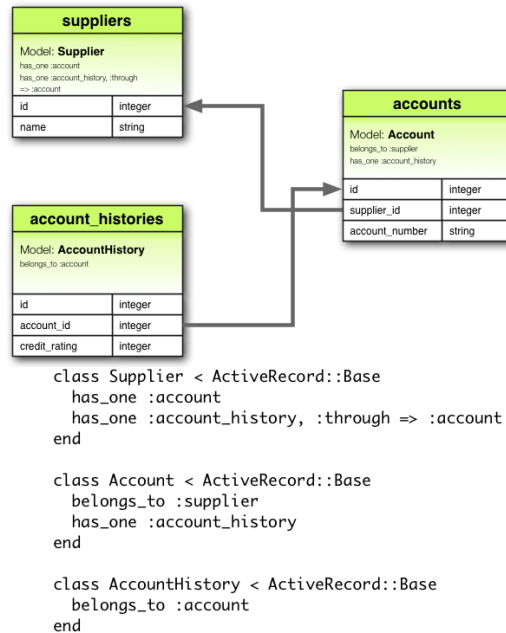
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
  
```

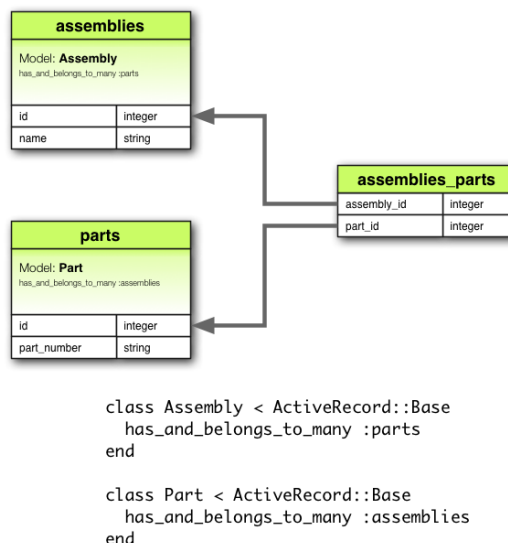
- *has_one :through*

- ❖ Erzeugt eine 1:1 Verbindung mit einem anderen Model und zwar so, dass jede Instanz des Models, das diese *Association* verwendet eine Instanz des anderen Models besitzt aber zusätzlich durch eine dritte Klasse verbunden ist.



- *has_and_belongs_to_many*

- ❖ Erzeugt eine direkte n:n Verbindung mit einem anderen Model und zwar so, dass jede Instanz das diese *Association* verwendet eine Instanz des anderen Models zugeordnet ist und die abhängige Klasse direkt abhängig von dieser Klasse ist. Analog zu *has_many :through* mit dem Unterschied, dass die dritte Tabelle keine weiteren Spalten für Informationen enthält.



3.4.2 Methoden, die durch Associations zur Verfügung stehen

Für jede *Association* stehen unterschiedliche und spezifische Methoden zur Verfügung, die die Beziehung zu den anderen Models vereinfacht. Nachfolgend finden sich eine Liste der *Associations* und deren Methoden mit Erläuterungen.

- *belongs_to*

Für diese Methoden wird die fiktive Model.Klasse Order verwendet.

```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

```
@order = Order.find(23)
```

- ❖ *association(force_reload = false)*
 - `@order.customer` liefert den Kunden zur Bestellung mit id = 23
- ❖ *association=(associate)*
 - `@order.customer = Customer.find(42)` weist der Bestellung ein bestehendes Kundenobjekt zu.
- ❖ *build_association(attributes = {})*
 - `@order.build_customer(:name => „Karl Koch“)` weist der Bestellung ein neues Kundenobjekt zu, die neue id wird automatisch gesetzt, das Objekt aber noch nicht gespeichert.
- ❖ *create_association(attributes = {})*
 - `@order.create_customer(:name => „Karl Koch“)` weist der Bestellung ein neues Kundenobjekt zu (das bereits gespeichert wurde), die neue id wird automatisch gesetzt.

- *has_one*

- ❖ *association(force_reload = false)*
- ❖ *association=(associate)*
- ❖ *build_association(attributes = {})*
- ❖ *create_association(attributes = {})*

- *has_many*
 - ❖ *collection*(force_reload = false)
 - ❖ *collection*<<(object, ...)
 - ❖ *collection.delete*(object, ...)
 - ❖ *collection=objects*
 - ❖ *collection_singular_ids*
 - ❖ *collection_singular_ids=ids*
 - ❖ *collection.clear*
 - ❖ *collection.empty?*
 - ❖ *collection.size*
 - ❖ *collection.find*(...)
 - ❖ *collection.where*(...)
 - ❖ *collection.exists?*(...)
 - ❖ *collection.build*(attributes = {}, ...)
 - ❖ *collection.create*(attributes = {})

- *has_many :through* und *has_one :through*
 - ❖ Analog zu *has_many* und *has_one*

- *has_and_belongs_to_many*
 - ❖ *collection*(force_reload = false)
 - ❖ *collection*<<(object, ...)
 - ❖ *collection.delete*(object, ...)
 - ❖ *collection=objects*
 - ❖ *collection_singular_ids*
 - ❖ *collection_singular_ids=ids*
 - ❖ *collection.clear*
 - ❖ *collection.empty?*
 - ❖ *collection.size*
 - ❖ *collection.find*(...)
 - ❖ *collection.where*(...)
 - ❖ *collection.exists?*(...)
 - ❖ *collection.build*(attributes = {})
 - ❖ *collection.create*(attributes = {})

Für weitere Informationen zu den sehr ausführlichen Guides sei auf die *Ruby on Rails Guides* Homepage verwiesen.⁴

Als Beispiel für *ActiveRecord Associations* zeigt die Klasse *OzbKonto* am Besten, wie Verbindungen zwischen Models funktionieren.

Das *OzbKonto* gehört zu einer *OzbPerson* und ist über die Spalte *Mnr* mit diesem Model verknüpft. Hierzu wird folgendes definiert.

```
# associations
belongs_to :ozb_person, :foreign_key => :Mnr
```

Ein *OzbKonto* hat aber auch immer einen *KKL-Verlauf* für jedes Konto. Ein *KKL-Verlauf* besteht allerdings aus mehreren Datensätzen, bei dem aber zu einem bestimmten Zeitpunkt nur einziger Datensatz von Interesse ist. Deshalb weicht das in *Ruby on Rails* implementierte Model, von dem tatsächlich zu Grunde liegenden Datenschema ab.

```
has_one :kkl_verlauf,
  :foreign_key => :KtoNr,
  :primary_key => :KtoNr,
  :dependent => :destroy,
  :class_name => "KklVerlauf",
  :autosave => true,
  :order => "KKLAbDatum DESC"
```

Bei dieser Beziehung wird definiert, dass eine *OzbKonto*-Instanz zu einem Zeitpunkt nur einer Instanz aus dem Model *KklVerlauf* zugeordnet ist. Die Verbindung erfolgt über den Primärschlüssel *:KtoNr* auf Seite des *KklVerlauf* Models und dem Fremdschlüssel *:KtoNr* auf der Seite des *OzbKonto* Models.

Der Parameter *:dependent* gibt mit dem Schlüssel *:destroy* an, dass in diesem Fall der zugeordnete Datensatz gelöscht wird und weist diesen Datensatz wiederum an, seine abhängigen Datensätze zu löschen und dieser wiederum seine. Der Aufruf ist hier rekursiv und sorgt dafür, dass alle abhängigen Daten vollständig gelöscht werden. Wäre nur das Schlüsselwort *:delete* angegeben, würde auch nur dieser direkte abhängige Datensatz gelöscht werden.

Über die Option *:class_name* wird (meistens nur bei abweichenden Klassennamen) der Name der verbundenen Klasse angegeben.

:autosave definiert das Speichern des *KklVerlauf* Datensatzes, wenn die *OzbKonto*-Instanz die Methode *save* ausführt.

⁴http://guides.rubyonrails.org/association_basics.html

Als letzten Parameter ist `:order` angegeben. Dieser gibt an, dass der erste Datensatz (der Erste, weil `:has_one`) der Menge an Datensätzen, die über `:KtoNr` zusammen gehören und nach `KKLabDatum` absteigend sortiert sind. Die Absteigende Sortierung ist deshalb wichtig, dass beim Zugriff über `OzbKonto` auf `KklVerlauf` auch tatsächlich auf den letzten (also neuesten) Datensatz zugegriffen wird.

3.5 Leserliche Spaltennamen („Humanized Attributes“)

Generell können Spaltennamen für Models auch immer in die Lokalisierungsdateien der Rails-Anwendung geschrieben werden, jedoch bietet sich bei einer Anwendung die hauptsächlich in einem Sprachengebiet verwendet wird, die direkte Implementierung innerhalb der Models an.

Mit Hilfe der statischen Methode `human_attribute`

```
self.human_attribute_name(attr, options={})  
  HUMANIZED_ATTRIBUTES[attr.to_sym] || super  
end
```

erreicht man in Verbindung mit den Konstanten

```
HUMANIZED_ATTRIBUTES = {  
  :attr1 => "Name des Attributes #1",  
  :attr2 => "Name des Attributes #2"  
}
```

vor allem für Spaltennamen eine für den Benutzer leserliche Form der jeweiligen Spalte. Fehlen diese Angaben, so erhält der Benutzer den Namen der Spalten - wie sie als Attribut definiert wurden.

3.6 Leserliche Fehlermeldungen

Um Fehlermeldungen leserlicher zu gestalten, können in den `locales` Einträge hinzugefügt werden, die Attribute eines Models definieren. `Locales` sind Sprachdateien, die zur Laufzeit von der Anwendung gelesen und verwendet werden. Damit lassen sich Übersetzungen in verschiedenen Sprachen hinterlegen.

Fehlermeldungen, die beispielsweise aus abhängigen *Models* stammen erhalten ohne Übersetzung nacheinander die Namen des Pfads vom ersten bis zum letzten Model. Das sieht nicht besonders schön aus und sollte in einer produktiven Umgebung auch so nicht verwen-

det werden. Deshalb kann man unter *config/locales/de.yml* - in diesem Beispiel die deutsche Übersetzung - festlegen, welchen Namen das Attribut eines Models und dessen abhängigen Models' Attributen bekommen soll.

Als Beispiel sei hier die aktuelle YML-Datei gezeigt.

```
de:
  activerecord:
    # all the names for the error messages
    attributes:
      buergschaft:
        Pnr_B: "Bürgschafter:"
        Pnr_G: "Gläubiger:"
        KtoNr: "ZE Konto-Nr.:"
        SichAbDatum: "Beginn:"
        SichEndDatum: "Ende:"
        SichBetrag: "Betrag:"
        SichKurzbez: "Kurzbezeichnung:"
        SachPNR: "Sachbearbeiter:"
```

de gibt an, dass es sich hierbei um die deutsche Übersetzung handelt. Alle Angaben, die unterhalb (und mit zwei Leerzeichen eingerückt) von *activerecord* stehen, sind nur innerhalb der *ActiveRecord*-Klassen (aber allen!) verfügbar. Darunter wird der Name der Klasse (in obigem Beispiel *buergschaft*) angegeben. Darunter wiederum die Attribute der Klasse *Buergschaft* als key/value Paare, wobei der Wert den Namen des Attributes der Fehlermeldung bestimmt. Hiermit werden automatisch alle Fehlermeldungen bezogen auf die Attribute der Klasse *Buergschaft* übersetzt. Es müssen keine weiteren Änderungen vorgenommen werden. Voraussetzung für die korrekte Funktionsweise ist allerdings, dass die railseigenen *FormHelper* verwendet werden.

Haben *ActiveRecord*-Klassen Abhängigkeiten zu anderen Models, so können die Übersetzungen der abhängigen Klassen unter Umständen andere Namen haben, als wenn diese direkt verwendet werden. Das macht auch dann Sinn, wenn es evtl. mehrere Attribute gibt, die den gleichen Namen haben. Der Anwender kann dann nicht mehr unterscheiden, welches Attribut gemeint ist.

Als Beispiel sei folgende Übersetzung gegeben:

```
de:
  activerecord:
    # all the names for the error messages
    attributes:
      ee_konto:
        KtoNr: "EE Konto-Nr.:"
        BankID: "Bank ID:"
        Kreditlimit: "Kreditlimit:"
        SachPNR: "Sachbearbeiter:"
        GueltigVon: "Gültig von:"
        GueltigBis: "Gültig bis:"
        # associated model names
        Bankverbindung:
          one: "Bankverbindung"
          other: "Bankverbindungen"
          BankKtoNr: "Bankverbindung > Konto-Nr.:"
          IBAN: "Bankverbindung > IBAN:"
          BLZ: "Bankverbindung > BLZ:"
        Bank:
          one: "Bank"
          other: "Banken"
          BankName: "Bankverbindung > Bankname:"
          BIC: "Bankverbindung > BIC:"
          IBAN: "Bankverbindung > IBAN:"
          BLZ: "Bankverbindung > BLZ:"
```

Hier werden die Attribute der Klasse *Bankverbindung* unterhalb von der Klasse *EeKonto* definiert, damit der Benutzer eine strukturierte Fehlermeldung erhält.

Zusätzlich wurde hier auch eine Pluralisierung verwendet. Das Attribut *one* (unterhalb von *Bankverbindung*) definiert, wie das Wort „Bankverbindung“ in der Einzahl (falls das *EeKonto* nur eine Bankverbindung hat) innerhalb der Fehlermeldung ausgegeben werden soll. Das Attribut *other* definiert den Plural des Wortes Bankverbindung. Das ist immer dann wichtig, wenn eine *has_many Association* verwendet wird und der Benutzer in einem einzigen Formular die Möglichkeit hat, mehrere abhängige Datensätze zu erzeugen.

Eine hierzu passende Fehlermeldung innerhalb der o/ZB-Anwendung hat folgendes Aussehen:

Fehler beim Speichern der Daten:

- Konto-Nr.: Bitte geben Sie eine gültige Kontonummer an.
- Bankverbindung > BLZ: ist keine ZahlBitte geben Sie eine BLZ an.
- Bankverbindung > Bankname: Bitte geben Sie einen Banknamen an.
- Bankverbindung > Konto-Nr.: Bitte geben Sie eine gültige Bankkonto-Nummer an (nur Zahlen 0-9).
- Bankverbindung > BLZ: Bitte geben Sie eine BLZ an.
- Kreditlimit: Bitte geben Sie ein gültiges Kreditlimit ein.
- Konto-Nr. Bitte geben Sie eine gültige Kontonummer an.

Abbildung 1: Fehlerausgabe einer strukturierten und korrekt übersetzten Fehlermeldung

3.7 Leserliche Formular-Labels

Analog zu den Übersetzungen der Fehlermeldungen, gibt es auch Übersetzungen zu den *FormHelper* Klassen. Diese müssen sich nicht von den Fehlermeldungen unterscheiden. Allerdings wäre es sinnvoll bei verschachtelten Attributen, einen kürzeren Namen zu wählen und stattdessen diese Formularfelder unter einer passenden Überschrift zu platzieren.

Ein Beispiel für die Gliederung findet sich nachfolgend.

```
de:
  helpers:
    # all the names for the label form helper
    label:
      buergschaft:
        Pnr_B: "Bürgschafter*:"
        Pnr_G: "Gläubiger*:"
        KtoNr: "ZE Konto-Nr.:"
        SichAbDatum: "Beginn*:"
        SichEndDatum: "Ende*:"
        SichBetrag: "Betrag*:"
        SichKurzbez: "Kurzbezeichnung:"
        SachPNR: "Sachbearbeiter:"
```

Unterhalb von *de* -> *helpers* -> *label* werden hier zuerst die Models und dann deren Attribute als *key/value* Paare definiert.

4 Beschreibung der Geschäftsvorfälle (PH/AL)

Die nachfolgenden Abschnitte enthalten Screenshots, die den Ablauf und die Funktionsweise der einzelnen Geschäftsvorfälle genauer beschreiben und erklären. Es wurde stets darauf geachtet, dass sich die Usability und die Anordnungen der einzelnen Stilelemente (Buttons, Icons, Links, ...) an den selben Positionen befinden und auch in Ihrer Handhabung konsistent sind. Rote Buttons signalisieren beispielsweise Aktionen, die tiefere Auswirkungen auf die Anwendung besitzen als bspw. gelbe oder gar grüne Buttons. Gelbe Buttons werden überall dort verwendet, wo sich Daten ändern, dunkelblaue Buttons dort, wo Daten hinzugefügt werden, Grüne und Hellblaue greifen meist nur lesend auf Daten zu. Graue Buttons dienen der Navigation innerhalb der Anwendung.

Die meisten Bereiche der Anwendung besitzen bereits ein funktionsfähiges Rechtemanagement und verwehren Benutzern, die nicht ausreichend Rechte besitzen den Zugriff. Das gilt insbesondere für angezeigte Links.

Die einzelnen CRUD-Funktionen (Create, read, update und delete) der *Kontenverwaltung* und *Bürgschaften* arbeiten nach dem RESTful-Prinzip. Die einzelnen HTTP-Request-Anfragen an den Webserver unterscheiden zwar nur GET und POST, Rails bindet aber automatisch den Parameter „*_method*“ an eine Anfrage, wenn dabei Daten geändert oder gelöscht werden sollen. GET-Anfragen dienen dem reinen Lesen von Daten und POST-Anfragen dazu, Daten hinzuzufügen, zu ändern (*_method=put*) und zu löschen (*_method=delete*). Dieses Prinzip sollte auch für zukünftige Erweiterungen beibehalten werden.

Beispielsweise können keine Datensätze mehr durch einen einfachen Klick auf einen Link gelöscht werden. Stattdessen muss innerhalb einer Seite ein Formularbutton angeklickt werden, der eine POST-Anfrage anstößt. Das ist sicherer und macht innerhalb einer Rails-Anwendung mehr Sinn.

4.1 Kontenverwaltung (OZB-Konten: EE-/ZE-Konten) (PH)

Die Kontenverwaltung stellt neben der Mitgliederverwaltung einen zentralen Punkt innerhalb der o/ZB-Webanwendung dar. Nachfolgend wird auf die Designentscheidungen und Implementierungsdetails eingegangen, um eine möglichst einfache Einarbeitung für nachfolgende Entwickler zu geben.

