

Dokumentation zur Projektarbeit

Michael Hoprich
Stephan Stroh
Nikolai Usow

Betreut von: Prof. Dr. Frank Schaefer, Hochschule Karlsruhe
Auftraggeber: Tassilo Kienle, o/ZB Stuttgart
Erstellt im: Wintersemester 2012/13

Inhaltsverzeichnis

1 Einleitung (M. Hoprich, S. Stroh und N. Usow)	5
2 Kennwort (S. Stroh)	6
2.1 Ansicht „Passwort ändern“	6
2.2 One-Way-Encryption	7
2.3 View	7
2.4 Controller	8
2.5 Model	8
3 Veranstaltungsverwaltung (S. Stroh)	10
3.1 Ansichten	10
3.1.1 Liste der Veranstaltungen	10
3.1.2 Neue Veranstaltung	11
3.1.3 Veranstaltungsteilnahmen	11
3.1.4 Veranstaltungsteilnahmen eines Mitgliedes	13
3.2 Attribute	14
3.3 View	14
3.3.1 editVeranstaltungen	14
3.3.2 listTeilnahmen	15
3.3.3 listVeranstaltung	15
3.3.4 newVeranstaltung	16
3.4 Veranstaltung_controller	16
3.4.1 listTeilnahmen	16
3.4.2 createDeleteTeilnahme	16
3.4.3 listVeranstaltung	16
3.4.4 editVeranstaltung	17
3.4.5 newVeranstaltung	17
3.4.6 createVeranstaltung	17
3.5 Models	17
3.6 Routes	17
4 Sonderberechtigungsverwaltung (S. Stroh)	18
4.1 Ansichten	18
4.1.1 Tabelle Sonderberechtigungen	18
4.1.2 Neue Sonderberechtigung	18
4.1.3 Liste der Geschäftsprozesse	19
4.2 Attribute	20
4.3 Sonderberechtigungs_Controller	21
4.3.1 listVeranstaltung	21
4.3.2 createBerechtigungRollen	21
4.3.3 deleteBerechtigungRollen	21
4.3.4 editRollen	21
4.3.5 createSonderberechtigung	21
4.4 Views	21
4.4.1 editRollen	21
4.4.2 createSonderberechtigungen	22
4.4.3 listGeschaeftsprozesse	22
4.5 Routes	22
5 Bürgschaften (S. Stroh)	23
5.1 Ansichten	23
5.1.1 Aktuelle Bürgschaften	23
5.1.2 Historisierte Bürgschaften	23
5.1.3 Neue Bürgschaft	24
5.1.4 Bürgschaft editieren	25

5.2 Attribute	26
5.3 Buergschaften_Controller	26
5.3.1 index	26
5.3.2 edit und editB	26
5.3.3 create	26
5.3.4 update	26
5.3.5 delete	26
5.3.6 listHisto	27
5.4 Routes	27
5.5 Views	27
5.5.1 _form	27
5.5.2 edit	28
5.5.3 editB	28
5.5.4 listHisto	28
6 RoR unter Windows mit Xampp (S. Stroh)	28
6.1 Port besetzt	28
6.2 Localhost wird nicht gefunden	29
7 Historisierung (N. Usow)	30
7.1 Beschreibung	30
7.2 Attribute der Historisierung	30
7.2.1 Vollständige Historisierung	31
7.2.2 Teilweise Historisierung	31
7.3 Beispielvorgang einer Historisierung	31
7.4 Implementierung der Historisierung	32
7.4.1 Automatische Historisierung	32
7.5 Implementierung	33
7.5.1 Callback: before_update	33
7.5.2 Callback: after_update	33
7.5.3 Adresse.get	34
8 Mitgliederverwaltung (N. Usow)	34
8.1 Beschreibung	34
8.2 Serverseitige Sortierung der Mitglieder	34
8.2.1 Parameter zur Steuerung der Sortierung	35
8.2.2 GET-Sortierparameter	35
8.2.3 Implementierung	36
8.3 Schnellsuche	37
8.3.1 Implementierung	38
8.3.2 Wiederverwendung der Implementierung	39
8.4 Benutzerbezogene Berechtigungen	40
8.4.1 Beschreibung	40
8.4.2 Dynamische Rechteanzeige für „Berechtigung hinzufügen“	41
8.4.3 Implementierung der dynamischen Rechteanzeige	41
8.5 Mitglied hinzufügen	43
8.5.1 Implementierung	44
8.5.2 Weitere Änderungen in „Mitglied hinzufügen“	45
9 Kontenverwaltung (N. Usow)	46
9.1 Beschreibung	46
9.2 Horizontales Scrollen	46
9.3 Kontendarstellung	47
9.3.1 Neue EE-Konto-Anzeige	47
9.3.2 Neue ZE-Konto-Anzeige	47
9.3.3 Weitere Änderungen an der Kontendarstellung	48
9.4 Letzte Bank und Bankverbindung beim EE-Konto	49