Zu finden ist die Kontenverwaltung der OZB-Konten über die Mitgliederverwaltung, bei der man sich zunächst ein Mitglied auswählt, um dann dessen Konten mit einem Klick auf Konten anzeigen lassen kann.

Verwaltung: Mitglieder

Alle Mitglieder anzeigen			
Schnellsuche			
Neue Person hinzufügen			
◆ Mitglied-Nr.	◆ Name, Vorname	◆ Rolle	◆ Aufnahmedatum
5	Seidel, Wolfgang	Gesellschafter	2005-01-21
6	Juhas, Anton	Gesellschafter	2005-01-21
7	Ochs, Armin	Gesellschafter	2005-01-21
8	Peckmann, Ulrich	Gesellschafter	2005-01-21
9	Sander, Uwe	Gesellschafter	2005-01-21
10	Schmitz, Christoph M.	Gesellschafter	2005-01-21
11	Schmitz, Monika	Mitglied	2005-01-21
12	Kienle, Hannelore	Mitglied	2005-01-21
13	Kienle, Tassilo	Gesellschafter	2005-01-21
14	Hamann, Rolf	Mitglied	2005-01-21
15	Mequid-Haag, Käte	Mitglied	2005-01-21

Abbildung 2: Mitgliederliste

Auf der daraufhin folgenden Seite hat der Anwender die Möglichkeiten das Mitglied zu verwalten, sowie neue Konten (EE- oder ZE-Konten) anzulegen, zu ändern oder zu löschen, sowie deren KKL-Verlauf anzuzeigen. Alle Funktionen wurden implementiert und getestet. Sie sollten korrekt funktionieren. Die Anzeige eines Kontoauszuges ist leider noch nicht möglich und muss noch implementiert werden.

Kienle, Tassilo

Details

Personaldaten

Kontaktdaten

Rolle

Sonderrechte

Bürgschaften

OZB-Konten

Personaldaten

Mitgliedsnummer:

13

Name:

Kienle, Tassilo

Geburtsdatum:

1982-08-11

Vermerk:

Antragsdatum:

2004-09-01

Aufnahmedatum:

2005-01-21

Austrittsdatum:

Rolle

Rolle:

Gesellschafter

FA Steuernummer:

133/133/133

FA Lfd. Nr.:

9

Wohnsitzfinanzamt

Stuttgart II

Notar:

Beurkundungsdatum:

Kontaktdaten

Email:

Tel.:

0711-12345-6789

Mobil:

09 30-12345-67 89

Fax:

0711-1234-5678

Adresse:

Lehrstrasse 101, 70372 Stuttgart

Zurück

Person sicher löschen

Abbildung 3: Detailansicht eines Mitglieds

Über den Reiter „OZB-Konten“ erreicht man die Übersicht aller Konten des Mitglieds und erhält dadurch nachfolgende Ansicht.

Kienle, Tassilo

Details Personaldaten Kontaktdaten Rolle Sonderrechte Bürgschaften OZB-Konten

EE-Konten

EE Konto-Nr.	Bank Konto-Nr.	BLZ	Bankname	Kreditlimit	
70013	123456789	12345678	Bank X	5000	Kontoauszug KKL-Verlauf Ändern Löschen
60013	987654321	98765432	Bank Y	1000	Kontoauszug KKL-Verlauf Ändern Löschen
50013	111111111	11111111	Bank Z	2000	Kontoauszug KKL-Verlauf Ändern Löschen

Zurück

EE-Konto hinzufügen

ZE-Konten

ZE Konto-Nr.	EE Konto-Nr.	ZE-Nr.	Betrag	Laufzeit	
10013	70013	D110125	10000	2 Monate	Kontoauszug KKL-Verlauf Ändern Löschen
20013	60013	D100331	5000	0 Monate	Kontoauszug KKL-Verlauf Ändern Löschen
30013	50013	D100430	10000	0 Monate	Kontoauszug KKL-Verlauf Ändern Löschen
10013	70013	D110125	10000	2 Monate	Kontoauszug KKL-Verlauf Ändern Löschen

Zurück

ZE-Konto hinzufügen

Abbildung 4: Kontenansicht aller OZB-Konten

Über den Button „Kontoauszug“ soll es später möglich sein, den Kontoauszug für ein Mitglied und einem bestimmten Konto anzeigen zu können. Über den Button „KKL-Verlauf“ erhält man eine Übersicht über die bisherigen Kontenklassen, denen dieses Konto zugewiesen war und ist. Nachfolgender Screenshot (Abbildung 5) zeigt den zeitlichen Verlauf. Bei jeder Änderung der Kontenklasse wird ein neuer Datensatz mit der Klasse und dem Datum angelegt.

Kienle, Tassilo

Details
Personaldaten
Kontaktdaten
Rolle
Sonderrechte
Bürgschaften
OZB-Konten

KKL-Verlauf > Konto 70013

↕ Ab Datum	↕ Kontenklasse
2005-01-01	A
2008-01-01	B
2008-08-01	C
2009-01-01	B

Zurück

Abbildung 5: Kontenklassenverlauf

Über den Button „Zurück“ gelangt der Benutzer wieder auf die zuvor aufgerufene Seite. Dort hat er die Möglichkeit über den Button „*EE-Konto hinzufügen*“ ein neues *EE-Konto* hinzuzufügen. Der Screenshot in Abbildung 6 zeigt das Formular und dessen Eingabefelder, die für einen vollständigen *EE-Konto* Datensatz notwendig sind. Werden bei der Eingabe nicht alle Felder korrekt ausgefüllt, so werden Fehlermeldungen für die benötigten Felder angezeigt und der Benutzer zu einer weiteren Eingabe aufgefordert.

Für die Datumauswahl wird das jQuery Widget Datepicker verwendet, das die Datumauswahl erleichtert. In Kapitel 6 wird hierauf näher eingegangen.

OZB-Konto

Konto-Nr.:

Einrichtungsdatum:

2012-09-24

Währung:

Kontenklasse:

KKL Ab Datum:

EE-Konto

Kreditlimit:

September 2012

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Abbildung 6: jQuery's Datepicker Widget

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Bürgschaften](#) [OZB-Konten](#)

Neues EE-Konto hinzufügen

OZB-Konto

Konto-Nr.:	<input type="text"/>
Einrichtungsdatum:	<input type="text" value="2012-09-24"/>
Währung:	<input type="text" value="STR"/>
Kontenklasse:	<input type="text" value="Klasse A - 100.00%"/> ▼
KKL Ab Datum:	<input type="text" value="2012-09-24"/>

EE-Konto

Kreditlimit:	€ <input type="text"/>
--------------	------------------------

Bankverbindung

Bank Konto-Nr.:	<input type="text"/>
BLZ:	<input type="text"/>
Name der Bank:	<input type="text"/>
BIC:	<input type="text"/>
IBAN:	<input type="text"/>

Felder die mit einem * markiert sind müssen ausgefüllt werden.

Abbildung 7: EE-Eingabeformular

Die *Konto-Nr.* beschreibt das Konto eindeutig und kann nur einmal vergeben werden. Das *„Einrichtungsdatum“* gibt den Zeitpunkt an, an dem dieses Konto eingerichtet wurde oder werden soll. Die *„Währung“* gibt die verwendete Währung für das Konto an. Die *„Kontenklass“e* beschreibt die Klasse, die diesem Konto zugeordnet ist. Das zugehörige Datum bestimmt *„KKL Ab Datum“* und zeigt zusätzlich an, ab wann diese Klasse gelten soll. Das *„Kreditlimit“* gibt den Kreditrahmen an und die Bankverbindungsfelder beschreiben die Bankdaten eines Girokontos bei einem Kreditinstitut.

Über den Button *„Speichern“* lassen sich alle Eingaben speichern und der Benutzer gelangt zur Übersichtsseite aller Konten für das zuvor ausgewählte Mitglied.

Wie bereits erwähnt, werden auch Fehleingaben durch Ausgabe einer entsprechenden Fehlermeldung angezeigt. Als Beispiel sei hier ein exemplarischer Fehler dargestellt, bei dem kein Feld ausgefüllt wurde.

Fehler beim Speichern der Daten:

- Konto-Nr.: Bitte geben Sie eine gültige Kontonummer an.
- Bankverbindung > BLZ: ist keine ZahlBitte geben Sie eine BLZ an.
- Bankverbindung > Bankname: Bitte geben Sie einen Banknamen an.
- Bankverbindung > Konto-Nr.: Bitte geben Sie eine gültige Bankkonto-Nummer an (nur Zahlen 0-9).
- Bankverbindung > BLZ: Bitte geben Sie eine BLZ an.
- Kreditlimit: Bitte geben Sie ein gültiges Kreditlimit ein.
- Konto-Nr. Bitte geben Sie eine gültige Kontonummer an.

Abbildung 8: Fehler bei vollständig leerem EE-Formular

Werden hingegen vollständig korrekte Eingaben gespeichert, wird der Benutzer auf die Übersichtsseite mit allen Konten des Benutzers weitergeleitet. Dort erscheint oberhalb der Auflistung eine Meldung, über das erfolgreiche Anlegen eines neuen Datensatzes.

OZBKonto erfolgreich angelegt. ✕

EE-Konten

↕ EE Konto-Nr.	↕ Bank Konto-Nr.	↕ BLZ	↕ Bankname	↕ Kreditlimit	
70013	00000000	10000000	0000 BANK	10000	Kontoauszug KKL-Verlauf Ändern Löschen
60013	00000000	10000000	0000 BANK	10000	Kontoauszug KKL-Verlauf Ändern Löschen
50013	00000000	10000000	0000 BANK	10000	Kontoauszug KKL-Verlauf Ändern Löschen
34341123	5435345	34341123	345345345	99999.99	Kontoauszug KKL-Verlauf Ändern Löschen

Zurück
EE-Konto hinzufügen

Abbildung 9: Erfolgreiches Speichern eines neuen Kontos

Analog zum Anlegen eines neuen Datensatzes, können auch bestehende Datensätze über den Button „Ändern“ bearbeitet werden. Dabei erscheint das gleiche Formular, wie auch bei einer Neuanlage - jedoch bereits vorausgefüllt mit den vorhandenen Daten. Beim Speichern eines bereits existierenden Datensatzes werden zur Historisierung neue Datensätze angelegt und die bisherigen als nicht mehr aktuell gekennzeichnet. Dazu später mehr.

Das Anlegen eines *ZE-Kontos* funktioniert analog zu einem *EE-Konto*. Das Formular erscheint nach einem Klick auf den Button „ZE-Konto“ hinzufügen und hat den folgenden Aufbau:

Details

Personaldaten

Kontaktdaten

Rolle

Sonderrechte

Bürgschaften

OZB-Konten

Neues ZE-Konto hinzufügen

OZB-Konto

Konto-Nr.*:

Einrichtungsdatum*:

2012-09-24

Währung*:

STR

Kontenklasse*:

Klasse A - 100.00%

KKL Ab Datum*:

2012-09-24

ZE-Konto

EE Konto-Nr.*:

[00000] Gängiges, Osnabrück

Zahlungsmodus*:

M

Projekt:

Hauskauf, Hauserhaltung

Tilgungsrate*:

0.0

ZE-Nr.*:

Ansparrate*:

€ 0.0

Gültig ab:

KDU-Rate*:

€ 0.0

Gültig bis:

RDU-Rate*:

€ 0.0

Betrag*:

€

Status*:

A

Laufzeit*:

Felder die mit einem * markiert sind, müssen ausgefüllt werden.

Zurück

Speichern

Abbildung 10: Formular des ZE-Konto

4.1.1 Meine Konten (EE-/ZE-Konten) (PH)

Ein Mitglied, das sich am System angemeldet hat, landet auf der Seite „*Meine Konten*“ und hat dort direkten Zugriff auf seine EE- und ZE-Konten. Diese Ansicht entspricht derselben wie auch im Verwaltungsbereich, jedoch mit eingeschränkten Funktionen. So kann das Mitglied nur seine Konten und einen Kontoauszug für jedes Konto ansehen. Weitere Optionen sollen nicht möglich sein.

Meine Konten

Guten Tag, Tassilo Kienle

EE-Konten

◆ EE Konto-Nr.	◆ Bank Konto-Nr.	◆ BLZ	◆ Bankname	◆ Kreditlimit	
70013	123456789	12345678	ABC BANK	50000	Kontoauszug
60013	987654321	98765432	DEF BANK	80000	Kontoauszug
50013	456789012	45678901	GHI BANK	40000	Kontoauszug
34341123	098765432	09876543	JKL BANK	200000	Kontoauszug

ZE-Konten

◆ ZE Konto-Nr.	◆ EE Konto-Nr.	◆ ZE-Nr.	◆ Betrag	◆ Laufzeit	
10013	70013	100001	100000	2 Monate	Kontoauszug
20013	60013	200001	150000	0 Monate	Kontoauszug
30013	50013	300001	800000	0 Monate	Kontoauszug
10013	70013	400001	200000	2 Monate	Kontoauszug

Abbildung 11: Übersicht der eigenen Konten nach dem Login

4.1.2 Bürgschaften

Über den Reiter „*Verwaltung*“, „*Mitglieder*“, (Auswahl einer Person), „*Bürgschaften*“, können Bürgschaften für dieses Mitglied verwaltet werden. Die Übersichtsseite sieht wie folgt aus.

Kienle, Tassilo

The screenshot shows the 'Bürgschaften' (Guarantees) overview page for Tassilo Kienle. At the top, there is a navigation bar with tabs: Details, Personaldaten, Kontaktdaten, Rolle, Sonderrechte, Bürgschaften (selected), and OZB-Konten. Below the navigation bar, the title 'Bürgschaften' is displayed. A table with columns: Gläubiger, ZE Konto-Nr., Kurzbezeichnung, Beginn, Ende, and Betrag. The table is empty, and a message below it states: 'Es sind noch keine Bürgschaften für diese Person vorhanden.' At the bottom, there are two buttons: 'Zurück' and 'Bürgschaft hinzufügen'.

Abbildung 12: Übersichtsseite der Bürgschaften für das Mitglied Tassilo Kienle

Über den Button „*Bürgschaft hinzufügen*“ gelangt man zum Eingabeformular und kann dort eine neue Bürgschaft hinzufügen. Dieses Formular hat den folgenden Aufbau:

Kienle, Tassilo

The screenshot shows the 'Neue Bürgschaft hinzufügen' (Add new guarantee) form for Tassilo Kienle. At the top, there is a navigation bar with tabs: Details, Personaldaten, Kontaktdaten, Rolle, Sonderrechte, Bürgschaften (selected), and OZB-Konten. Below the navigation bar, the title 'Neue Bürgschaft hinzufügen' is displayed. The form contains the following fields: 'Bürgschafter*:' with a dropdown menu showing 'Demut, Roman'; 'ZE Konto-Nr.:' with a dropdown menu showing '10013'; 'Beginn*:' with a text input field; 'Ende*:' with a text input field; 'Betrag*:' with a text input field and a Euro symbol; and 'Kurzbezeichnung:' with a text input field. Below the form, a message states: 'Felder die mit einem * markiert sind, müssen ausgefüllt werden.' At the bottom, there are two buttons: 'Zurück' and 'Speichern'.

Abbildung 13: Eingabeformular, um eine neue Bürgschaft zu erfassen

Im Feld „*Bürgschafter*“ muss der Bürge für die ausgewählte Person eingetragen werden. Dieser trägt die Verantwortung für das unter „*ZE Konto-Nr.*“ ausgewählte Konto im Zeitraum von „*Beginn*“ bis „*Ende*“ und in Höhe des unter „*Betrag*“ angegebenen Betrags. Optional kann hier eine „*Kurzbezeichnung*“ angegeben werden, was einem Bemerkungsfeld entspricht.

Nach dem Speichern gelangt auch dieses Formular zurück zur Übersichtsseite aller Bürgschaften des ausgewählten Mitglieds. Werden nicht alle Pflichtfelder ausgefüllt, so erscheint eine entsprechende Fehlermeldung zur Aufforderung der Korrektur der gemachten Eingaben.

Kienle, Tassilo

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Bürgschaften](#) [OZB-Konten](#)

Bürgschaft erfolgreich angelegt.

Bürgschaften

↕ Gläubiger	↕ ZE Konto-Nr.	↕ Kurzbezeichnung	↕ Beginn	↕ Ende	↕ Betrag	
Demut, Roman	10013	Test	2012-09-01	2012-12-31	500.0	Ändern Löschen

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

Abbildung 14: Nach dem erfolgreichen Anlegen einer Bürgschaft erscheint eine Erfolgsmeldung

Bürgschaften dürfen aber ebenso geändert, wie gelöscht werden. Das Ändern führt zum selben Formular, wie auch die Neuanlage - jedoch mit bereits vorausgefülltem Formular. Über „*Löschen*“ wird der Datensatz unwiderruflich - aber erst durch Bestätigung des Benutzers - endgültig gelöscht.

4.1.3 Routen

Die hier verwendeten Routen wurden vollständig umgeschrieben. Hier wäre für die Zukunft noch zu vermerken, dass diese Routen genauer überprüft werden sollten und evtl. sollte auch jede CRUD-Aktion durch eine Rails Resource ersetzt werden.

Die einzelnen Zuordnungen von Routen zu Controllern werden im nächsten Abschnitt beschrieben.

4.1.4 OzbKontoController

Der *OzbKontoController* verwaltet alle OZB-Konten inklusive den EE- und ZE-Konten, sowie den zugehörigen Kontenklassenverläufe. In diesem Abschnitt wird beschrieben, welche Methoden, Models und Views mit diesem Controller zusammenhängen.

4.1.4.1 Index

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Konten*
- Verwendete Methode: *OzbKontoController::index*
- Verwendete Models: *OzbKonto*
- Verwendete View:
 - */views/ozb_konto/index.html.erb*
 - */views /verwaltung/_navigation.html.erb (Partial)*
 - */views/application/_flash_notifier.html.erb (Partial)*
 - */views/ozb_konto/_index_table_ee.html.erb (Partial)*
 - */views/ozb_konto/_index_table_ze.html.erb (Partial)*

Der *Index* zeigt eine Auflistung aller Konten (EE- und ZE-Konten) des ausgewählten Mitglieds an. Hier hat der Benutzer (der die dafür nötigen administrativen Rechte besitzt) die Möglichkeit, die Kontoauszüge und den KKL-Verlauf aller Konten des ausgewählten Benutzers anzusehen sowie das Konto zu ändern und zu löschen. Das Hinzufügen von neuen Konten ist ebenfalls möglich.

4.1.4.2 New

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Konten/:kontotyp/Neu*
- Verwendete Methode: *OzbKontoController::new*
- Verwendete Models: *OzbKonto*
- Verwendete View:
 - */views/ozb_konto/new.html.erb*
 - */views /verwaltung/_navigation.html.erb (Partial)*
 - */views /ozb_konto/_form.html.erb (Partial)*

Die *New* Methode zeigt ein neues Formular für ein EE- oder ZE-Konto (je nach Auswahl des Benutzers) an.

4.1.4.3 Create

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Konten/:kontotyp/Neu'*
- Verwendete Methode: *OzbKontoController::create*
- Verwendete Models: *OzbKonto*
- Verwendete View:
 - */views/ozb_konto/new.html.erb*
 - */views /verwaltung/_navigation.html.erb (Partial)*
 - */views /ozb_konto/_form.html.erb (Partial)*

Die *Create* Methode nimmt die Daten des Formulars per *POST* entgegen und speichert, sofern alle Angaben korrekt waren, als neuen Kontodatensatz in die Datenbank. Über den Parameter *kontotyp* ist zwischen EE- und ZE-Konten zu unterscheiden. Falls alle Formulareingaben korrekt waren, wird auf die *Index* Seite weitergeleitet und eine Erfolgsmeldung ausgegeben. Andernfalls wird das Formular erneut mit den zuvor eingegebenen Werten angezeigt.

4.1.4.4 Edit

- Verwendete Route:
`/Verwaltung/OZBPerson/:Mnr/Konten/:kontotyp/:KtoNr/Aendern`
- Verwendete Methode: `OzbKontoController::edit`
- Verwendete Models: `OzbKonto`
- Verwendete View:
 - `/views/ozb_konto/edit.html.erb`
 - `/views /verwaltung/_navigation.html.erb (Partial)`
 - `/views /ozb_konto/_form.html.erb (Partial)`

Die *Edit* Methode liefert analog der *New* Methode das Formular aus, allerdings mit dem Unterschied, dass hier bereits ein Datensatz ausgewählt wurde und somit die Daten zunächst aus der Datenbank ausgelesen werden müssen und diese dann über die *FormHelper* in das Formular eingefügt werden.

4.1.4.5 Update

- Verwendete Route:
`/Verwaltung/OZBPerson/:Mnr/Konten/:kontotyp/:KtoNr/Aendern`
- Verwendete Methode: `OzbKontoController::update`
- Verwendete Models:
 - `OzbKonto, EeKonto, Bank, Bankverbindung, ZeKonto, KklVerlauf`
- Verwendete View:
 - `/views/ozb_konto/edit.html.erb`
 - `/views /verwaltung/_navigation.html.erb (Partial)`
 - `/views /ozb_konto/_form.html.erb (Partial)`

Bei der *Update* Methode wurde ausnahmsweise im Controller eine manuelle Zuweisung von abhängigen Models erstellt. Das ist damit zu begründen, dass das *gem* (eine Art Plugin fürs Ruby on Rails) *composite_primary_keys* Probleme verursachte, die nicht gelöst werden konnten. In Zukunft könnte man evtl. versuchen diesen Code aufzuräumen, um somit nur mittels der Methode *update_attributes* alle abhängigen Datensätze zu ändern. Bisher wird das in einigen Controllern manuell vorgenommen.

4.1.4.6 Delete

- Verwendete Route:
/Verwaltung/OZBPerson/:Mnr/Konten/:kontotyp/:KtoNr/Loeschen
- Verwendete Methode: *OzbKontoController::delete*
- Verwendete Models: *EeKonto, ZeKonto*
- Verwendete View: -

Die *Delete* Methode löscht den angegebenen Datensatz und alle zugehörigen und abhängen Datensätze durch Löschweitergabe über die Models.

4.1.5 OzbKonto Model

Das OzbKonto Model ist das zentrale Model für Konten. Es werden alle EE- und ZE-Konten samt ihren Abhängigkeiten über dieses Model abgebildet. Nachfolgend folgt eine Erklärung der verwendeten *ActiveRecord Associations*.

4.1.5.1 table_name und primary_keys

Primärschlüssel wirken sich auch auf die Models aus. So wird durch die Angabe von `self.primary_keys = :KtoNr, :GueltigBis` festgelegt, dass ein Datensatz genau dann eindeutig definiert ist, wenn seine Kontonummer und sein *GueltigBis* Zeitstempel angegeben sind. Diese Angabe wirkt sich insbesondere auf die *ActiveRecord::find* Methode aus. Dort werden die Primärschlüssel als Parameter erwartet. Demnach erweitert sich die Methode *find* um den zweiten Parameter *:GueltigBis*.

So wird es mit dieser Angabe möglich, einen eindeutigen *OzbKonto*-Datensatz zu suchen, indem man die Methode wie folgt verwendet:

```
OzbKonto.find(700728, "9999-12-31 23:59:59")
```

Hierbei wird die aktuell gültige Kontonummer 700728 ausgelesen. Analog zur *find* Methode kann auch

```
OzbKonto.where(:Mnr => mnr, :GueltigBis => "9999-12-31 23:59:59")
```


verwendet werden. Wobei darauf zu achten ist, dass die *where* Methode ein Array an *ActiveRecord* Daten zurückliefert, während *find* direkt ein *ActiveRecord* Objekt liefert.

Über das Attribut `self.table_name = „ozbkonto“` legt man den Namen der zugrundeliegenden Tabelle in der Datenbank fest. In diesem Falle wird auf die Tabelle *ozbkonto* zugegriffen.

4.1.5.2 *alias_attribute*

Mit *alias_attribute* lassen sich Aliase für Attribute definieren, um mehrere Schreibweisen zu erlauben. Insbesondere wegen häufigen Problemen zwischen Linux und Windows Betriebssystemen wurden diese für jedes Model definiert und wie folgt definiert:

```
# aliases
alias_attribute :ktoNr, :KtoNr
alias_attribute :mnr, :Mnr
alias_attribute :ktoEinrDatum, :KtoEinrDatum
alias_attribute :waehrung, :Waehrung
alias_attribute :wSaldo, :WSaldo
alias_attribute :pSaldo, :PSaldo
alias_attribute :saldoDatum, :SaldoDatum
```

Als Beispiel kann auf das Attribut Kontonummer (*KtoNr*) über *OzbKonto.ktoNr* aber auch mit *OzbKonto.KtoNr* zugegriffen werden.

4.1.5.3 *Associations*

Für die Verbindungen zu anderen Models wurden die nachfolgenden Associations definiert.

```
belongs_to :ozb_person,
:foreign_key => :Mnr
```

Beschreibt die Verbindung zu einer einzigen *OzbPerson*, zu der dieses *OzbKonto* zugehörig ist. Ohne diese Person, kann dieses Konto nicht existieren. Der Fremdschlüssel ist auf Seiten von *OzbKonto* die Spalte/Attribut *:Mnr*.

```
has_many :buchung,
:foreign_key => :KtoNr,
:dependent => :destroy
```

Ein *OzbKonto* besitzt mehrere Buchungen, die über den Fremdschlüssel *KtoNr* miteinander verknüpft werden. Beim Löschen des *OzbKonto* werden auch alle zugehörigen Buchungen (und evtl. weitere Abhängigkeiten, ausgehend von Buchungen) gelöscht.

```
has_one :kkl_verlauf,  
  :foreign_key => :KtoNr,  
  :primary_key => :KtoNr,  
  :dependent => :destroy,  
  :class_name => "KklVerlauf",  
  :autosave => true,  
  :order => "KKLabDatum DESC"
```

Aus Sicht der Anwendung hat ein *OzbKonto* zu einem Zeitpunkt genau einen Datensatz. Das weicht von der Sicht der Datenbank ab, denn es ist für die Anwendung nur wichtig, welcher Kontenklasse das *OzbKonto* aktuell zugeordnet ist. Nur für den Geschäftsvorfall KKL-Verlauf sind alle Datensätze interessant. Deshalb wird an dieser Stelle neben den Primär-/Fremdschlüssel-, der Löschweitergabe und dem Klassennamen eine Sortierung angegeben. Diese Sortierung wird bei jedem Zugriff auf die Menge der KKLVerlauf-Datensätze angewendet. Rails sortiert alle Datensätze, die zu dem *OzbKonto* gehören, absteigend nach der Spalte *KKLabDatum*, wählt wegen *has_one* den ersten Datensatz aus und liefert diesen zurück. Eine aufsteigende Sortierung würde demnach den ältesten Datensatz zurückliefern.

```
has_one :ze_konto,  
  :foreign_key => :KtoNr,  
  :primary_key => :KtoNr,  
  :dependent => :destroy,  
  :class_name => "ZeKonto",  
  :autosave => true,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis]}
```

Ein *OzbKonto* hat zu einem bestimmten Zeitpunkt genau ein zugeordnetes *ZeKonto*. Hier wird ähnlich dem KKL-Verlauf verfahren, allerdings mit dem Unterschied, dass ein *ZE-Konto* genau dann zu einem *OZB-Konto* gehört, wenn dessen *GueltigBis* Zeitstempel identisch sind und die Fremdschlüssel zueinander passen. Damit bei Auswahl eines historisierten Datensatzes nicht das aktuellste *ZE-Konto* für dieses Konto ausgewählt wird, sondern ebenfalls das zugehörige historisierte Konto, wird eine Einschränkung über den Parameter *:conditions* bestimmt. Das bedingt allerdings auch, dass zugehörige Datensätze nur mit exakt gleichem Zeitstempel gespeichert werden dürfen. Bereits eine Sekunde Abweichung zerstört die Abhängigkeit.

```
has_one :ee_konto,  
  :foreign_key => :KtoNr,  
  :primary_key => :KtoNr,  
  :dependent => :destroy,  
  :class_name => "EeKonto",  
  :autosave => true,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Das EE-Konto ist analog dem ZE-Konto konfiguriert.

```
belongs_to :sachbearbeiter,  
  :foreign_key => :Pnr,  
  :primary_key => :SachPNR,  
  :class_name => "Person"
```

Die Abhängigkeit zu einem Sachbearbeiter benötigt keine Historisierung, sondern kann über simple Angaben des Fremd-/Primärschlüssels erfolgen. Bei dieser Beziehung ist es wichtig, den Klassennamen anzugeben, denn Sachbearbeiter existiert nicht als Klassenname.

4.1.5.4 *nested_attributes und attr_accessible*

```
attr_accessible :KtoNr, :Mnr, :KtoEinrDatum, :Waehrung, :WSaldo, :PSaldo,  
:SaldoDatum, :GueltigBis, :SachPNR, :KtoEinrDatum, :ee_konto_attributes,  
:ze_konto_attributes, :kkl_verlauf_attributes  
accepts_nested_attributes_for :ee_konto, :ze_konto, :kkl_verlauf
```

4.1.6 EeKonto Model

Das EeKonto Model verwaltet abstrahiert alle Abläufe, die zwischen den EE-Konten Datensätzen und der Anwendung ablaufen.

4.1.6.1 *table_name und primary_keys*

```
self.table_name = "eekonto"  
self.primary_keys = :KtoNr, :GueltigBis
```

Das EE-Konto besitzt historisierte Daten und benötigt deshalb neben der Kontonummer einen weiteren Primärschlüssel, *GueltigBis*.

4.1.6.2 *alias_attribute*

```
alias_attribute :ktoNr, :KtoNr  
alias_attribute :bankId, :BankID  
alias_attribute :kreditlimit, :Kreditlimit
```

Damit es beim Wechsel von Linux / Windows Betriebssystemen zu keinen Problemen kommt, werden Aliase für Attribute definiert.

4.1.6.3 *Associations*

```
belongs_to :ozb_konto,  
  :primary_key => :KtoNr,  
  :foreign_key => :KtoNr,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Ohne OZB-Konto Datensatz kann kein EE-Konto existieren, deshalb wird hier eine *belongs_to* Beziehung mit dem *OzbKonto* Model definiert. Zusätzlich muss sich der Datensatz des EE-Kontos immer auf den zugehörigen OZB-Konto Datensatz beziehen. Diese Einschränkung erledigt der Parameter *conditions*, der nur den dazugehörigen Datensatz auswählt.

```
belongs_to :Bankverbindung,  
  :primary_key => :ID,  
  :foreign_key => :BankID,  
  :class_name => "Bankverbindung",  
  :autosave => true,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Jedes EE-Konto benötigt eine Bankverbindung, weshalb hier eine 1:1 Beziehung zur Bankverbindung deklariert wird. Da auch Bankverbindungen historisiert sind, wird auch hier eine Einschränkung dafür definiert.

```
has_one :ze_konto,  
  :foreign_key => :KtoNr,  
  :dependent => :destroy,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Ein EE-Konto kann - muss aber nicht - ein ZE-Konto besitzen, deshalb wird hier auch definiert, dass das zugehörige ZE-Konto beim Löschen des EE-Kontos gelöscht werden muss. Ebenfalls findet sich hier wieder die Einschränkung der historisierten Daten.

```
belongs_to :sachbearbeiter,  
  :class_name => "Person",  
  :foreign_key => :Pnr,  
  :primary_key => :SachPNR,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Jedes EE-Konto wird durch einen Sachbearbeiter hinzugefügt oder geändert. Damit es möglich ist, ausgehend vom EE-Konto, auf einen Sachbearbeiter über die Rails eigenen Methoden zuzugreifen, wird auch dieser definiert.

4.1.6.4 *nested_attributes und attr_accessible*

```
accepts_nested_attributes_for :Bankverbindung  
attr_accessible :KtoNr, :BankId, :Kreditlimit, :GueltigVon,  
  :GueltigBis, :Bankverbindung_attributes, :sachbearbeiter_attributes
```

Das Model akzeptiert Attribute für die Bankverbindung und bietet die Massen-Zuweisung für die unter *attr_accessible* angegebenen Attribute. Das bietet den Vorteil, dass Attribute im Controller nicht einzeln dem Objekt zugewiesen werden müssen, sondern bereits über die

Strukturierung des HTML-Formulars. Nur Attribute, die unter *attr_accessible* stehen, dürfen durch Formularparameter gesetzt werden. Das trifft insbesondere für die *ActiveRecord* Methode *update_attributes()* zu.

4.1.7 Bankverbindung Model

4.1.7.1 *table_name* und *primary_keys*

```
self.table_name = "bankverbindung"  
self.primary_keys = :ID, :GueltigBis
```

Bankverbindungen sind historisiert und benötigen deshalb neben der *ID* einen zweiten Primärschlüssel, der die Zeit der Daten angibt.

4.1.7.2 *alias_attribute*

```
alias_attribute :id, :ID  
alias_attribute :pnr, :Pnr  
alias_attribute :bankKtoNr, :BankKtoNr  
alias_attribute :blz, :BLZ  
alias_attribute :iban, :IBAN
```

Auch hier werden wieder Aliase angegeben.

4.1.7.3 *Associations*

```
belongs_to :person,  
  :foreign_key => :Pnr,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Eine Bankverbindung ist direkt mit einer Person verbunden. Die Abhängigkeit wird über das Attribut *GueltigBis* aufgelöst.

```
has_one :ee_konto,  
  :foreign_key => :BankId,  
  :autosave => true,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Eine weitere direkte Verbindung besteht auch mit einem EE-Konto.

```
belongs_to :Bank,  
  :foreign_key => :BLZ,  
  :autosave => true
```

Eine Bankverbindung gehört zu einer Bank, um redundante Daten zu vermeiden.

4.1.7.4 *nested_attributes und attr_accessible*

```
attr_accessible :ID, :Pnr, :BankKtoNr, :BLZ, :IBAN, :GueltigVon, :GueltigBis,  
:Bank_attributes  
accepts_nested_attributes_for :Bank, :reject_if => :bank_already_exists
```

Eine Bankverbindung akzeptiert Attribute für den Bank Datensatz. Das sind BLZ, IBAN, BIC und Bankname. Der Parameter *:reject_if* gibt an, dass kein Datensatz für Bank erzeugt werden soll, falls die Methode *bank_already_exists* *true* liefert.

4.1.8 Bank Model

4.1.8.1 *table_name und primary_keys*

```
self.table_name = "bank"  
self.primary_key = :BLZ
```

Der Name der Tabelle Bank und der einfache Primärschlüssel BLZ werden direkt in den ersten Zeilen Code angegeben.

4.1.8.2 *alias_attribute*

```
alias_attribute :bic, :BIC
```

Für den BIC wurde wegen Kompatibilität ein Alias definiert.

4.1.8.3 Associations

```
has_many :Bankverbindung,  
  :foreign_key => :BLZ,  
  :dependent => :destroy
```

Um direkt alle Bankverbindungen einer Bank zu erhalten, wird auch umgekehrt zur Bankverbindung eine 1:n Verknüpfung mit den Bankverbindungen definiert. Wird ein Bank-Datensatz gelöscht, so werden auch alle zugehörigen Bankverbindungen gelöscht.

4.1.8.4 nested_attributes und attr_accessible

```
attr_accessible :BLZ, :BIC, :BankName
```

BLZ, BIC und Bankname dürfen über *Mass-Assignment* erzeugt werden.

4.1.9 ZeKonto Model

4.1.9.1 table_name und primary_keys

```
self.table_name = "zekonto"  
self.primary_keys = :KtoNr, :GueltigBis
```

Analog zum EE-Konto hat auch ein ZE-Konto zwei Primärschlüssel, *KtoNr* und *GueltigBis* für die historisierten Daten.

4.1.9.2 alias_attribute

```
alias_attribute :ktoNr, :KtoNr  
alias_attribute :eeKtoNr, :EEKtoNr  
alias_attribute :pgNr, :Pgnr  
alias_attribute :zeNr, :ZENr  
alias_attribute :zeAbDatum, :ZEAbDatum  
alias_attribute :zeEndDatum, :ZEEndDatum  
alias_attribute :zeBetrag, :ZEBetrag  
alias_attribute :laufzeit, :Laufzeit  
alias_attribute :zahlModus, :ZahlModus  
alias_attribute :tilgRate, :TilgRate  
alias_attribute :ansparRate, :AnsparRate  
alias_attribute :kduRate, :KDURate  
alias_attribute :rduRate, :RDURate
```

Aus Kompatibilitätsgründen werden auch hier für alle Attribute Aliase definiert.

4.1.9.3 Associations

```
belongs_to :ozb_konto,  
  :foreign_key => :KtoNr,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Ein ZE-Konto ist zugehörig zu einem OZB-Konto an einem bestimmten Zeitpunkt. Auch hier muss der Zeitstempel identisch zu dem des abhängigen Models sein.

```
belongs_to :ee_konto,  
  :foreign_key => :EEKtoNr,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Neben einem OZB-Konto gehört ein ZE-Konto Datensatz auch noch zu einem EE-Konto. Auch hier sind die Daten - und damit auch die Verknüpfung zwischen den Datensätzen - historisiert.