9.4.1	Implementierung	49
9.5	<i>Punkteberechnung</i>	50
9.5.1	Beschreibung	50
9.5.2	Implementierung	50
9.5.3	Benutzung	52
10	Webimport (N. Usow)	53
10.1	<i>Beschreibung</i>	53
10.2	<i>Dateistruktur</i>	53
10.3	<i>Anpassungen für die neue Ruby-Version</i>	53
10.4	<i>Implementierung</i>	54
11	Darlehensverlauf (M. Hoprich)	58
11.1	<i>Umsetzung Darlehensverlauf</i>	58
11.1.1	Anlegen des Controllers und der View	58
11.1.2	Umsetzung des Links auf der Kontonummer	58
11.1.3	Implementierung des Controllers	59
11.1.4	Implementierung des Falls, dass auf den Link eines Kontos geklickt wurde	59
11.1.5	Implementierung für den Fall das ein Datumsintervall angegeben wurde	61
11.1.6	Kontoauszug erstellen	63
11.1.7	Implementierung der View	63
12	Darlehensvertrag berechnen (M. Hoprich)	64
12.1	<i>Umsetzung Darlehensvertrag</i>	65
12.1.1	Anlegen des Controllers und der View	65
12.1.2	Erstellen der Eingabemöglichkeit	65
12.1.3	Berechnung der Werte	67
13	Fazit (M. Hoprich, S. Stroh und N. Usow)	71
14	Abbildungsverzeichnis	72

1 Einleitung (M. Hoprich, S. Stroh und N. Usow)

Die o/ZB (Ohne-Zins-Bewegung) Stuttgart ist eine Solidargemeinschaft für Sparen und Leihen. Sie verleiht ihren Mitgliedern Geld ohne Zinsen zu verlangen. Das dahinterstehende Konzept ist Ansparen, Leihen und Tilgen bzw. Nachsparen. Angespartes Geld kann jederzeit nach dem Nachsparen ausbezahlt oder für weitere Projekte genutzt werden. Dieses Konzept, welches in diesen drei Phasen abläuft ist so gestaltet, dass das Geben und Nehmen von Geld stets im Gleichgewicht bleibt. Der Ausgleich zwischen geliehenem und angespartem Geld wird dabei durch ein Punktesystem realisiert, welches ein faires Leihen und Sparen unter den Mitgliedern sicherstellt. Für eingehendere Informationen über die Ziele, das Konzept und die Struktur der o/ZB, sei hier auf die o/ZB eigene Website verwiesen.

Diese ist unter folgender Adresse zu erreichen: <http://www.ozb.eu>

Die Bearbeitung des o/ZB-Projekts mit Ruby-On-Rails setzt dabei auf die vorhergegangene Implementierung von Herr Aleksej Lednev, Robert Mrasek und Patrick Hillert auf. Sie haben sichergestellt, dass eine grobe Implementierung einiger Geschäftsprozesse und eine Vereinheitlichung des Datenmodells umgesetzt wurden.

Aufbauend auf dieses Gerüst konnten nun weitere Implementierungen an der Codebasis, Optimierungen am Datenmodell und ein Einsatz auf einem produktiven Test-Server realisiert werden. Das Ganze sollte dabei auf einem entwicklungsunabhängigen Server mit einer Ruby-Laufzeitumgebung umgesetzt werden.

Die Bearbeitung des Projekts wurde von vier Personen durchgeführt: Boris Filipov, Michael Hoprich, Stephan Stroh und Nikolai Usow.

Dieser Teil der Dokumentation beinhaltet dabei die folgenden und *teils* vorab festgelegten Punkte:

- Ergonomie und einheitlicheres Bildschirmdesign (N. Usow)
- Grundprinzipien für andere Entwickler (N. Usow)
- Anwendung von ergonomischen Richtlinien und Erweiterung der Geschäftsprozesse „Personen verwalten“ und „Konten verwalten“ (N. Usow)
- Punktberechnung (Darstellung der Konten) (N. Usow)
- Erweiterter Webimport (N. Usow)
- Anpassung und Erweiterung der Historisierung (N. Usow)
- Implementierung des Geschäftsprozesses für Darlehnsverwaltung (M. Hoprich)
- Darlehensverlauf (M. Hoprich)
- Darlehensvertrag (M. Hoprich)
- Rechteverwaltung (S. Stroh)
- Passwort ändern (S. Stroh)
- Veranstaltungsverwaltung (S. Stroh)
- Bürgschaften (S. Stroh)

Das Ziel dieser Entwicklungsiteration besteht darin, das System um weitere Geschäftsprozesse zu erweitern und eine bessere Konsolidierung von bestehenden Geschäftsprozessen sicherzustellen. Im Zuge der durchgeföhrten Datenbankmigration, mussten aber auch eine Vielzahl von bestehenden Geschäftsprozessen und Logiken erweitert bzw. sogar neuimplementiert werden.

2 Kennwort (S. Stroh)

Mitglieder des o/ZB sollen die Möglichkeit besitzen ihre persönlichen Passwörter zu ändern. Da einige Nutzer dazu neigen sich oftmals unsichere Kennwörter auszusuchen, wurde ein gewisser Sicherheitsstandard ausgesucht, welchen das gewünschte Passwort erfüllen muss, bevor dieses in die Datenbank übernommen wird. Das Benutzerkennwort kann demnach nicht mehr aus schlichten, schnell herauszufindenden Strings bestehen, beispielsweise sind unsichere Kennwörter wie „asdf“ oder „123456“ nicht mehr zugelassen. Auf die derzeitig bestehende Passwortkomplexität wird im Teil 1.5 genauer eingegangen. Diese Eigenschaft bietet sowohl dem Mitglied selbst, als auch den Geschäftsführern um einiges mehr an Sicherheit und eine geringere Angriffsfläche in Hinsicht auf Passwortdiebstahl.

Da die Authentifizierungslösung Devise bereits eingebunden wurde, konnte man auf die von Devise zur Verfügung gestellten Ressourcen zugreifen, um die Passwortänderung für die Benutzer zu ermöglichen.

2.1 Ansicht „Passwort ändern“

Der Bildschirm auf welchem der Nutzer die Möglichkeit der Passwortänderung erhält, ist recht trivial aufgebaut. Er wurde nicht überladen und dient nur dem Zweck der Änderung des aktuellen Passwortes, siehe Abb.1. Das Mitglied findet diesen, indem es unter „Meine Daten“ den Reiter „Passwort ändern“ aufruft.

Es wird grundsätzlich das aktuelle Passwort benötigt, dadurch wird sichergestellt, dass sich keine dritte Person bei offenem Zugang erlaubt, die Zugangsdaten eines ihm fremden Accounts zu verändern. Nach dem Ausfüllen des aktuellen Passwortfeldes, kann das neue Passwort eingegeben und anschließend bestätigt werden. Sollte die Passwortänderung erfolgreich verlaufen, wird das Mitglied auf seinen eigenen Startbildschirm weitergeleitet und erhält dort die Meldung einer erfolgreichen Aktualisierung. Sollte jedoch ein Fehler aufgetreten sein, so verbleibt das Mitglied auf der derzeitigen Ansicht und erhält die Ursachen, welche für die fehlerhafte Speicherung verantwortlich sind, angezeigt. Um dem Mitglied die Richtlinien des Passwortes näherzubringen, wurden diese auf die rechte „leere“ Seite verschoben, sodass diese mehr ins Auge des Betrachters des Bildschirms fallen.