```
has_many :buergschaft,  
  :foreign_key => :KtoNr  
  
belongs_to :projektgruppe,  
  :inverse_of => :ZEKonto,  
  :foreign_key => :Pgnr
```

Die Verknüpfung zu Bürgschaften und Projektgruppen wird über die obigen Codezeilen definiert.

```
has_one :sachbearbeiter,  
  :class_name => "Person",  
  :foreign_key => :Pnr,  
  :primary_key => :SachPNR,  
  :conditions => proc { ["GueltigBis = ?", self.GueltigBis] }
```

Ein ZE-Konto besitzt - wie auch ein OZB-Konto - einen Sachbearbeiter, der beim Editieren oder Hinzufügen eines neuen Datensatzes automatisch hinzugefügt wird. Hierüber ist immer ersichtlich, wer welchen Datensatz wann geändert/hinzugefügt hat. Auch hier müssen die historisierten Daten verwendet werden. Zusätzlich muss der Klassennamen angegeben werden, in diesem Fall Person.

4.1.9.4 *nested_attributes und attr_accessible*

```
attr_accessible :KtoNr, :EEKtoNr, :Pgnr, :ZENr, :ZEAbDatum, :ZEEndDatum,  
:ZEBetrag,  
:Laufzeit, :ZahlModus, :TilgRate, :AnsparRate, :KDURate,  
:RDURate, :ZEStatus
```

Die oben aufgelisteten Attribute dürfen per *Mass-Assignment* angegeben werden, für alle anderen Attribute müssen die Daten über einen Controller angegeben werden. Das verhindert, das bspw. das Attribut *GueltigBis* über ein HTML-Formular verändert werden kann.

4.1.10 KKL-Verlauf Controller

Hier werden die Methoden des KKL-Verlauf Controllers näher erläutert, sowie deren verwendeten Routen, Models und Views angegeben.

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Konten/:KtoNr/KKLVerlauf*
- Verwendete Methode: *OzbKontoController::verlauf*
- Verwendete Models: *OzbKonto, KklVerlauf*
- Verwendete View:
 - */views/ozb_konto/verlauf.html.erb*

Der KKL-Verlauf zeigt den Verlauf der bisher zugeteilten Kontenklassen des jeweiligen Kontos und informiert ab welchem Datum diese zugeteilt waren.

4.1.11 KklVerlauf Model

4.1.11.1 *table_name und primary_keys*

Das Model *KklVerlauf* besitzt den Tabellennamen *kklverlauf*.

Nach Datenbankschema besitzt die Tabelle *kklverlauf* zwei Primärschlüssel, wobei jedoch leider - aufgrund von Problemen mit dem *gem composite_primary_keys* - auf den zweiten Primärschlüssel *KKLAbDatum* verzichtet werden muss. Hier wäre evtl. ein *before_save* Call-back möglich, der beim Ändern eines Datensatzes, nur einen neuen Datensatz hinzufügt, statt den alten zu löschen.

4.1.11.2 *alias_attribute*

```
alias_attribute :kkl, :KKL  
alias_attribute :kklAbDatum, :KKLABDatum
```

Analog zu den anderen Aliasen.

4.1.11.3 *Associations*

```
# associations  
belongs_to :ozb_konto,  
  :foreign_key => :KtoNr,  
  :class_name => "OzbKonto"  
  