Meine Daten

Details Personaldaten Kontaktdaten Rolle Passwort ändern

Passwort zum internen o/ZB-Bereich ändern:

Aktuelles Passwort:	<input type="text" value="Aktuelles Passwort"/>	* Das Passwort muss mindestens einen Großbuchstaben, einen Kleinbuchstaben und eine Zahl enthalten. Die Passwortlänge darf sechs Zeichen nicht unterschreiten.
*Neues Passwort:	<input type="text" value="Neues Passwort"/>	
*Neues Passwort bestätigen:	<input type="text" value="Neues Passwort bestätigen"/>	

[Zurück](#) Passwort ändern

Abb. 1: Ansicht "Passwort ändern"

2.2 One-Way-Encryption

Die Aufbewahrung des Kennwertes erfolgt in der Tabelle „*ozbperson*“ in der Spalte „*encrypted_password*“. Darin wird der Hashwert des Benutzerpasswortes gespeichert. Aus diesem Hashwert ist es praktisch unmöglich das ursprünglich eingegebene Kennwort herauszufinden. Um die Richtigkeit des vom Nutzer eingegebenen Passwortes zu überprüfen, wird der eingegebene String erneut gehasht und die beiden Hashwerte miteinander verglichen. Sollten diese übereinstimmen, wurde von Benutzer das richtige Passwort eingegeben.

Hierfür verwendet Devise die bcrypt-Hashfunktion. Diese gilt als besonders zeitaufwendig in der Hinsicht auf Brute-Force-Attacken. Im Gegensatz zu den Hashfunktionen Md5 und SHA ist das Hashing bei bcrypt möglichst aufwendig gestaltet worden. Für alltägliche Anwendungszwecke fällt dieser Aufwand, gegenüber anderen Faktoren, nicht ins Gewicht, erst wenn die Berechnung häufig hintereinander durchgeführt werden soll, tritt eine entscheidende Verlangsamung ein.

2.3 View

Um an die Views zu gelangen, welche Devise bereits zur Verfügung stellt, muss man im cmd mit Ruby in den Ordner der OzB-Anwendung navigieren und den Code

„rails generate devise:views Ozbperson“

ausführen. Dann werden alle notwendigen Views als editierbare Dateien generiert und in einem Unterordner von View abgespeichert. Die View welche für das Ändern des Passwortes von Bedeutung ist, befindet sich in „*/view/devise/registration/edit*“. Diese beinhaltet eine grobe Vorlage für die Änderung des Benutzerpasswortes. Um in der Anwendung auf diese View zugreifen zu können, bietet Devise vordefinierte Pfade, so müssen diese nicht mehr in „*routes.rb*“ nachträglich eingetragen werden.

Die Pfade von Devise erhält man am einfachsten, wenn man „*rake routes*“ in der cmd ausführt. Im Falle der Passwortänderung war „*edit_OZBPerson_registration_path*“ der Pfad für die oben genannte View. Dieser Pfad wurde als Listenpunkt zu den bereits vorhandenen Punkten in der Navigation der Ozbperson hinzugefügt(Abb.2):

```
<li <% if getCurrentLocation == edit_OZBPerson_registration_path %>class="active"<%end%>>
  <%= link_to "Passwort ändern", edit_OZBPerson_registration_path %>
</li>
```

Abb. 2: Pfad „Passwort ändern“

Die erforderlichen Textfelder haben Platzhalter erhalten, um leere Felder zu vermeiden und somit dem Nutzer eine ansprechendere Ansicht zu bieten. Die Form mit den zu ändernden Passwort wird nur abgesendet, wenn die erscheinende Bestätigungsanzeige mit „ok“ quittiert wird, ansonsten kann die Passwortänderung noch abgebrochen werden (Abb.3). Eine unbeabsichtigte Änderung des Passwortes ist, durch die erwähnten Maßnahmen, also unwahrscheinlich.

```
<%= f.submit "Passwort ändern", :onclick => "return confirm('Passwort wird geändert')",
:klass => "btn btn-primary", :notice =>"Passwort erfolgreich geändert"%>
```

Abb. 3: Passwort Bestätigung

2.4 Controller

Der Controller von Devise lässt sich nicht ohne weiteres als editierbare Datei generieren, wie es bei den Views der Fall war. Jedoch bietet es sich an, Teile der Devise Methoden zu ersetzen, oder ggfs. eigenen Code hinzuzufügen, welchen Devise mitausführen soll.

Es war gewünscht, dass nach einem Ändern des Passwortes das Mitglied auf seine Startseite verwiesen wird und dort eine Bestätigungsmeldung erscheint. In diesem Fall konnte der „root“-Index nicht verwendet werden,

„root :to => ,willkommen#index“

da zwischen Mitgliedern mit Sonderberechtigung und Mitgliedern ohne Sonderberechtigung unterschieden werden musste. Die erstenen haben als Startseite, die Liste der Mitglieder, die letzteren die Auflistung ihrer verschiedenen Konten. Um dennoch eine funktionierende Weiterleitung zu gewährleisten, wurde im „application_controller“ der Pfad, „after_update_path_for(resource)“, den Devise für die Weiterleitung nach dem Aktualisieren der Registrationsdaten verwendet überschrieben (Abb.4):

```
def after_update_path_for(resource)
  if is_allowed(current_OZBPerson, 1) then
    "/Verwaltung/Mitglieder"
  else
    "/MeineKonten"
  end
end
```

Abb. 4: Überschreiben After_Update

Es wird jeweils eine Benachrichtigung über die erfolgreiche Aktualisierung des Kontos eingeblendet. Die deutsche Übersetzung der Nachrichten welche von Devise vorgegeben sind, befindet sich unter „config/locales/devise.de.yml“ und lässt sich ebenfalls schnell und ohne großen Aufwand anpassen.

2.5 Model

Das Passwort sollte eine Mindestlänge, hier sechs Zeichen, aufweist, es müsste mindestens ein Großbuchstabe, ein Kleinbuchstabe und eine Zahl im Passwort vertreten sein. Diese Vorgabe wurde dadurch erreicht, dass das eingegebene Passwort, vor dem Eintragen in die Datenbank, auf seine Komplexität überprüft wird. Falls dies den vorgegebenen Richtlinien nicht entspricht, wird eine Fehlermeldung zurückgeliefert. Da die Passwörter den Ozb_Personen zugewiesen werden, wird das Model der „ozb_person“ um den folgenden Code der Passwort-Validierung ergänzt (Abb.5):

```

validate :password_zusammensetzung

def password_zusammensetzung
  if password.present? and not password.match(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).+/)
    errors.add :password, " muss mindestens einen Großbuchstaben, einen Kleinbuchstaben und eine Zahl enthalten.
                           Die Passwortlänge darf sechs Zeichen nicht unterschreiten."
  end
end

```

Abb. 5: Validierung des Passwortes

Die Passwort-Validierung wird lediglich bei einem vorhandenen Kennwort erfolgen, dies ist nur der Fall, wenn ein neues Mitglied erstellt wird, oder ein bereits bestehendes Mitglied aktualisiert wird. Es wird durch den obigen (Abb.5) regulären Ausdruck („*regex*“, regular expression) sichergestellt, dass eines der Zeichen des Passwortes ein kleiner Buchstabe (`(?=.*[a-z])`), ein großer Buchstabe (`(?=.*[A-Z])`) und eine Zahl(digit) (`(?=.*\d)`) ist. Bei Prüfung der Zahl wäre auch (`(?=.*[0-9])`) denkbar gewesen. Demzufolge kann ein akzeptables Passwort wie folgt aussehen: „Test56“. Die Verifizierung der Länge erfolgt durch Devise selbst, da die gewünschte Passwortlänge, mit der von Devise standardmäßig vorgegebenen, übereinstimmt, musste diese nicht angepasst werden. Sollte ein Fehler bei der Passwortänderung auftreten, wird auf der Seite verweilt und die entsprechenden Fehlermeldungen ausgegeben. Bei erfolgreicher Änderung gelangt das Mitglied auf jeweilige Startseite.

3 Veranstaltungsverwaltung (S. Stroh)

Veranstaltungen lassen sich von Mitgliedern mit den dafür befähigten Sonderberechtigungen anlegen und bearbeiten. Hierfür ist eine der drei Sonderberechtigungen notwendig:

- MV (Mitgliederverwaltung)
- IT (Administrator)
- OeA (Öffentlichkeitsverwaltung)

Diese erhalten die Möglichkeit neue Veranstaltungen zu erstellen und in die Datenbank einzutragen. Zu den entsprechenden Veranstaltungen können dann Teilnahmen hinzugefügt und wieder entfernt werden. Mitgliedern ohne Sonderberechtigung wird in der Navigationsleiste die Option der Veranstaltungsverwaltung nicht angezeigt. Es sind die vier folgenden Veranstaltungarten vorhanden:

- Schulung
- Gesellschafterversammlung
- Vortrag
- Seminar

Je nach Veranstaltungsart wird beim Erstellen das gewünschte Publikum eingeladen. Dieser Vorgang wird in der Methode 2.4.6 „createVeranstaltung“ ausführlich behandelt.