belongs_to :kontenklasse,  
  :foreign_key => :KKL
```

Hier erfolgen nur einfache Verknüpfungen mit anderen Models, auf eine Erklärung wird an dieser Stelle verzichtet. Es sei auf die vorherigen Seiten der Dokumentation verwiesen.

4.2 Verwaltung von Bürgschaften (PH)

Hier werden die Methoden des Bürgschaften Controllers näher erläutert, sowie deren verwendeten Routen, Models und Views angegeben.

4.2.1 Routen

- */Verwaltung/OZBPerson/:Mnr/Buergschaften* => *buergschaft#index*
- */Verwaltung/OZBPerson/:Mnr/Buergschaften/Neu* => *buergschaft#new*
- */Verwaltung/OZBPerson/:Mnr/Buergschaften/Neu* => *buergschaft#create*
- */Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Aendern* => *buergschaft#edit*
- */Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Aendern* => *buergschaft#update*
- */Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Loeschen* => *buergschaft#delete*

4.2.2 Bürgschaften Controller

4.2.2.1 Index

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Buergschaften*
- Verwendete Methode: *BuergschaftController::index*
- Verwendete Models: *Buergschaft*
- Verwendete View:
 - */views/buergschaft/index.html.erb*

Die *Index* Methode liefert eine Übersicht über alle Bürgschaften.

4.2.2.2 New

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Buergschaften/Neu*
- Verwendete Methode: *BuergschaftController::new*
- Verwendete Models: *Buergschaft*
- Verwendete View:
 - */views/buergschaft/new.html.erb*
 - */views/buergschaft/_form.html.erb (Partial)*

Die *New* Methode zeigt das Eingabeformular an.

4.2.2.3 Edit

- Verwendete Route:
/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Aendern
- Verwendete Methode: *BuergschaftController::edit*
- Verwendete Models: *Buergschaft*
- Verwendete View:
 - */views/buergschaft/edit.html.erb*
 - */views/buergschaft/_form.html.erb (Partial)*

Die *Edit* Methode ruft den ausgewählten Datensatz von der Datenbank ab und zeigt dessen Daten vorausgefüllt an.

4.2.2.4 Create

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Buergschaften/Neu*
- Verwendete Methode: *BuergschaftController::create*
- Verwendete Models: *Buergschaft*
- Verwendete View:
 - */views/buergschaft/new.html.erb*
 - */views/buergschaft/_form.html.erb (Partial)*

Die *Create* Methode speichert bei korrekten Formulareingaben einen neuen Datensatz ab und leitet den Benutzer auf die Übersichtsseite (Index) weiter. Dort wird zusätzlich eine Meldung angezeigt. Bei ungültigen Eingaben wird das Formular mit den falschen Eingaben erneut angezeigt und einen Hinweis zu den Fehlern angegeben.

4.2.2.5 Update

- Verwendete Route:
/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Aendern
- Verwendete Methode: *BuergschaftController::update*
- Verwendete Models: *Buergschaft*
- Verwendete View:
 - */views/buergschaft/edit.html.erb*
 - */views/buergschaft/_form.html.erb (Partial)*

Die *Update* Methode aktualisiert einen zuvor ausgewählten und durch den Benutzer geänderten Datensatz, prüft ihn auf Gültigkeit und speichert ihn dann in der Datenbank ab. Bei erfolgreicher Speicherung wird der Benutzer auf die Übersichtsseite weitergeleitet und erhält eine Meldung über die Änderung der Daten. Waren einige Eingaben ungültig, so wird dem Benutzer das Formular mit den fehlerhaften Werten erneut angezeigt und eine Auflistung mit Fehlern ausgegeben.

4.2.2.6 Delete

- Verwendete Route:
/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/Loeschen
- Verwendete Methode: *BuergschaftController::delete*
- Verwendete Models: *Buergschaft*
- Verwendete View: -

Die *Delete* Methode löscht einen ausgewählten Datensatz aus der Datenbank.

4.3 Mitgliederverwaltung (AL)

Die Mitgliederverwaltung der o/ZB Anwendung erfolgt auf zwei Weisen. Einerseits kann ein eingeloggter Benutzer seine persönlichen Daten selbst verwalten. Andererseits können Benutzer, die über Sonderberechtigungen verfügen, die Daten anderer Benutzer bearbeiten.

4.3.1 Sonderberechtigungen

Ein Benutzer kann über Sonderberechtigungen verfügen, die einem innerhalb der Anwendung erlauben, bestimmte Aktionen, basierend auf den Geschäftsprozessen der o/ZB, durchzuführen. Die folgende Tabelle zeigt, welche Aktionen ein Benutzer in einer entsprechenden Verwaltungsgruppe ausführen darf.

ID	Beschreibung	Admin AG-IT	Mitglieder AG-MV	Finanz AG-RW	Projekte AG-ZE	Öffentlichkeit AG-ÖA
1	Alle Mitglieder anzeigen	1	1	1	1	1
2	Details einer Person anzeigen	1	1	1	1	1
3	Mitglieder hinzufügen	1	1	0	0	0
4	Mitglieder bearbeiten	1	1	0	0	0
5	Mitglieder löschen	1	1	0	0	0
6	Rolle eines Mitglieds zum Gesellschafter ändern	1	1	0	0	0
7	Mitglied Administratorrechte hinzufügen	1	0	0	0	0
8	Kontenklassen hinzufügen	1	0	1	0	0
9	Kontenklassen bearbeiten	1	0	0	0	0
10	Kontenklassen löschen	1	0	0	0	0
11	Alle Konten anzeigen	1	1	1	1	1
12	Details eines Kontos anzeigen	1	1	1	1	1
13	Einlage/Entnahmekonten hinzufügen	1	0	1	0	0
14	Einlage/Entnahmekonten bearbeiten	1	0	1	0	0
15	Zusatzentnahmekonten hinzufügen	1	0	1	0	0
16	Zusatzentnahmekonten bearbeiten	1	0	1	0	0
17	Bürgschaften anzeigen	1	1	1	1	1
18	Bürgschaften hinzufügen	1	0	0	1	0
19	Bürgschaften bearbeiten	1	0	0	1	0

Abbildung 15: Rechteverwaltung

4.3.2 Eigene Daten verwalten

Verfügt die angemeldete Person über keine Sonderberechtigungen, kann diese nur ihre eigene Daten verwalten. Dazu gehören folgende Aktionen mit jeweils eigener view:

- *Übersicht eigener Daten*
- *Personaldaten bearbeiten*
- *Kontaktdaten bearbeiten*
- *Rolle bearbeiten*
- *Logindaten bearbeiten*

Aus gestalterischen Gründen wurde hier die Aktion "Mitglieder bearbeiten" (siehe Abbildung 15) in vier Teilen aufgeteilt: Personaldaten, Kontaktdaten, Rolle und Logindaten bearbeiten.

4.3.2.1 Routen für die Verwaltung eigener Daten

<i>/MeineDaten</i>	=>	<i>o_z_b_person#detailsOZBPerson</i>
<i>/MeineDaten/Details</i>	=>	<i>o_z_b_person#detailsOZBPerson</i>
<i>/MeineDaten/Personaldaten</i>	=>	<i>o_z_b_person#editPersonaldaten</i>
<i>/MeineDaten/Personaldaten</i>	=>	<i>o_z_b_person#updatePersonaldaten</i>
<i>/MeineDaten/Kontaktdaten</i>	=>	<i>o_z_b_person#editKontaktdaten</i>
<i>/MeineDaten/Kontaktdaten</i>	=>	<i>o_z_b_person#updateKontaktdaten</i>
<i>/MeineDaten/Rolle</i>	=>	<i>o_z_b_person#editRolle</i>
<i>/MeineDaten/Rolle</i>	=>	<i>o_z_b_person#updateRolle</i>
<i>/MeineDaten/Logindaten</i>	=>	<i>o_z_b_person#editLogindaten</i>
<i>/MeineDaten/Logindaten</i>	=>	<i>o_z_b_person#updateLogindaten</i>

4.3.2.2 OZBPerson Controller

Der *OZBPersonController* beinhaltet die nachfolgenden Methoden:

- Die *detailsOZBPerson* Methode liefert eine Übersicht über alle personenbezogenen Daten.
- Die *editPersonaldaten*, *editKontaktdaten*, *editRolle* und *editLogindaten* Methoden liefern jeweils die ausgewählten Datensätze aus der Datenbank, die im entsprechenden Formularen zur Bearbeitung der Personaldaten vorausgefüllt werden.
- Die *updatePersonaldaten*, *updateKontaktdaten*, *updateRolle* und *updateLogindaten* Methoden prüfen jeweils die im Formular eingegebenen Daten auf Gültigkeit und aktualisieren die entsprechenden Datensätze in der Datenbank. Bei einer ungültigen Eingabe wird der Benutzer auf die fehlerhafte Eingaben verwiesen. Bei erfolgreicher Speicherung wird der Benutzer auf die Übersichtsseite (*/MeineDaten/Details*) weitergeleitet und erhält eine Meldung über die Änderung der Daten.

4.3.2.3 *DetailsOZBPerson*

- Verwendete Route: */MeineDaten/Details*
- Verwendete Methode: *OZBPersonController::detailsOZBPerson*
- Verwendete Models: *OZBPerson, Person, Adresse, Telefon, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
/views/ozb_person/detailsOZBPerson.html.erb
/views/ozb_person/_navigation.html.erb (Partial)

Die *detailsOZBPerson.html.erb* zeigt eine Detailansicht eigener Daten. Hier werden Personaldaten, Kontaktdaten sowie die Rolle zusammengefasst. Denkbar wäre hier noch sonstige Daten wie z.B. Sonderberechtigungen oder Anzahl Konten anzuzeigen. Die Hilfsnavigation in Form von Tabs bietet die Möglichkeit, die personenbezogenen Daten voneinander getrennt zu bearbeiten.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Personaldaten
Mitgliedsnummer: 13
Name: Kienle, Tassilo
Geburtsdatum: 1982-08-19
Vermerk:
Antragsdatum: 2004-09-01
Aufnahmedatum: 2005-01-21
Austrittsdatum:

Rolle
Rolle: Gesellschafter
FA Steuernummer: 1000000000
FA Lfd. Nr.: 9
Wohnsitzfinanzamt: Stuttgart II
Notar:
Beurkundungsdatum:

Kontaktdaten
Email:
Tel.: 0711-3000-0000
Mobil: 0711-3000-10 10
Fax: 0711-3000-0000
Adresse: Landstrasse 103, 70372 Stuttgart

Zurück

Abbildung 16: Meine Daten - Details

Wurden die Daten erfolgreich bearbeitet, gelangt man zurück zur Detailübersicht, wo eine Erfolgsmeldung angezeigt wird.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Personaldaten wurden erfolgreich aktualisiert. x

Abbildung 17: Meine Daten - Details - Erfolgsmeldung

4.3.2.4 EditPersonaldaten

- Verwendete Route: */MeineDaten/Personaldaten (GET)*
- Verwendete Methode: *OZBPersonController::editPersonaldaten*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/ozb_person/editPersonaldaten.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Die *editPersonaldaten.html.erb* stellt die Form zur Bearbeitung der Personaldaten dar.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Personaldaten bearbeiten

Mitgliedsnummer:	<input type="text" value="13"/>
*Name:	<input type="text" value="Kienle"/>
*Vorname:	<input type="text" value="Tassilo"/>
Geburtsdatum:	<input type="text" value="1983-08-13"/>
Vermerk:	<input type="text"/>
*Antragsdatum:	<input type="text" value="2004-09-01"/>
Aufnahmedatum:	<input type="text" value="2005-01-21"/>
Austrittsdatum:	<input type="text"/>

* Pflichtfelder

Abbildung 18: Meine Daten - Personaldaten bearbeiten

4.3.2.5 UpdatePersonaldaten

- Verwendete Route: */MeineDaten/Personaldaten (POST)*
- Verwendete Methode: *OZBPersonController::updatePersonaldaten*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/ozb_person/editPersonaldaten.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Nach der Ausfüllung und Abschicken der Form (*editPersonaldaten.html.erb*) werden die Eingaben auf die Richtigkeit überprüft. Sind die Eingaben fehlerhaft, wird eine Fehlermeldung angezeigt, weswegen die Personaldaten nicht aktualisiert werden konnten.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Personaldaten wurden nicht aktualisiert:

- Name muss ausgefüllt werden
- Vorname muss ausgefüllt werden
- Antragsdatum muss ausgefüllt werden

Personaldaten bearbeiten

Mitgliedsnummer:	<input type="text" value="13"/>
*Name:	<input type="text"/>
*Vorname:	<input type="text"/>
Geburtsdatum:	<input type="text" value="2005-01-21"/>
Vermerk:	<input type="text"/>
*Antragsdatum:	<input type="text"/>
Aufnahmedatum:	<input type="text" value="2005-01-21"/>
Austrittsdatum:	<input type="text"/>

* Pflichtfelder

Abbildung 19: Meine Daten - Personaldaten bearbeiten - Fehlermeldung

4.3.2.6 EditKontaktdaten

- Verwendete Route: */MeineDaten/Kontaktdaten (GET)*
- Verwendete Methode: *OZBPersonController::editKontaktdaten*
- Verwendete Models: *OZBPerson, Person, Adresse, Telefon*
- Verwendete View:
 - */views/ozb_person/editKontaktdaten.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Die *editKontaktdaten.html.erb* stellt die Form zur Bearbeitung der Kontaktdaten dar.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Kontaktdaten bearbeiten

***Email:**

test1@ozb.eu

Tel.:

07-9-123456789

Mobil:

0151-123456789

Fax:

07-9-123456789

Straße:

Landstraße

Hausnummer:

10

PLZ:

70372

Ort:

Stuttgart

*** Pflichtfelder**

Zurück

Speichern

Abbildung 20: Meine Daten - Kontaktdaten bearbeiten

4.3.2.7 UpdateKontaktdaten

- Verwendete Route: */MeineDaten/Kontaktdaten (POST)*
- Verwendete Methode: *OZBPersonController::updateKontaktdaten*
- Verwendete Models: *OZBPerson, Person, Adresse, Telefon*
- Verwendete View:
 - */views/ozb_person/editKontaktdaten.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Nach der Ausfüllung und Abschicken der Form (*editKontaktdaten.html.erb*) werden die Eingaben auf die Richtigkeit überprüft. Sind die Eingaben fehlerhaft, wird eine Fehlermeldung angezeigt, weswegen die Kontaktdaten nicht aktualisiert werden konnten.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Kontaktdaten wurden nicht aktualisiert:

- Email muss ausgefüllt werden

Kontaktdaten bearbeiten

*Email:

Tel.:

Mobil:

Fax:

Straße:

Hausnummer:

PLZ:

Ort:

Stuttgart

* Pflichtfelder

Zurück

Speichern

Abbildung 21: Meine Daten - Kontaktdaten bearbeiten - Fehlermeldung

4.3.2.8 *EditRolle*

- Verwendete Route: */MeineDaten/Rolle (GET)*
- Verwendete Methode: *OZBPersonController::editRolle*
- Verwendete Models:
 - *OZBPerson, Person, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - */views/ozb_person/editRolle.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Die *editRolle.html.erb* stellt die Form zur Bearbeitung der Rolle dar.

4.3.2.9 *UpdateRolle*

- Verwendete Route: */MeineDaten/Rolle (POST)*
- Verwendete Methode: *OZBPersonController::updateRolle*
- Verwendete Models:
 - *OZBPerson, Person, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - */views/ozb_person/editRolle.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

Nach der Ausfüllung und Abschicken der Form (*editRolle.html.erb*) werden die Eingaben auf die Richtigkeit überprüft. Sind die Eingaben fehlerhaft, wird eine Fehlermeldung angezeigt, weswegen die Rolle nicht aktualisiert werden konnten.

Meine Daten

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Logindaten](#)

Konnte keine Rolle aktualisieren:

- Ausbildungsbezeichnung muss ausgefüllt werden
- Institutname muss ausgefüllt werden
- Studienort muss ausgefüllt werden
- Studienbeginn muss ausgefüllt werden
- Abschluss muss ausgefüllt werden

Rolle bearbeiten

*Rolle:

Student

*Ausbildungsbez.:

*Institut:

*Studienort:

*Studienbeginn:

Studienende:

*Abschluss:

* Pflichtfelder

Zurück

Speichern

Abbildung 22: Meine Daten - Rolle bearbeiten - Fehlermeldung

4.3.2.10 EditLogindaten

- Verwendete Route: `/MeineDaten/Logindaten (GET)`
- Verwendete Methode: `OZBPersonController::editLogindaten`
- Verwendete Models: `OZBPerson, Person`
- Verwendete View:
 - `/views/ozb_person/editLogindaten.html.erb`
 - `/views/ozb_person/_navigation.html.erb (Partial)`

Die `editLogindaten.html.erb` stellt die Form zur Bearbeitung der Logindaten dar. Nächste Gruppe bitte bearbeiten!

4.3.2.11 UpdateLogindaten

- Verwendete Route: */MeineDaten/ Logindaten (POST)*
- Verwendete Methode: *OZBPersonController::updateLogindaten*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/ozb_person/editLogindaten.html.erb*
 - */views/ozb_person/_navigation.html.erb (Partial)*

4.3.3 Daten anderer Benutzer verwalten

Bei der Anmeldung wird der Benutzer auf die Sonderberechtigungen überprüft. Ist der aktuell angemeldete Benutzer in der Mitgliederverwaltung Gruppe, so stehen ihm folgende Aktionen mit jeweils eigener view zur Verfügung:

- *Alle Mitglieder anzeigen*
- *Details einer ausgewählten Person anzeigen*
- *Ein neues Mitglied hinzufügen*
- *Ein Mitglied bearbeiten*
 - *Personaldaten bearbeiten*
 - *Kontaktdaten bearbeiten*
 - *Rolle bearbeiten (beinhaltet unter anderem auch die "Rolle eines Mitglieds zum Gesellschafter ändern")*
- *Ein Mitglied löschen*

Ist der aktuell angemeldete Benutzer in der Administratoren Gruppe, kann er unter anderem auch

- *einem Mitglied Sonderberechtigungen hinzufügen und löschen*

4.3.4 Routen für die Mitgliederverwaltung

- /Verwaltung/Mitglieder => verwaltung#listOZBPersonen
- /Verwaltung/OZBPerson/:Mnr/Details => verwaltung#detailsOZBPerson
- /Verwaltung/OZBPerson/NeuePerson => verwaltung#newOZBPerson
- /Verwaltung/OZBPerson/NeuePerson => verwaltung#createOZBPerson
- /Verwaltung/OZBPerson/:Mnr/Personaldaten => verwaltung#editPersonaldaten
- /Verwaltung/OZBPerson/:Mnr/Personaldaten => verwaltung#updatePersonaldaten
- /Verwaltung/OZBPerson/:Mnr/Kontaktdaten => verwaltung#editKontaktdaten
- /Verwaltung/OZBPerson/:Mnr/Kontaktdaten => verwaltung#updateKontaktdaten
- /Verwaltung/OZBPerson/:Mnr/Rollen => verwaltung#editRolle
- /Verwaltung/OZBPerson/:Mnr/Rollen => verwaltung#updateRolle
- /Verwaltung/OZBPerson/:Mnr/Loeschen => verwaltung#deleteOZBPerson
- /Verwaltung/OZBPerson/:Mnr/Sonderberechtigungen
 - => verwaltung#editBerechtigungen
- /Verwaltung/OZBPerson/:Mnr/Sonderberechtigungen
 - => verwaltung#createBerechtigung
- /Verwaltung/OZBPerson/:Mnr/Sonderberechtigung/:id/Loeschen
 - => verwaltung#deleteBerechtigung

4.3.5 Verwaltung Controller

Der VerwaltungController beinhaltet die nachfolgenden Methoden:

- Die *listOZBPersonen* Methode liefert eine Übersicht aller OZBPersonen.
- Die *detailsOZBPerson* Methode liefert eine Übersicht aller OZBPersonen.
- Die *newOZBPerson* Methode zeigt das Eingabeformular für die Erstellung einer neuer OZBPerson an.
- Die *createOZBPerson* Methode speichert bei gültigen Formulareingaben eine neue OZBPerson in der Datenbank ab und leitet den Benutzer auf die Übersichtsseite aller Mitglieder weiter. Dort wird zusätzlich eine Meldung angezeigt. Bei einer ungültigen Eingabe wird der Benutzer auf die fehlerhafte Eingaben verwiesen.
- Die *editPersonaldaten*, *editKontaktdaten*, *editRolle* und *editBerechtigungen* Methoden liefern jeweils die ausgewählten Datensätze aus der Datenbank, die im entsprechenden Formularen zur Bearbeitung der Daten vorausgefüllt werden.

- Die *updatePersonaldaten*, *updateKontaktdaten*, *updateRolle* Methoden prüfen jeweils die im Formular eingegebenen Daten auf Gültigkeit und aktualisieren die entsprechenden Datensätze in der Datenbank. Bei einer ungültigen Eingabe wird der Benutzer auf die fehlerhafte Eingaben verwiesen. Bei erfolgreicher Speicherung wird der Benutzer auf die Übersichtsseite weitergeleitet und erhält eine Meldung über die Änderung der Daten.
- Die *createBerechtigung* Methode speichert bei gültigen Formulareingaben eine neue Sonderberechtigung für die Person in der Datenbank ab und leitet den Benutzer auf die Übersichtsseite aller Berechtigungen der Person weiter. Dort wird zusätzlich eine Meldung angezeigt. Bei einer ungültigen Eingabe wird der Benutzer auf die fehlerhafte Eingaben verwiesen.
- Die *deleteBerechtigung* Methode löscht die ausgewählte Sonderberechtigung der Person aus der Datenbank und leitet den Benutzer zur Übersichtsseite aller Berechtigungen der Person weiter.
- Die *deleteOZBPerson* Methode löscht die ausgewählte OZBPerson aus der Datenbank und leitet den Benutzer zur Auflistung aller Mitglieder weiter.

4.3.5.1 *ListOZBPersonen*

- Verwendete Route: */Verwaltung/Mitglieder*
- Verwendete Methode: *VerwaltungController::detailsOZBPerson*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/verwaltung/listOZBPersonen.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Da jeder Bearbeiter unabhängig von der Verwaltungsgruppe die Möglichkeit haben sollte, sich alle Mitglieder sowie Details einer bestimmten Person anzeigen zu lassen, wurde die *listOZBPersonen.html.erb* als Startseite der Verwaltung festgelegt. Hier werden alle Mitglieder aufgelistet. Außerdem, wenn der angemeldete Benutzer sich in der Administratoren- oder in der Mitgliederverwaltungsgruppe befindet, kann er von hier aus ein neues Mitglied aufnehmen. Bei der Mitgliederauflistung wird zur besseren Übersicht die Pagination mit einer

durch den Controller einstellbaren Anzahl der Mitglieder pro Seite verwendet. Die aktuelle Seite der Auflistung bietet die Möglichkeit Mitglieder nach Mitgliedsnummer, Namen, Rolle und Aufnahme datum auf- oder absteigend zu sortieren. Klickt man ein vorhandenes Mitglied an, so wird man zur Detailansicht weitergeleitet.

Verwaltung: Mitglieder

Alle Mitglieder anzeigen
Schnellsuche
Neues Mitglied hinzufügen

◆ Mitglied-Nr.	◆ Name, Vorname	◆ Rolle	◆ Aufnahmedatum
5	Seidel, Wolfgang	Partner	2005-01-21
6	Juhas, Anton	Gesellschafter	2005-01-21
7	Ochs, Armin	Gesellschafter	2005-01-21
8	Peckmann, Ulrich	Gesellschafter	2005-01-21
9	Sander, Uwe	Gesellschafter	2005-01-21
10	Schmitz, Christoph M.	Gesellschafter	2005-01-21
11	Schmitz, Monika	Mitglied	2005-01-21
12	Kienle, Hannelore	Mitglied	2005-01-21
13	Kienle, Tassilo	Gesellschafter	2005-01-21
14	Hamann, Rolf	Mitglied	2005-01-21
15	Meguid-Haag, Käte	Mitglied	2005-01-21
16	Juhas, Inge	Partner	2005-01-21
17	Starke, Klaus	Gesellschafter	2006-03-16
18	Hamann, Peter	Mitglied	2005-01-21
19	Fischer, Günter	Gesellschafter	2005-01-21

«
1
2
3
4
5
Page Gap
10
»

Zurück
Neues Mitglied hinzufügen

Abbildung 23: Verwaltung - Alle Mitglieder anzeigen

Wurde eine neue Person erfolgreich in der Datenbank gespeichert, so wird der Benutzer zur Auflistung aller Mitglieder weitergeleitet, wo eine Erfolgsmeldung zu sehen ist.

Verwaltung: Mitglieder

Person wurde erfolgreich hinzugefügt.

Alle Mitglieder anzeigen

[Schnellsuche](#)

[Neues Mitglied hinzufügen](#)

↕ Mitglied-Nr.	↕ Name, Vorname	↕ Rolle	↕ Aufnahmedatum
5	Seidel, Wolfgang	Partner	2005-01-21
6	Juhas, Anton	Gesellschafter	2005-01-21

Abbildung 24: Verwaltung - Neue Person hinzufügen - Erfolgsmeldung

4.3.5.2 DetailsOZBPerson

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Details (GET)*
- Verwendete Methode: *VerwaltungController::detailsOZBPerson*
- Verwendete Models:
 - *OZBPerson, Person, Adresse, Telefon, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - */views/verwaltung/detailsOZBPerson.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zum Kapitel 4.3.2.3, doch die Hilfsnavigation bietet zusätzliche Möglichkeit. Je nach dem wie der aktuelle Bearbeiter berechtigt ist, werden ihm zusätzliche Menüpunkte angezeigt: Personaldaten, Kontaktdaten, Rolle, Sonderrechte, Konten, Bürgschaften usw. Darf der Bearbeiter die Mitglieder löschen, so sieht er unten rechts die Schaltfläche „Mitglied sicher löschen“.

Mustermann, Max

Details

Personaldaten

Kontaktdaten

Rolle

Sonderrechte

Konten

Personaldaten

Mitgliedsnummer:

903

Name:

Mustermann, Max

Geburtsdatum:

1983-01-01

Vermerk:

Max Mustermann ist eine Testperson

Antragsdatum:

2012-09-01

Aufnahmedatum:

2012-09-02

Austrittsdatum:

2012-09-26

Rolle

Rolle:

Student

Ausbildungsbez.:

Informatik

Institut:

Hochschule Karlsruhe

Studienort:

Karlsruhe

Studienbeginn:

2010-03-01

Studienende:

Abschluss:

Bachelor

Kontaktdaten

Email:

max.mustermann@ozb_test.de

Tel.:

072112345678

Mobil:

017612345678

Fax:

072112345679

Adresse:

Moltkestr. 30, 76133 Karlsruhe

Zurück

Mitglied sicher löschen

Abbildung 25: Verwaltung - Details einer Person anzeigen

4.3.5.3 NewOZBPerson

- Verwendete Route: `/Verwaltung/OZBPerson/NeuePerson`
- Verwendete Methode: `VerwaltungController::newOZBPerson`
- Verwendete Models:
 - *OZBPerson, Person, Adresse, Telefon, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - `/views/verwaltung/newOZBPerson.html.erb`

Die `newOZBPerson.html.erb` dient zur Erstellung einer neuen Person. Die Hilfsnavigation ist hier statisch, sodass man sich bei der Ausfüllung der Form (nicht so wie bei der Bearbeitung)

zwischen den Personaldaten, Kontaktdaten, Rolle und Logindaten frei navigieren kann, ohne die Form neuzuladen.

Neue Person hinzufügen

Personaldaten

Kontaktdaten

Rolle

Logindaten

Personaldaten

*Name:

*Vorname:

Geburtsdatum:

Vermerk:

*Antragsdatum:

Aufnahmedatum:

Austrittsdatum:

* Pflichtfelder

Zurück

Speichern

Abbildung 26: Verwaltung - Neues Mitglied hinzufügen

4.3.5.4 CreateOZBPerson

- Verwendete Route: */Verwaltung/OZBPerson/NeuePerson (POST)*
- Verwendete Methode: *VerwaltungController::createOZBPerson*
- Verwendete Models:
 - *OZBPerson, Person, Adresse, Telefon, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - */views/verwaltung/newOZBPerson.html.erb*

Hat man bei der Ausfüllung der Form (*newOZBPerson.html.erb*) nicht alle Pflichtfelder ausgefüllt, oder waren die Angaben ungültig, bekommt man eine Fehlermeldung, weswegen die Person nicht angelegt werden konnte.

Neue Person hinzufügen

Personaldaten

Kontaktdaten

Rolle

Logindaten

Konnte keine Person anlegen:

- Name muss ausgefüllt werden
- Vorname muss ausgefüllt werden
- Email muss ausgefüllt werden
- Passwort muss ausgefüllt werden
- Antragsdatum muss ausgefüllt werden

Personaldaten

*Name:

*Vorname:

Geburtsdatum:

Vermerk:

*Antragsdatum:

Aufnahmedatum:

Austrittsdatum:

* Pflichtfelder

Zurück

Speichern

Abbildung 27: Verwaltung - Neue Person hinzufügen - Fehlermeldung

4.3.5.5 EditPersonaldaten

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Personaldaten (GET)*
- Verwendete Methode: *VerwaltungController::editPersonaldaten*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/verwaltung/editPersonaldaten.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zu Abschnitt 4.3.2.4.

4.3.5.6 *UpdatePersonaldaten*

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Personaldata (POST)*
- Verwendete Methode: *VerwaltungController::updatePersonaldata*
- Verwendete Models: *OZBPerson, Person*
- Verwendete View:
 - */views/verwaltung/editPersonaldata.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zu Abschnitt 4.3.2.5.

4.3.5.7 *EditKontaktdaten*

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Kontaktdaten (GET)*
- Verwendete Methode: *VerwaltungController::editKontaktdaten*
- Verwendete Models: *OZBPerson, Person, Adresse, Telefon*
- Verwendete View:
 - */views/verwaltung/editKontaktdaten.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zu Abschnitt 4.3.2.6.

4.3.5.8 *UpdateKontaktdaten*

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/ Kontaktdaten (POST)*
- Verwendete Methode: *VerwaltungController::updateKontaktdaten*
- Verwendete Models: *OZBPerson, Person, Adresse, Telefon*
- Verwendete View:
 - */views/verwaltung/editKontaktdaten.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zu Abschnitt 4.3.2.7.

4.3.5.9 EditRolle

- Verwendete Route: `/Verwaltung/OZBPerson/:Mnr/Rolle (GET)`
- Verwendete Methode: `VerwaltungController::editRolle`
- Verwendete Models:
 - `OZBPerson`, `Person`, `Foerdermitglied`, `Gesellschafter`, `Mitglied`, `Partner`, `Student`
- Verwendete View:
 - `/views/verwaltung/editRolle.html.erb`
 - `/views/verwaltung/_navigation.html.erb (Partial)`

Die Bearbeitung der Rolle der ausgewählten Person durch den Verwalter unterscheidet sich von der Bearbeitung der eigenen Rolle dadurch, dass hier zusätzlich, wenn der Bearbeiter die Berechtigung dazu hat, die Rolle ändern kann, z.B. vom Studenten zum Gesellschafter. Die alte Rolle wird in diesem Fall gelöscht bzw. auf ungültig gesetzt.

Mustermann, Max

[Details](#) [Personaldaten](#) [Kontaktdaten](#) **Rolle** [Sonderrechte](#) [Konten](#)

Rolle bearbeiten

*Rolle:

Student

Mitglied

Gesellschafter

Partner

Student

Foerdermitglied

Hochschule Karlsruhe

*Ausbildungsbez.:

*Institut:

*Studienort:

Karlsruhe

*Studienbeginn:

2010-03-01

Studienende:

*Abschluss:

Bachelor

* Pflichtfelder

Zurück

Speichern

Abbildung 28: Verwaltung - Rolle bearbeiten

4.3.5.10 UpdateRolle

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/ Rolle (POST)*
- Verwendete Methode: *VerwaltungController::updateRolle*
- Verwendete Models:
 - *OZBPerson, Person, Foerdermitglied, Gesellschafter, Mitglied, Partner, Student*
- Verwendete View:
 - */views/verwaltung/editRolle.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Analog zu Abschnitt 4.3.2.9.

4.3.5.11 EditBerechtigungen

- Verwendete Route: */Verwaltung/OZBPerson/:Mnr/Sonderberechtigungen*
- Verwendete Methode: *VerwaltungController::editBerechtigungen*
- Verwendete Models: *OZBPerson, Person, Sonderberechtigung*
- Verwendete View:
 - */views/verwaltung/editBerechtigungen.html.erb*
 - */views/verwaltung/_navigation.html.erb (Partial)*

Die *editBerechtigungen.html.erb* listet alle Sonderberechtigungen der ausgewählten Person. Die aufgelisteten Berechtigungen können gelöscht werden. Außerdem kann eine neue Sonderberechtigung erstellt werden.

Mustermann, Max

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Konten](#)

Diese Person hat folgende Sonderberechtigungen:

Berechtigung	Email	Aktionen
Administrator (IT)	max.mustermann.admin@ozb_test.de	Löschen
Mitgliederverwaltung (MV)	max.mustermann.admin.mv@ozb_test.de	Löschen

Neue Berechtigung hinzufügen

*Berechtigung:

Bitte auswählen ▼

*Email:

* Pflichtfelder

[Zurück](#)

[Speichern](#)

Abbildung 29: Verwaltung - Sonderberechtigungen bearbeiten

4.3.5.12 CreateBerechtigung

- Verwendete Route: `/Verwaltung/OZBPerson/:Mnr/ Sonderberechtigungen`
- Verwendete Methode: `VerwaltungController::createBerechtigung`
- Verwendete Models: `OZBPerson, Person, Sonderberechtigung`
- Verwendete View:
 - `/views/verwaltung/editBerechtigungen.html.erb`
 - `/views/verwaltung/_navigation.html.erb (Partial)`

Bei der Erstellung einer neuen Berechtigung für die ausgewählte Person werden die Eingaben auf die Gültigkeit überprüft. Waren die Eingaben fehlerhaft wird die Fehlermeldung angezeigt, weswegen keine Sonderberechtigung hinzugefügt werden konnte.

Mustermann, Max

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Konten](#)

Konnte keine Sonderberechtigung hinzufügen/löschen:

- Berechtigung muss ausgefüllt werden
- Email muss ausgefüllt werden

Neue Berechtigung hinzufügen

*Berechtigung:

Bitte auswählen

*Email:

* Pflichtfelder

Abbildung 30: Verwaltung - Sonderberechtigung - Fehlermeldung

Waren die Eingaben richtig, so wird eine neue Berechtigung erstellt. Man gelangt auf die Übersichtsseite mit einer Erfolgsmeldung, wo er die Berechtigungen anschauen, löschen sowie neue erstellen kann.

Mustermann, Max

Berechtigung wurde erfolgreich hinzugefügt. ×

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Konten](#)

Diese Person hat folgende Sonderberechtigungen:

Berechtigung	Email	Aktionen
Administrator (IT)	max.mustermann.admin@ozb_test.de	Löschen
Mitgliederverwaltung (MV)	max.mustermann.admin.mv@ozb_test.de	Löschen
Finanzenverwaltung (RW)	max.mustermann.admin.rw@ozb_test.de	Löschen

Neue Berechtigung hinzufügen

*Berechtigung:

*Email:

* Pflichtfelder

[Zurück](#)

[Speichern](#)

Abbildung 31: Verwaltung - Sonderberechtigung hinzufügen - Erfolgsmeldung

4.3.5.13 DeleteBerechtigung

- Verwendete Route: `/Verwaltung/OZBPerson/:Mnr/ Sonderberechtigungen`
- Verwendete Methode: `VerwaltungController::createBerechtigung`
- Verwendete Models: `OZBPerson, Person, Sonderberechtigung`
- Verwendete View:
 - `/views/verwaltung/editBerechtigungen.html.erb`
 - `/views/verwaltung/_navigation.html.erb (Partial)`

Wurde die ausgewählte Berechtigung erfolgreich gelöscht, gelangt man auf die Übersichtsseite mit einer Erfolgsmeldung, wo er die Berechtigungen anschauen, löschen sowie neue erstellen kann.

Mustermann, Max

Berechtigung wurde erfolgreich gelöscht.

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Konten](#)

Diese Person hat folgende Sonderberechtigungen:

Berechtigung	Email	Aktionen
Administrator (IT)	max.mustermann.admin@ozb_test.de	Löschen
Finanzenverwaltung (RW)	max.mustermann.admin.rw@ozb_test.de	Löschen

Neue Berechtigung hinzufügen

*Berechtigung:

*Email:

* Pflichtfelder

[Zurück](#)

[Speichern](#)

Abbildung 32: Verwaltung - Sonderberechtigung löschen - Erfolgsmeldung

Hat die ausgewählte Person keine Sonderberechtigungen mehr, gibt es die Möglichkeit nur eine neue Berechtigung zu erstellen.

Mustermann, Max

Berechtigung wurde erfolgreich gelöscht.

[Details](#) [Personaldaten](#) [Kontaktdaten](#) [Rolle](#) [Sonderrechte](#) [Konten](#)

Neue Berechtigung hinzufügen

*Berechtigung:

*Email:

* Pflichtfelder

[Zurück](#)

[Speichern](#)

Abbildung 33: Verwaltung - Sonderberechtigung hinzufügen

5 Beschreibung des CSV-Import für Buchungen (PH)

Für die Entwicklung eines CSV-Imports musste zunächst eine Funktion entwickelt werden, die das Einlesen einer Datei über den Upload eines Webformulars ermöglicht und die darin enthaltenen Daten so aufbereitet, dass nur die relevanten Daten - in diesem Fall nur die tatsächlich benötigten Spalten - eingelesen werden. Aber auch die Validierung bei fehlenden Spalten musste implementiert werden. So entstand die nachfolgende Klasse, die hier erläutert wird.

Die Klasse selbst findet sich unter */lib/CSVImporter.rb*.

5.1 Routen

- */webimport => webimport#index*
- */webimport/csvimport_buchungen => webimport#csvimport_buchungen*

5.2 Struktur der CSV-Datei

Die Datei enthält die Spaltennamen in der ersten Zeile. Spalten sind mit Semikolon voneinander getrennt. Keine Verwendung von Whitespaces und Tabulatoren. Ab der zweiten Zeile finden sich die Daten, die importiert werden müssen.

Anhand der Spaltenköpfe wird die Zuordnung der zu importierenden Daten festgehalten. Als Standardformat werden die Spalten Belegdatum, Buchungsdatum, Belegnummernkreis, Belegnummer, Buchungstext, Buchungsbetrag, Euro, Sollkonto und Habenkonto (in dieser Reihenfolge) erwartet.

Als Beispiel sind hier zwei Zeilen von Beispieldaten gegeben.