3.1 Ansichten

3.1.1 Liste der Veranstaltungen

Diese Ansicht bietet eine Auflistung der gesamten Veranstaltungen an. Es ist möglich durch Klicken auf die jeweilige Veranstaltung, zu dieser direkt zu gelangen, um daran Änderungen vorzunehmen. Um den Bildschirm zu erreichen wird auf der oberen Navigationsleiste „Weiteres/Veranstaltungen“ gewählt. Wie bereits im vorherigen Teil erwähnt, bietet sich den Mitgliedern ohne die entsprechende Sonderberechtigung der Zutritt zu den Veranstaltungen nicht an.

♦ Nr.	♦ Veranstaltung	♦ Datum	♦ Ort
1	Gesellschafterversammlung	21.01.2005	Kolpinghaus
50	Gesellschafterversammlung	14.04.2012	BZ-Stuttgart Ost
51	Schulung	12.05.2012	BZ-Stuttgart Ost
52	Vortrag	01.05.2012	Vaihingen/Enz

Abb. 6: Tabelle Veranstaltung

Das Datum ist im europäischen Format gehalten. Die Sortierung der einzelnen Elemente ist möglich, dies wird hier nicht durch eine serverseitige Sortierung erreicht. Es werden also nur die

gelisteten Veranstaltungen sortiert, da allerdings keine Unmenge an Veranstaltungen zu erwarten ist und die Sortierung nebensächlich ist, ist eine clientseitige Sortierung durch JQuery angemessen.

3.1.2 Neue Veranstaltung

Um eine neue Veranstaltung anzulegen, muss eine der vier vorgegebenen Veranstaltungsarten ausgewählt werden. Die Felder mit dem Austragungsort und dem Austragungsdatum sind ebenfalls beide Pflichtfelder, ohne die eine Erstellung der Veranstaltung nicht möglich ist. Um die gewünschte Veranstaltung anzulegen, klickt man auf die Schaltfläche „Speichern“. Ist der Speichervorgang erfolgreich verlaufen, wird das Mitglied auf die Seite mit den gelisteten Veranstaltungen (Abb.6) verwiesen und es erscheint eine Bestätigungsmeldung. Die soeben eingetragene Veranstaltung ist nun bereits in dieser Tabelle anzufinden und kann auch bearbeitet werden. Bei fehlerhafter Speicherung verbleibt man auf diesem Bildschirm und erhält eine Fehlermeldung angezeigt.

Besser ohne / Zins Meine Konten Meine Daten Protokolle o/ZBlick Mitglieder Weiteres ▾ ⌂ Ausloggen

Neue Veranstaltung hinzufügen

*Veranstaltungsart: Bitte auswählen

*Ort: Veranstaltungsort

*Datum: 25.02.2013

* Pflichtfelder

Zurück Speichern

Abb. 7: Ansicht neue Veranstaltung

3.1.3 Veranstaltungsteilnahmen

Durch Klicken auf eine Zeile in der Tabelle mit den Veranstaltungen, gelangt man auf die gelistete Veranstaltung und bekommt die Mitglieder mit den dazugehörigen Teilnahmearten angezeigt. Die möglichen Teilnahmearten lauten:

- l (eingeladen)
- a (anwesend)
- e (entschuldigt)
- u (unentschuldigt)

Beim Hovern über die Kürzel werden immer die ausgeschriebenen Teilnahmearten eingeblendet. Um ein weiteres Mitglied der Veranstaltung hinzuzufügen, benutzt man die Form „Teilnehmer hinzufügen“, welche sich über der Tabelle mit den Teilnahmen befindet.

Das einzutragende Mitglied wird über das Textfeld mit der Auto vervollständigung gesucht, dann wird die erbrachte Teilnahmeart ausgewählt und durch das Bestätigen durch die „Hinzufügen“-Schaltfläche wird die neue Teilnahme in die Datenbank eingetragen. Anschließend wird die Seite neu geladen und die eingetragene Teilnahme wird in der Tabelle gelistet. Wenn sich das Mitglied welches man mit Hilfe der „Teilnehmer hinzufügen“-Form einzutragen versucht, bereits in der Tabelle befindet, wird die Teilnahmeart aktualisiert. Eine entsprechende Benachrichtigung wird angezeigt. Die folgende Abbildung (Abb.8) zeigt die Anzeige mit den Teilnahmen:

Teilnehmer hinzufügen

Mnr.	Mitglied	Anwesenheit
<input type="text" value="Mnr."/>	<input type="text" value="Mitglied eingeben"/>	<input type="text" value="u"/> <input type="button" value="Hinzufügen"/>

Gesellschafterversammlung am 14.04.2012, BZ-Stuttgart Ost

Mnr.	Rolle	Name	Vorname	Teilnahmeart
5	S	Seidel1	Wolfgang1	a
6	G	Juhas	Anton	e
7	G	Ochs	Armin	e
8	G	Peckmann	Ulrich	a
10	G	Schmitz	Christoph M.	a
11	M	Schmitz	Monika	a
12	M	Kienle	Hannelore	a
13	G	Kienle1	Tassilo1	a
14	M	Hamann	Rolf	e
16	P	Juhas	Inge	e
19	G	Fischer	Günter	a
24	M	Müller	Rita	a
27	M	Eisenlohr	Annegret	u
28	M	Fitz	Ursula	u
30	M	Mühlbauer	Lina-Elisabeth	e
33	M	Christmann	Hartmut	e
35	G	Kienle	Friederike	e

Abb. 8: Auflistung Teilnahmen

Die Teilnahmeart lässt sich ändern, indem man auf das Kürzel der Teilnahmeart von der zu editierenden Person klickt. Es erscheint Anstelle des einfachen Kürzels eine DropDown Auswahl mit den verfügbaren Teilnahmearten (l, a, e, u). Hierdurch kann nun die Teilnahmeart bequem verändert werden. Beim Auswählen einer neuen Teilnahmeart in der DropDown-Auswahl wird eine Form, mit den zu der Person gehörenden Daten, versendet. Die eingetragene Teilnahmeart wird durch die neu ausgewählte überschrieben und kann somit nicht mehr rekonstruiert werden.

Die Teilnahmen lassen sich durch das Betätigen der roten Schaltfläche mit dem Mülltonnensymbol entfernen. Durch das Hovern darüber wird „Löschen“ eingeblendet. Bevor man jedoch unbedacht eine Teilnahme entfernt, wird man erst aufgefordert dies zu bestätigen. Falls dies wirklich gewünscht ist, wird die Teilnahme restlos aus der Liste und somit der Datenbank entfernt. D.h. an dieser Stelle ist keine Historisierung vorhanden. Am unteren Ende jeder Teilnehmerliste befindet sich eine tabellarische Unterteilung der einzelnen Teilnahmearten und der zugehörigen Rolle der Mitglieder (Abb.9).

The screenshot shows a software interface with a dark header bar containing navigation links: "Besser ohne / Zins", "Meine Konten", "Meine Daten", "Protokolle", "o/ZBlick", "Mitglieder", "Weiteres ▾", and "Ausloggen". Below this is a table of 141 participants, each with a small red trash icon in the first column. The columns represent participation counts for four categories: I (Individuum), a (Anonym), e (Einheit), u (Unbekannt), and Gesamt (Total). The table data is as follows:

	I	a	e	u	Gesamt
Gesellschafter	0	19	4	9	32
Mitglieder	0	15	14	25	54
Partner	0	2	3	1	6
Studenten	0	1	4	5	10
	0	37	25	40	102

In the bottom right corner of the main content area, there is a small button labeled "Zurück".

Abb. 9: Teilnehmer-Statistik

Es wird zu jeder Rolle angezeigt, wie viele Personen, welche der vier Teilnahmearten aufweisen. Auf der rechten Seite wird die Anzahl der Vertreter der entsprechenden Rolle angezeigt. Im Footer ist die Anzahl der einzelnen Teilnahmearten zu sehen. Da die Menge der Mitglieder und die Menge der Teilnehmer übereinstimmen muss, kann die Gesamtanzahl unten rechts sowohl den Teilnehmern als auch den Mitgliedern zugeordnet werden. Beide Zuordnungen sind richtig.