```
Belegdatum;Buchungsdatum;Belegnummernkreis;Belegnummer;Buchungstext;Buchungsbetrag Euro;Sollkonto;Habenkonto
30.07.2012;31.07.2012;B-;471;0097 Max Mustermann Einlage;69,44;1200;70095
31.07.2012;31.07.2012;B-;472;0006 Meier Gerlinde Einlage D082647;500,00;1200;70412
```

Abbildung 34: Beispiel der Formatierung einer CSV-Datei

5.3 Web-Import

Der Web-Import umfasst den *WebimportController* und die Klasse *CSVImporter*. Der Controller ist mit der Methode *index* zuständig für das Anzeigen des Formulars zur Auswahl einer Importdatei und mit der Methode *csvimport_buchungen* für den Import der Buchungen in die Datenbank.

Der Web-Import ist über das Navigationsmenü innerhalb der Seite über „Verwaltung“ -> WebImport zu erreichen. Dort erscheint folgendes minimalistisches - aber völlig ausreichendes - Formular:

WebImport > Buchungen

1. Wählen Sie zunächst eine CSV-Datei auf Ihrem Computer aus.
2. Klicken Sie auf den Button "importieren", um den Importvorgang zu starten.

CSV-Datei:

Nachdem der Benutzer eine CSV-Datei ausgewählt hat und auf den Button „importieren“ geklickt hat, wird diese über das Formular an die Methode *csvimport_buchungen* gesendet. Dort wird zunächst überprüft, ob überhaupt eine Datei hochgeladen wurde. Ist dies nicht der Fall, so wird dem Benutzer eine entsprechende Meldung angezeigt. Hat der Benutzer eine Datei hochgeladen, so wird diese zunächst auf dem Webserver gespeichert und über die Klasse *CSVImporter* versucht, die angegebenen Spalten zu importieren. Liefert der *Importer* einen oder mehrere Fehler, so wird der Vorgang abgebrochen und der Benutzer wird aufgefordert eine gültige Datei anzugeben. Die temporäre Datei auf dem Webserver wird gelöscht. Hat der Benutzer hingegen eine gültige Datei hochgeladen, so werden für jeden Datensatz die Buchungen, nach der Vorgabe der o/ZB Stuttgart und Vorlage aus dem bisherigen System, importiert. Die Implementierung des CSV-Import lag als PHP-Code vor und wurde in Ruby äquivalent umgesetzt. Der Import berechnet aus den übergebenen Buchungen den *wSaldo* (Währungssaldo), wie auch den *pSaldo* (Punktesaldo) für alle betroffenen Buchungen, die importiert werden und speichert alle Buchungen dem jeweiligen Konto zu.

Nachfolgend eine detaillierte Beschreibung für die Importfunktion.

Das Formular selbst erhält als Array-Index den Eintrag „webimport“, die Datei kann mit dem Namen „file“ angesprochen werden. Hier wird zunächst geprüft, ob überhaupt eine Datei an-

gegeben wurde. Ist keine Datei angegeben, so wird hier bereits der *else*-Zweig ausgeführt und mit einer Fehlermeldung beendet.

```
if !params[:webimport].nil? && !params[:webimport][:file].nil?
```

Ist eine Datei hochgeladen worden, so wird sie von ihrem temporären Speicherplatz in das Verzeichnis */public/uploads/* kopiert und eine neue Instanz des *CSVImporter* erzeugt, der als Parameter den Pfad (einschließlich Datei), sowie alle zu importierenden Spalten entgegennimmt. Ist der Import durch den *CSVImporter* beendet, wird die Datei von der Festplatte gelöscht (ab hier an sind alle Daten im Speicher der Anwendung).

```
uploaded_io = params[:webimport][:file]

uploaded_disk = Rails.root.join('public', 'uploads', uploaded_io.original_filename)
File.open(uploaded_disk, 'w') do |file|
  file.write(uploaded_io.read)
end

# import CSV-File
csv = CSVImporter.new
csv.import_from_file(uploaded_disk, ["Belegdatum", "Buchungsdatum", "Belegnummernkreis", "Belegnummer", "Buchungstext", "Buchungsbetrag Euro", "Sollkonto", "Habenkonto"])

# <-> compared to old import.php:
# $buchungsdatum, $wertstellungsdatum, $belegnummernkreis, $belegnummer,
# $buchungstext, $betrag, $sollkontonummer, $habenkontonummer

# delete CSV-File
require 'FileUtils'
FileUtils.rm(uploaded_disk)
```

Anschließend folgt der eigentliche, Zeilenweise, Import der Datensätze. Dazu werden die Zeilen nacheinander durchlaufen und in einer Transaktion in die Datenbank eingefügt.

Da der Quellcode sehr umfangreich ist, sei an dieser Stelle für weitere und detaillierte Informationen auf den Quelltext verwiesen. Er ist nicht sehr komplex, aber sehr lange und wäre unleserlich hier abzubilden.

5.4 Klasse CSVImporter

Damit der Import aus einer Datei zunächst überhaupt möglich ist, sind einige Zeilen Code erforderlich, die die Daten aus der Datei in die Anwendung einlesen.

Die Klasse verwendet *iconv*, um das Transformieren von unterschiedlichen *Encodings* zu ermöglichen. Das ist wichtig für die Eingabedateien, um bspw. Umlaute korrekt einzulesen. Aber die Klasse verwendet auch *csv*, um das Rad nicht neu zu erfinden. Die nachfolgenden Zeilen sind hierfür nötig.

```
require 'iconv'
require 'csv'
```

Damit der Zugriff auf die Attribute der Klasse möglich sind, müssen diese auch so definiert werden und im Konstruktor initialisiert werden.

```
attr_accessor :processed, :number_records, :error, :notice, :cols, :rows

# constructor
def initialize
  @processed = false
  @number_records = 0
  @error = ""
  @notice = ""
  @cols = []
  @rows = []
end
```

Die Methode *import_from_file* akzeptiert zwei Parameter. Der erste ist ein String und beinhaltet den absoluten Pfad (oder relativen Pfad vom *root* Verzeichnis) zur zu importierenden Datei. Der zweite Parameter ist eine Liste (Array) von den Namen der zu importierenden Spalten. Damit können auch nur einzelne Spalten aus der Datei importiert werden, ohne vorher manuell die Spalten aus der Datei entfernen zu müssen.

```
def import_from_file(file, cols_to_import)
  # Vordefinierte Spalten. Reihenfolge muss mit Quelldatei übereinstimmen!
  (case insensitive)
  def_cols = cols_to_import
  def_map = {} # Hashmap für Zuweisung von Spaltenname -> Index in row
```

Das *Encoding* der Eingabedatei muss zu dem der Anwendung passen. In diesem Fall wird dies auf *LATIN1* voreingestellt und konvertiert die Daten nach *UTF-8*. Das heißt auch, dass die Eingabedatei eine *LATIN1* codierte Datei sein muss. Ist dies nicht der Fall, so wird unter Umständen überhaupt nichts importiert, da auch die Zeilenumbrüche nicht erkannt werden können.

```
i = Iconv.new('UTF-8', 'LATIN1')

utf8_encoded_file = ""
file = File.new(file, "r")
while (line = file.gets)
  utf8_encoded_file += i.iconv(line)
end
file.close
```

Nach dieser Konvertierung beginnt das Zeilenweise Einlesen in das Array `@rows`. Hierzu wird ein neues Array erzeugt und in einer einzigen Transaktion versucht, alle Spalten einzulesen.

```
@rows = Array.new
n = -1
begin
  # try
  CSV::Reader.parse(utf8_encoded_file, ';') do |row|
    n += 1
    if n == 0
```

Der Zeilencounter (`n`) gibt an, an welcher Zeile gerade gelesen wird. Dabei werden auch die angegebenen Kopfdaten (Spaltennamen) der CSV-Datei berücksichtigt. Denn nur hier kann erkannt werden, um welche Spalte es sich handelt. Dazu wird im ersten Durchlauf ($n = 1$) die Zeile mit den Spaltennamen eingelesen und alle Eingabezeichen in Kleinbuchstaben umgewandelt, damit an dieser Stelle mit den Namen, der tatsächlich zu importierenden Spalten verglichen werden kann.

```
# Erster Durchlauf = Spaltennamen
col_headers = row

# Case insensitive
col_headers.map!{ |i|
  i.downcase if !i.nil?
}.uniq
```

Anschließend wird überprüft, ob alle im Parameter *cols_to_import* angegebenen Spalten, auch tatsächlich in der CSV-Datei vorhanden sind. Dazu wird ein Array verwendet, das auch die Position der Spalte speichert, an dem die Spalte in der Datei vorkommt und beim Fehlen einer Spalte, den Namen der Spalte in ein zweites Array speichert.

```
# Überprüfe, ob alle Spalten vorhanden sind
not_included = Array.new
col_pos = 0
def_cols.each do |dc|
  if (!col_headers.include?(dc))
    # enthält die Spalte dc nicht
    not_included.push(dc)
  else
    # enthält die Spalte dc an Position col_pos
    def_map[dc] = col_pos
  end

  col_pos += 1
end
```

Sind einige Spalten (jedoch mindestens eine) nicht in der Datei enthalten, wird bereits nach Durchlauf aller Spalten, an dieser Stelle abgebrochen und ein Fehler erzeugt. Das Überprüfen, ob ein Fehler vorlag, kann dann mittels Abfragen auf das Attribut *@processed* erfolgen. Liefert es *true*, so wurden alle Zeilen durchlaufen. Falls es *false* ist, so kann aus dem Attribut *@error* der Fehler ausgelesen werden.

```
if (not_included.size > 0)
  @error = "Folgende Spalten sind nicht vorhanden: " +
not_included.join(", ")
  break
end
```

Kam es zu keinem Fehler, so wird die Zuweisung zwischen der Spalte in der Datei und der Position der Spalte, wie sie eingelesen werden soll, nach Index sortiert.

```
# Sortiere Hashmap
@cols = def_map.sort_by{ |key, value| value }.to_a
```

Ab dem zweiten Durchlauf findet der eigentliche Import aus der Datei statt, hier werden die Daten Zeilenweise in das Array `@rows` gepusht.

```
else
  # Zweiter..n-ter Durchlauf = Daten
  a = Array.new
  def_cols.each do |dc|
    a << row[def_map[dc]]
  end

  @rows.push(a)
end
```

Kommt es zu Problemen beim Einlesen der Datei, so wirft der `CSV::Reader` eine `CSV::IllegalFormatError Exception`, die durch einen `rescue` (= catch Block in Ruby) Anweisung abgefangen wird.

```
rescue CSV::IllegalFormatError => e
  # catch
  @error = "Ungültige Datei (#{e.class})"
```

Beim Beenden aller Durchläufe wird der folgende Codeabschnitt durchlaufen. Allerdings nur, wenn alle Zeilen abgelaufen wurden.

```
else
  # final
  if n == @rows.size
    # done -> wenn alle Zeilen abgelaufen wurden
    @processed = true
    @number_records = n
    @notice = "Folgende Spalten wurden aus der Datei verwendet: " +
def_cols.join(", ")
  end
end
```

Die Variable `@processed` zeigt an, dass die Datei vollständig abgearbeitet wurde, mit der Anzahl von `@number_records` Datensätzen und der Rückbestätigung an den tatsächlich importierten Spalten.

6 Layout / Twitter Bootstrap (AL)

6.1 Bootstrap (Framework)⁵

„Bootstrap ist eine freie Sammlung von Hilfsmitteln für die Gestaltung von Websites und Webanwendungen. Es enthält auf HTML und CSS basierende Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-System, Navigations- und andere Oberflächengestaltungselemente sowie zusätzliche, optionale JavaScript-Erweiterungen. Es ist das populärste Projekt beim Open-Source-Hostingdienst Github und wird unter anderem von der NASA und MSNBC eingesetzt.“

6.2 Basisdaten⁵

Entwickler	Twitter
Erscheinungsjahr	2011
Aktuelle Version	2.1.0 (21. August 2012)
Betriebssystem	plattformunabhängig
Programmiersprache	CSS (LESS), JavaScript
Kategorie	Webdesign
Lizenz	Apache License 2.0 (Code) CC BY 3.0 (Dokumentation, Icons)
Deutschsprachig	Nein
Homepage	http://twitter.github.com/bootstrap/

Tabelle 6.2: Twitter Bootstrap Basisdaten

6.3 Grundgerüst: Grid-System und Responsive Design⁵

„Bootstrap wird standardmäßig mit einem 940 Pixel breiten, zwölfspaltigen Grid-Layout ausgeliefert. Alternativ kann der Entwickler auch ein Layout mit variabler Breite verwenden. Für beide Fälle bietet das Toolkit vier Variationen im Sinne des Responsive Designs an, welche verschiedene Auflösungen und Gerätetypen bedienen: Mobiltelefone, hoch- und querformatige Tablets, sowie PCs mit geringer und hoher (Widescreen-)Auflösung. Dabei passt sich die Breite der Spalten automatisch der zur Verfügung stehenden Fensterbreite an.“

⁵ [http://de.wikipedia.org/wiki/Bootstrap_\(Framework\)](http://de.wikipedia.org/wiki/Bootstrap_(Framework))

Über das Konfigurationsstylesheet hat der Entwickler die Möglichkeit die Anzahl und Breite der Spalten, den Abstand zwischen den Spalten sowie die Gesamtbreite des Layouts seinen Vorstellungen anzupassen.“

6.4 Grundlegendes CSS-Stylesheet⁴

„Bootstrap enthält eine Reihe von Stylesheets, welche grundlegende Stildefinitionen für alle wichtigen HTML-Komponenten enthalten. Diese gewährleisten ein browser- und systemübergreifend einheitliches, modernes Erscheinungsbild für die Textformatierung, Tabellen und Formularelemente. Der Entwickler profitiert dabei von den Erfahrungen, die bei der Entwicklung und Gestaltung von Twitter gemacht wurden und kann auf praxiserprobte Gestaltungsentscheidungen und bewährte Entwurfsmuster der Frontendgestaltung zurückgreifen.“

6.5 Wiederverwendbare Komponenten⁵

„Ergänzend zu den regulären HTML-Elementen enthält Bootstrap weitere, häufig verwendete Oberflächenelemente. Hierzu gehören unter anderem Buttons mit erweiterter Funktionalität (bspw. Gruppierung von Buttons oder Buttons mit Dropdown-Möglichkeit), Navigationselemente (Navigationslisten und -leisten, horizontale und vertikale Reiter, Brotkrümelnavigation, Paginierung, usw.), Labels, erweiterte typografische Möglichkeiten, Miniaturansichten, Formatierungen für Hinweismeldungen und Fortschrittsbalken.“

6.6 JavaScript-Plugins⁵

„Die JavaScript-Komponenten von Bootstrap basieren auf dem JavaScript-Framework jQuery. Die im Toolkit enthaltenen Plugins sind dementsprechend jQuery-Plugins. Sie bieten zusätzliche User-Interface-Elemente, wie beispielsweise Dialogfenster, Tooltips und Karussells. Außerdem erweitern sie die Funktionalität einiger vorhandener Oberflächenelemente, darunter zum Beispiel eine Auto-Vervollständigen-Funktion für Eingabefelder.“

6.7 Ruby on Rails und Twitter Bootstrap

6.7.1 Einbindung der Twitter Bootstrap Bibliotheken (v2.1.0)

Um das Twitter Bootstrap Framework in Ruby on Rails Anwendungen nutzen zu können, benötigen wir die JavaScript- sowie CSS-Dateien des Frameworks in entsprechende Ordner der Anwendung zu kopieren:

- *OZB/app/assets/img/*
 - *glyphicons-halflings.png*
 - *glyphicons-halflings-white.png*
- *OZB/app/assets/javascript/*
 - *bootstrap.min.js*
 - *jquery-1.7.2.min.js*
- *OZB/app/assets/stylesheets/*
 - *bootstrap.min.css*

6.7.2 Hauptnavigationsleiste

Bei der Anmeldung wird der Benutzer auf die Sonderberechtigungen überprüft. Dafür werden zunächst folgende Methoden im ApplicationController verwendet:

```
helper_method :isCurrentUserAdmin
def isCurrentUserAdmin
  return !(Sonderberechtigung.find(
    :all,
    :conditions => {
      :Mnr => current_OZBPerson.Mnr
    })
  ).first.nil?
end
```

Diese Methode wird als Hilfsmethode deklariert, damit sie sowohl in Controllern als auch in Views aufrufbar wird. Sie ist wichtig, um in der Navigationsleiste der Anwendung das “Verwaltung” Button anzuzeigen, falls der aktuell angemeldete Benutzer sich in einer der Verwaltungsgruppen befindet.

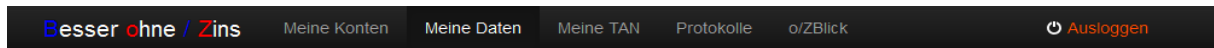


Abbildung 35: Navigationsleiste Benutzer

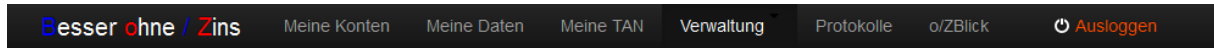


Abbildung 36: Navigationsleiste Administrator

Um ein Navigationselement hervorzuheben, soll dem entsprechenden HTML-Element, in unserem Fall Listenelement, ein Attribut `class="active"` hinzugefügt werden.

Die Funktion im *ApplicationController* `getCurrentLocation` liefert den Pfad, wo sich der Benutzer innerhalb der Anwendung momentan befindet.

```
helper_method :getCurrentLocation
def getCurrentLocation
  return request.path
end
```

Nun können die Pfade verglichen werden und entsprechende Navigationselemente hervorgehoben werden.

Der folgende Code implementiert die Navigationsleiste:

```

<% if OZBPerson_signed_in? %>
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="navbar-inner">
    <div class="container">
      <a class="brand" href="#">
        <font color="white">
          <font color="blue">B</font>esser
          <font color="red">o</font>hne
          <font color="blue">/ </font>
          <font color="red">Z</font>ins
        </font>
      </a>
      <div class="nav-collapse">
        <ul class="nav">
          <li>
            <% if getCurrentLocation == "/MeineKonten" || getCurrentLocation == "/"
            class="active"
            <% end %>>
            <a href="/MeineKonten">Meine Konten</a>
          </li>
          <li>
            <% if getCurrentLocation.include? "/MeineDaten" %>
              class="active"
            <% end %>>
            <a href="/MeineDaten/Details">Meine Daten</a>
          </li>
          <li><a href="#">Meine TAN</a></li>
          <% if isCurrentUserAdmin %>
            <li <% if getCurrentLocation.include? "Verwaltung"
            class="active"<%end%>>
              <a href="/Verwaltung/Mitglieder">Verwaltung <b class="caret"></b></a>
            </li>
          <% end %>
          <li><a href="#">Protokolle</a></li>
          <li><a href="#">o/ZBlick</a></li>
        </ul>
        <ul class="nav pull-right">
          <li>
            <%= link_to raw("<i class='icon-off icon-white'></i> Ausloggen"), de-
            stroy_OZBPerson_session_path, :method => :delete, :style => "color: #ff4500;"%>
          </li>
        </ul>
      </div><!-- /.nav-collapse -->
    </div><!-- /container -->
  </div><!-- /navbar-inner -->
</div><!-- /navbar -->
<% end %>

```

6.7.3 Sub Navigationsleiste (Tabs) (AL/PH)

Ist die Navigation sichtbar und hat ein Benutzer einen Verwaltungsbereich ausgewählt, so wird die Methode *is_allowed* benötigt, um festzustellen, welche Geschäftsvorfälle (siehe Abbildung Abbildung 15: Rechteverwaltung) er verwalten darf.

```
helper_method :is_allowed
def is_allowed(ozb_person, geschaeftsvorfallnr)
  if ozb_person && ozb_person.Mnr
    if prozess = Geschaeftsprozess.where(:ID => geschaeftsvorfallnr).first
      return true if prozess.IT && Sonderberechtigung.where(:Mnr =>
        ozb_person.Mnr).where(:Berechtigung => "IT").first
      return true if prozess.MV && Sonderberechtigung.where(:Mnr =>
        ozb_person.Mnr).where(:Berechtigung => "MV").first
      return true if prozess.RW && Sonderberechtigung.where(:Mnr =>
        ozb_person.Mnr).where(:Berechtigung => "RW").first
      return true if prozess.ZE && Sonderberechtigung.where(:Mnr =>
        ozb_person.Mnr).where(:Berechtigung => "ZE").first
      return true if prozess.OeA && Sonderberechtigung.where(:Mnr =>
        ozb_person.Mnr).where(:Berechtigung => "OeA").first
    else
      #Kein Prozess mit der id gefunden
      puts "kein Prozess"
    end
  else
    #Kein gültige OZBPerson angegeben
    puts "kein Person"
  end
  return false
end
```

Abhängig davon, welche Berechtigungen der aktuell angemeldete Benutzer hat, werden die Menüelemente der Navigationsleiste dynamisch angepasst.

Details Personaldaten Kontaktdaten Rolle Sonderrechte Konten

Abbildung 37: Sub Navigationsleiste (Tabs)

Der folgende Code implementiert die Navigationsleiste.

```
<header class="subhead" id="overview">
  <h1><%= @Person.Name + ", " + @Person.Vorname %></h1>
  <hr />
</header>

<%
  curr_link_route = request.fullpath.split("/")[4]
  link_prefix = "/Verwaltung/OZBPerson/#{@OZBPerson.Mnr.to_s}/"

  links = [
    # access - name of tab - name of route link
    [2, "Details", "Details"],
    [4, "Personaldaten", "Personaldaten"],
    [4, "Kontaktdaten", "Kontaktdaten"],
    [4, "Rolle", "Rollen"],
    [7, "Sonderrechte", "Sonderberechtigungen"],
    [17, "Bürgschaften", "Buergschaften"],
    [11, "OZB-Konten", "Konten"]
  ]
%>

<ul class="nav nav-tabs">
  <% links.each do |acc_level, link_name, link_route| %>
    <% if is_allowed(current_OZBPerson, acc_level) %>
      <li <%= 'class="active"'>.html_safe if curr_link_route ==
link_route %>><%= link_to link_name, link_prefix + link_route %></li>
      <% end %>
    <% end %>
  </ul>
```

6.7.4 will_paginate

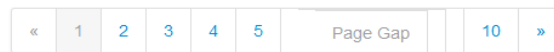


Abbildung 38: Pagination mit will_paginate und Twitter Bootstrap Design

Um die Pagination, wie auf dem Bild gezeigt, mit will_paginate und Twitter Bootstrap Design zu realisieren, muss der ApplicationHelper um den folgenden Code erweitert werden.

```
module ApplicationHelper
  # application_helper.rb
  # https://gist.github.com/1205828
  # Based on https://gist.github.com/1182136
  # Based on https://gist.github.com/1205828, in turn based on
  # https://gist.github.com/1182136
  class BootstrapLinkRenderer < ::WillPaginate::ActionView::LinkRenderer
    protected

    def html_container(html)
      tag :div, tag(:ul, html), container_attributes
    end

    def page_number(page)
      tag :li, link(page, page, :rel => rel_value(page)),
        :class => ('active' if page == current_page)
    end

    def gap
      tag :li, link(super, '#'), :class => 'disabled'
    end

    def previous_or_next_page(page, text, classname)
      tag :li, link(text, page || '#'),
        :class => [classname[0..3], classname, ('disabled' unless page)].join(' ')
    end

    def page_navigation_links(pages)
      will_paginate(pages, :class => 'pagination pagination-centered',
        :inner_window => 2, :outer_window => 0, :renderer => BootstrapLinkRenderer,
        :previous_label => '&laquo;'.html_safe, :next_label => '&raquo;'.html_safe)
    end
  end
end
```

Fragmente aus app/helpers/application_helper.rb zuständig für die will_paginate Optimierung.

Um jetzt z.B. alle Mitglieder mit Pagination aufzulisten benötigen wir in die gewünschte View folgendes hinzuzufügen:

```
<table id="personen" class="table table-bordered table-striped tablesorter">
  <thead>
    <tr>
      <th>Mitglied-Nr.</th>
      <th>Name, Vorname</th>
      <th>Rolle</th>
      <th>Aufnahmedatum</th>
    </tr>
  </thead>
  <tbody>
    <% @OZBPersonen.each do |ozbPerson| %>
      <tr>
        <% person = Person.get(ozbPerson.Mnr) %>
        <td><%= person.Pnr %></td>
        <td>
          <% details_link = '/Verwaltung/OZBPerson/' + ozbPerson.Mnr.to_s +
"/Details"%>
          <% details_link_name = person.Name + ", " + person.Vorname %>
          <%= link_to details_link_name, details_link, :class => "edit_link" %>
        </td>
        <td><%= @Rollen.fetch(person.Rolle) %></td>
        <td><%= ozbPerson.Aufnahmedatum %></td>
      </tr>
    <% end %>
  </tbody>
</table>

<%= page_navigation_links @OZBPersonen %>
```

6.7.5 jQuery Tablesorter

Für die Sortierung der Tabellen werden folgende Bibliotheken benötigt:

- OZB/app/assets/javascript/
 - jquery.tablesorter.min.js
- OZB/app/assets/stylesheets/
 - tablesorter.css
 - asc.gif
 - bg.gif
 - desc.gif

Um jetzt jQuery Tablesorter auf die existierende Tabelle (siehe Code im vorherigen Codeausschnitt) anzuwenden, benötigt man den Table-Tag um zwei Attribute `id="personen"` und `class="tablesorter"` zu erweitern und folgenden Javascript-Code in `listOZBPersonen.html.erb` zu platzieren.

```
<script>
$(document).ready(function() {
  $("#personen").tablesorter();
});
</script>
```

Die folgenden Bilder demonstrieren das jQuery Tablesorter Widget.

[Alle Mitglieder anzeigen](#)
[Schnellsuche](#)
[Neues Mitglied hinzufügen](#)

▲ Mitglied-Nr.	◆ Name, Vorname	◆ Rolle	◆ Aufnahme datum
5	Seidel, Wolfgang	Partner	2005-01-21
6	Juhas, Anton	Gesellschafter	2005-01-21
7	Ochs, Armin	Gesellschafter	2005-01-21
8	Peckmann, Ulrich	Gesellschafter	2005-01-21
9	Sander, Uwe	Gesellschafter	2005-01-21
10	Schmitz, Christoph M.	Gesellschafter	2005-01-21
11	Schmitz, Monika	Mitglied	2005-01-21
12	Kienle, Hannelore	Mitglied	2005-01-21
13	Kienle, Tassilo	Gesellschafter	2005-01-21
14	Hamann, Rolf	Mitglied	2005-01-21
15	Meguid-Haag, Käte	Mitglied	2005-01-21
16	Juhas, Inge	Partner	2005-01-21
17	Starke, Klaus	Gesellschafter	2006-03-16
18	Hamann, Peter	Mitglied	2005-01-21
19	Fischer, Günter	Gesellschafter	2005-01-21

« 1 2 3 4 5 Page Gap 10 »

Abbildung 39: Sortierung der Mitglieder nach Mitgliedsnummer aufsteigend

Alle Mitglieder anzeigen
Schnellsuche
Neues Mitglied hinzufügen

▼ Mitglied-Nr.	◆ Name, Vorname	◆ Rolle	◆ Aufnahmedatum
19	Fischer, Günter	Gesellschafter	2005-01-21
18	Hamann, Peter	Mitglied	2005-01-21
17	Starke, Klaus	Gesellschafter	2006-03-16
16	Juhas, Inge	Partner	2005-01-21
15	Meguid-Haag, Käte	Mitglied	2005-01-21
14	Hamann, Rolf	Mitglied	2005-01-21
13	Kienle, Tassilo	Gesellschafter	2005-01-21
12	Kienle, Hannelore	Mitglied	2005-01-21
11	Schmitz, Monika	Mitglied	2005-01-21
10	Schmitz, Christoph M.	Gesellschafter	2005-01-21
9	Sander, Uwe	Gesellschafter	2005-01-21
8	Peckmann, Ulrich	Gesellschafter	2005-01-21
7	Ochs, Armin	Gesellschafter	2005-01-21
6	Juhas, Anton	Gesellschafter	2005-01-21
5	Seidel, Wolfgang	Partner	2005-01-21

«
1
2
3
4
5
Page Gap
10
»

Abbildung 40: Sortierung der Mitglieder nach Mitgliedsnummer absteigend

6.7.6 Modaler Dialog und jQuery Autovervollständigung

In diesem Kapitel wird ein Modaler Dialog und Autovervollständigung mit Hilfe Twitter Bootstrap Frameworks demonstriert.

Als erstes fügen wir der gewünschten Form einen Link hinzu. Klickt man diesen Link an, soll ein Dialogfenster mit `id="searchPerson"` geöffnet werden. Dafür brauchen wir folgende Codezeile:

```
<a class="" data-toggle="modal" href="#searchPerson" >Suchen...</a>
```

Rolle bearbeiten

*Rolle:

*Partner (Mnr.): [Suchen...](#)

*Berechtigung:

Abbildung 41: Aufruf des modalen Dialogs bei „Suchen“

Als nächstes definieren wir einen Div-Container (Dialogfenster) mit den Attributen `class="modal hide"` und `id="searchPerson"` wie folgt:

```
<div class="modal hide fade" id="searchPerson">
  <div class="modal-header">
    <button class="close" data-dismiss="modal">x</button>
    <h3>Schnellsuche einer Person</h3>
  </div>
  <div class="modal-body">
    <div class="alert alert-info">
      Die Mitglied-Nr. wird automatisch übernommen,
      sobald Sie einen Namen ausgewählt haben.
    </div>
    <label>Name:</label>
    <input type="text" id="suchFeld" class="span4" />
  </div>
  <div class="modal-footer">
    <a href="#" class="btn" data-dismiss="modal">Schliessen</a>
  </div>
</div>
```

Die CSS-Klasse `.modal` besagt, dass dieser Container als ein Modaler Dialog dienen soll. Die Klasse `.hide` versteckt das definierte Dialogfenster.

Klickt man nun auf zuvor definierten „Suchen“ Link, öffnet sich ein Dialogfenster:

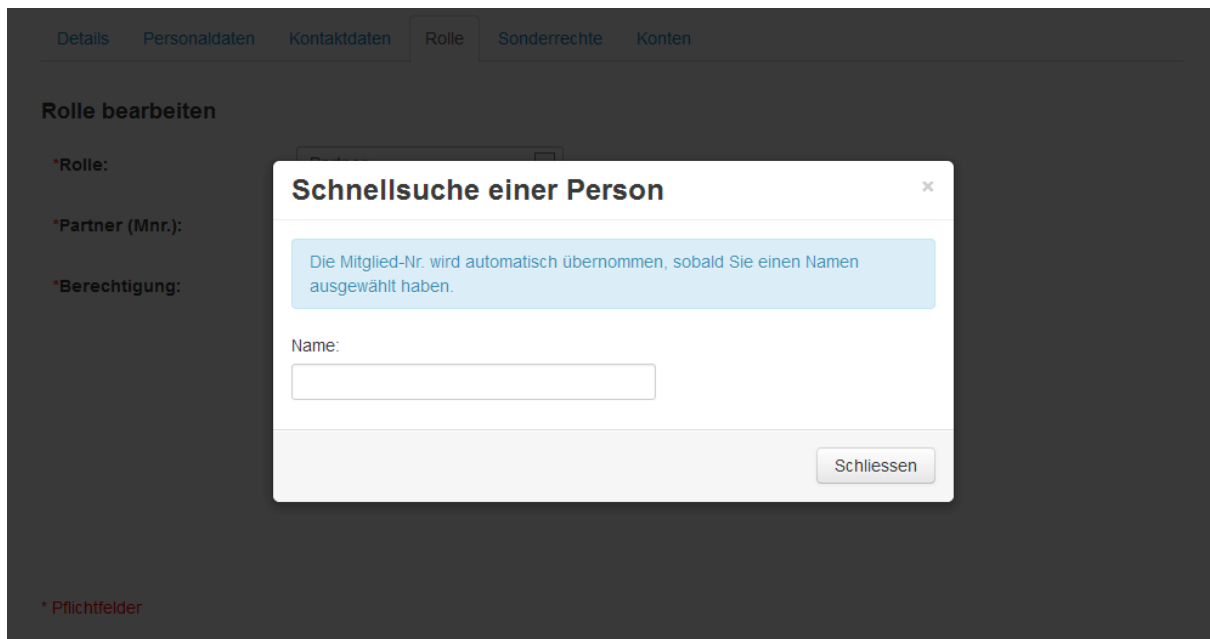


Abbildung 42: Modaler Dialog „Schnellsuche einer Person“

Nun wollen wir das Eingabefeld mit `id="suchFeld"` (Zeile 12, siehe Code im vorherigen Codeausschnitt) um Autovervollständigung erweitern.

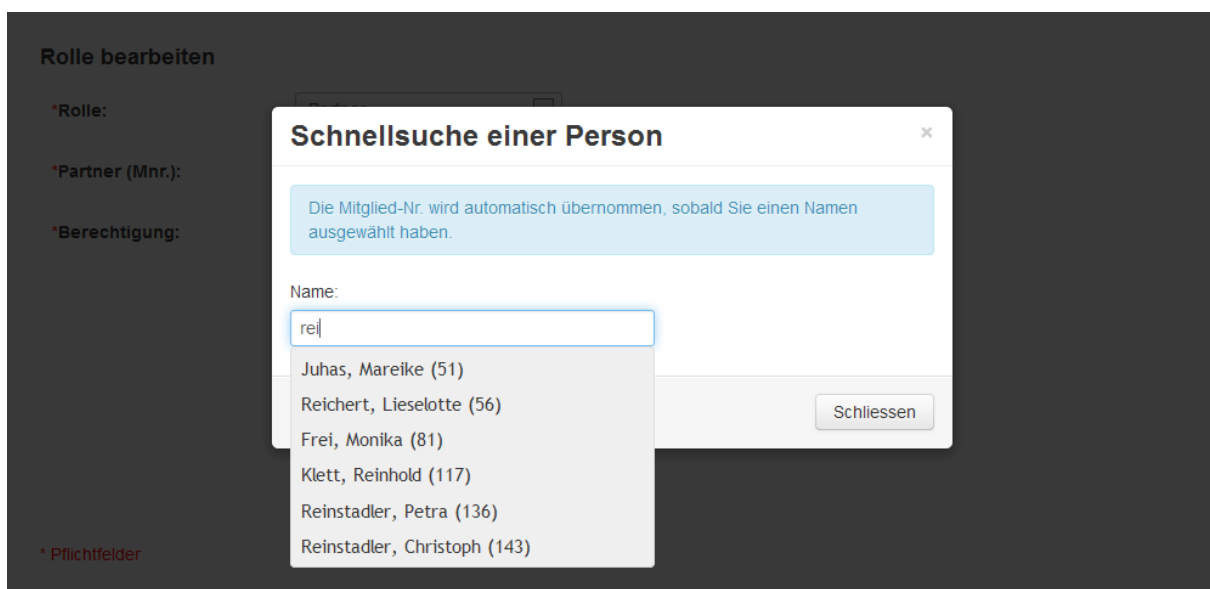


Abbildung 43: jQuery Autovervollständigung

Die jQuery Funktion sieht wie folgt aus:

```
<script>
$(function() {
  var names = [
    <% @DistinctPersonen.each do |person| %>
    { value: "<%= person.Pnr.to_s %>",
      label:
        "<%= person.Name.to_s + ', ' + person.Vorname.to_s + ' (' + person.Pnr.to_s +
        ')' %>"
    },
    <% end %>
  ];
  $( "#suchFeld" ).autocomplete({
    minLength: 0,
    source: names,
    select: function( event, ui ) {
      $( "#suchFeld" ).val( ui.item.label );
      $( "#partner" ).val( ui.item.value );
      $( "#notarPnr" ).val( ui.item.value );
      return false;
    }
  })
  .data( "autocomplete" )._renderItem = function( ul, item ) {
    return $( "<li></li>" )
      .data( "item.autocomplete", item )
      .append( "<a>" + item.label + "</a>" )
      .appendTo( ul );
  };
});
</script>
```

Mit Hilfe der Ruby Variable *@DistinctPersonen* (Datenbankabfrage aller Personen) wird die Javascript Variable *names* zweispaltig mit Personendaten gefüllt. Die erste Spalte „Value“ speichert die Pnr., die wir später brauchen werden, und die zweite „Label“ Namen, Vornamen und Pnr., die für die eigentliche Vervollständigung der betroffenen Personen benötigt werden. Sobald man eine Person aus der Liste ausgewählt hat, wird das Eingabefeld „Partner (Mnr.)“ mit der *id=„partner“* mit der davor erwähnten Pnr. ausgefüllt.

```
<script>
$(function() {
  var names = [
    <% @DistinctPersonen.each do |person| %>
    { value: "<%= person.Pnr.to_s %>",
      label:
        "<%= person.Name.to_s + ', ' + person.Vorname.to_s + ' (' + person.Pnr.to_s +
        ') ' %>"
    },
    <% end %>
  ];
  $( "#suchFeld" ).autocomplete({
    minLength: 0,
    source: names,
    select: function( event, ui ) {
      $( "#suchFeld" ).val( ui.item.label );
      $( "#partner" ).val( ui.item.value );
      $( "#notarPnr" ).val( ui.item.value );
      return false;
    }
  })
  .data( "autocomplete" )._renderItem = function( ul, item ) {
    return $( "<li></li>" )
      .data( "item.autocomplete", item )
      .append( "<a>" + item.label + "</a>" )
      .appendTo( ul );
  };
});
</script>
```

Rolle bearbeiten

*Rolle:	<input type="text" value="Partner"/>	
*Partner (Mnr.):	<input type="text" value="81"/>	Suchen...
*Berechtigung:	<input type="text" value="1"/>	

Abbildung 44: Übernahme der Mitgliedernummer nach Schliessen des modalen Dialogs

6.7.7 jQuery Datepicker

Um Datepicker von jQuery zu verwenden, werden folgende Bibliotheken benötigt:

- OZB/app/assets/javascript/
 - jquery-ui.js
- OZB/app/assets/stylesheets/
 - jquery-ui-1.8.16.custom.css

Personaldaten bearbeiten

Mitgliedsnummer:

*Name:

*Vorname:

Geburtsdatum:

Vermerk:

*Antragsdatum:

Aufnahmedatum:

Austrittsdatum:

* Pflichtfelder

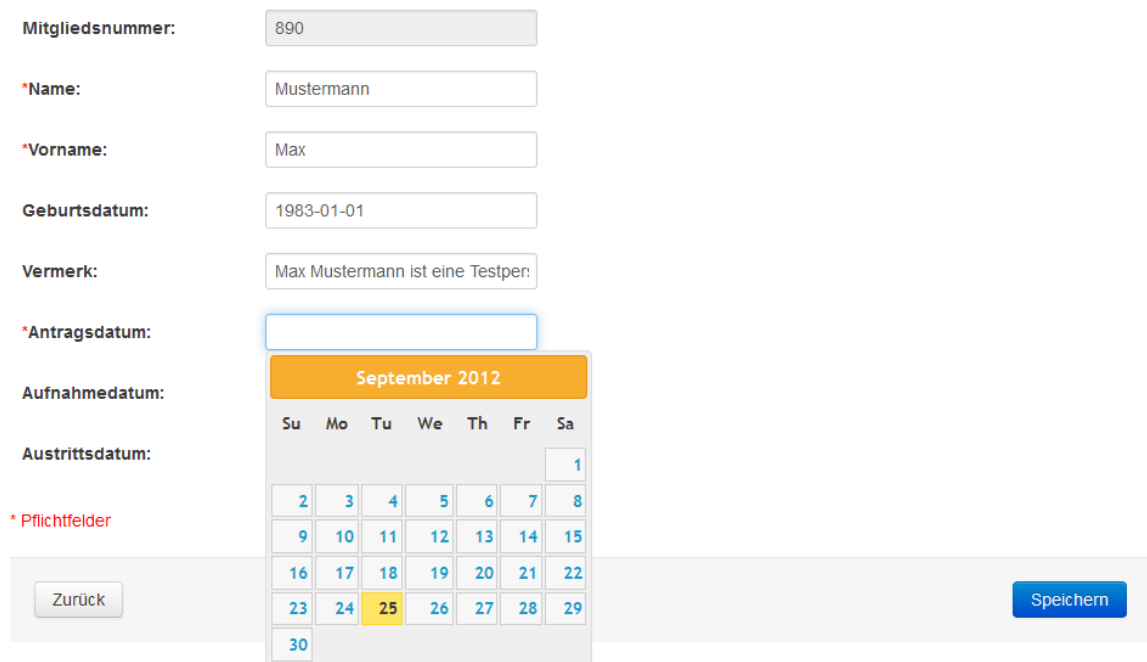


Abbildung 45: jQuery Datepicker

Das Eingabefeld „Antragsdatum“ hat `id="antragsdatum"`. Die jQuery Funktion benötigt diese `id` um den Kalender aufzurufen.

```
<script>
$(function() {
  $( "#antragsdatum" ).datepicker({
    dateFormat: 'dd.mm.yy'
  });
});
</script>
```

7 Fazit (PH/AL)

Die Projektarbeit gestaltete sich sehr angenehm, da das Projekt auch einen praktischen Nutzen hat, sollte aber vom zeitlichen Aufwand keinesfalls unterschätzt werden. Gerade die Absprache untereinander ist sehr wichtig für eine korrekte Koordination und einen ordentlichen Projektablauf. So sind beispielsweise für die Umsetzung der Geschäftsprozesse oft Fragen aufgekomen, wie das dahinterliegende Datenmodell mit diesen Daten umgeht und welche Daten für eine korrekte, logische Funktionsweise bereit stehen müssen.

Insgesamt hat auch diese Projektarbeit das Wissen merklich erweitert. Bei der Umsetzung in die Implementierung wurden mehrere Lösungen probiert und getestet. Falls es die Komplexität zulässt, wurde auch immer die effizientere Lösung bevorzugt. Leider konnte das nicht für jedes Problem so umgesetzt werden, da auch hier zwischen Aufwand und Nutzen abgewogen werden musste.

Insgesamt wurde von unseren Vorgängern ein guter „Durchstich“ in der Umsetzung der Anwendung erreicht, der jedoch nochmals vollständig überarbeitet wurde. Der Grund für die Überarbeitung lag darin, dass viele nützliche Features - die eigentlich für die Auswahl von Ruby on Rails als Webframework entscheidend waren - gar nicht umgesetzt wurden. Allerdings muss man an dieser Stelle anmerken, dass durch unsere Vorgänger sehr viele Geschäftsvorfälle umgesetzt worden sind.

Als Ausblick für die Zukunft sei an dieser Stelle auf die noch fehlende Lokalisierung und Formatierung der Daten (Datum), der Gleitkommazahlen (Komma statt Punkt) und evtl. falsche und fehlende Übersetzungen der Spalten hingewiesen. Auch die Validierung sollte nochmals als separater Punkt aufgefasst werden, denn nur ein Datenmodell mit gültigen Einträgen ist eine solide und brauchbare Datenbasis.

Als Grundlage über den aktuellen Stand der o/ZB-Webanwendung soll dieses Dokument dienen und damit weiteren Projektarbeiten eine aktuelle Wissensbasis geben.

Zum Ende dieses Dokumentes möchten wir uns an dieser Stelle bei Herrn Kienle, Herrn Kleinert und Herrn Schaefer bedanken. Sie standen uns während des gesamten Projektes jederzeit zur Verfügung. Der Kontakt war sehr freundlich und angenehm. Unsere Treffen im vier-wöchigen Rhythmus gaben uns eine Richtung und der Projektverlauf wurde klar aufgezeigt - wie in einem möglichen späteren Projekt im Berufsleben auch.

8 Abbildungsverzeichnis

Abbildung 1: Fehlerausgabe einer strukturierten und korrekt übersetzten Fehlermeldung	24
Abbildung 2: Mitgliederliste	27
Abbildung 3: Detailansicht eines Mitglieds	28
Abbildung 4: Kontenansicht aller OZB-Konten	29
Abbildung 5: Kontenklassenverlauf	30
Abbildung 6: jQuery's Datepicker Widget	30
Abbildung 7: EE-Eingabeformular	31
Abbildung 8: Fehler bei vollständig leerem EE-Formular	32
Abbildung 9: Erfolgreiches Speichern eines neuen Kontos	32
Abbildung 10: Formular des ZE-Konto	33
Abbildung 11: Übersicht der eigenen Konten nach dem Login	34
Abbildung 12: Übersichtsseite der Bürgschaften für das Mitglied Tassilo Kienle	35
Abbildung 13: Eingabeformular, um eine neue Bürgschaft zu erfassen	35
Abbildung 14: Nach dem erfolgreichen Anlegen einer Bürgschaft erscheint eine Erfolgsmeldung	36
Abbildung 15: Rechteverwaltung	55
Abbildung 16: Meine Daten - Details	58
Abbildung 17: Meine Daten - Details - Erfolgsmeldung	58
Abbildung 18: Meine Daten - Personaldaten bearbeiten	59
Abbildung 19: Meine Daten - Personaldaten bearbeiten - Fehlermeldung	60
Abbildung 20: Meine Daten - Kontaktdaten bearbeiten	61
Abbildung 21: Meine Daten - Kontaktdaten bearbeiten - Fehlermeldung	62
Abbildung 22: Meine Daten - Rolle bearbeiten - Fehlermeldung	64
Abbildung 23: Verwaltung - Alle Mitglieder anzeigen	68
Abbildung 24: Verwaltung - Neue Person hinzufügen - Erfolgsmeldung	69
Abbildung 25: Verwaltung - Details einer Person anzeigen	70
Abbildung 26: Verwaltung - Neues Mitglied hinzufügen	71
Abbildung 27: Verwaltung - Neue Person hinzufügen - Fehlermeldung	72
Abbildung 28: Verwaltung - Rolle bearbeiten	74
Abbildung 29: Verwaltung - Sonderberechtigungen bearbeiten	76
Abbildung 30: Verwaltung - Sonderberechtigung - Fehlermeldung	77
Abbildung 31: Verwaltung - Sonderberechtigung hinzufügen - Erfolgsmeldung	78
Abbildung 32: Verwaltung - Sonderberechtigung löschen - Erfolgsmeldung	79
Abbildung 33: Verwaltung - Sonderberechtigung hinzufügen	79
Abbildung 34: Beispiel der Formatierung einer CSV-Datei	80

Abbildung 35: Navigationsleiste Benutzer.....	90
Abbildung 36: Navigationsleiste Administrator	90
Abbildung 37: Sub Navigationsleiste (Tabs)	92
Abbildung 38: Pagination mit will_paginate und Twitter Bootstrap Design	93
Abbildung 39: Sortierung der Mitglieder nach Mitgliedernummer aufsteigend.....	96
Abbildung 40: Sortierung der Mitglieder nach Mitgliedernummer absteigend.....	97
Abbildung 41: Aufruf des modalen Dialogs bei „Suchen“	98
Abbildung 42: Modaler Dialog „Schnellsuche einer Person“	99
Abbildung 43: jQuery Autovervollständigung	99
Abbildung 44: Übernahme der Mitgliedernummer nach Schliessen des modalen Dialogs ..	101
Abbildung 45: jQuery Datepicker	102