3.1.4 Veranstaltungsteilnahmen eines Mitgliedes

Wenn man in der Ansicht mit den Veranstaltungen (Abb.7) auf die Zeile mit der Teilnahme einer Person klickt, gelangt man zu den persönlichen Veranstaltungsteilnahmen dieser Person. D.h. Es werden die Veranstaltungen angezeigt, zu denen eine Teilnahme von dieser Person in der Datenbank existiert. Es kann jede der vier Teilnahmearten auftreten.

Durch diese Auflistung erhält man einen guten Überblick über alle Teilnahmen einer Person und es kann durch Klicken auf die Spalte zu der Veranstaltung gesprungen werden. Dann wird der Bildschirm (Abb.8) mit den Teilnahmen dieser Veranstaltung dargestellt. Der Reiter „Teilnahmen“ wird nur den Mitgliedern mit den Sonderberechtigungen sichtbar gemacht:

- MV (Mitgliederverwaltung)
- IT (Administrator)
- OeA (Öffentlichkeitsverwaltung)

Palatini, Paolo

OZB-Konten Details Personaldaten Kontaktdaten Gesellschafterdaten Bürgschaften **Teilnahmen**

Veranstaltungsteilnahmen

Datum	Vnr.	Veranstaltungsart	Ort	Teilnahmeart
14.04.2012	50	Gesellschafterversammlung	BZ-Stuttgart Ost	a
12.05.2012	51	Schulung	BZ-Stuttgart Ost	a

[Zurück](#)

Abb. 10: Teilnahmen eines Mitglieds

3.2 Attribute

Da es durchaus vorkommen kann, dass unterschiedliche Mitglieder mit Sonderrechten an den Veranstaltungen arbeiten werden und diese ändern oder hinzufügen, ist es sinnvoll die Personennummer zu notieren welche die letzte Änderung vorgenommen hat. Daher wurde die Tabelle der Veranstaltung und die Tabelle der Teilnahmen jeweils um die Spalte „*SachPnr*“ erweitert. Nun wird bei jeglicher Änderung an den Veranstaltungen und Teilnahmen festgehalten, welches Mitglied diese getätigt hat. Allerdings ist hier keine Historisierung vorhanden. Es wird nur die jeweils letzte Änderung festgehalten, was davor gemacht wurde lässt sich dann nicht mehr rekonstruieren.

Vnr	vid	VADatum	VAOrt	SachPnr
1	1	2005-01-21	Kolpinghaus	NULL
50	1	2012-04-14	BZ-Stuttgart Ost	NULL
51	2	2012-05-12	BZ-Stuttgart Ost	NULL
52	3	2012-05-01	Vaihingen/Enz	NULL
53	2	2013-02-27	Stuttgart	13

Abb. 11: DB-Tabelle Veranstaltung

Die NULL-Werte ergeben sich daher, da die Einträge bereits in der Datenbank aufgelistet waren bevor die „*SachPnr*“ hinzugefügt wurde und man nicht mehr nachvollziehen konnte, welches Mitglied die Veranstaltung eingetragen hat.

Jede neue Veranstaltung hat eine eindeutige „*Vnr*“, diese wird nicht mehr durch das Mitglied ausgesucht welches die Veranstaltung erstellt. Die „*Vnr*“ wird nun einfach inkrementiert, dadurch spart man Tipparbeit und es gibt eine mögliche Fehlerquelle weniger.

3.3 View

3.3.1 editVeranstaltungen

Diese View beinhaltet die Anzeige der Tabelle mit den Veranstaltungen. Es werden lediglich die Veranstaltungen und deren Teilnahmeart in tabellarischer Form dargestellt.

3.3.2 listTeilnahmen

Die Pakete mit den logisch zusammenhängenden Objekten werden ausgelesen und dargestellt. So werden alle Teilnahmen, der ausgesuchten Person aufgelistet. Dass jede Spalte durch einen Klick auf die jeweilige Veranstaltung verweist, wird durch folgenden Code erreicht:

```
<% veranstaltung_link = "/Verwaltung/Veranstaltungen/" +veranstaltung.Vnr.to_s + "/Ansicht" %>
<tr onclick="location.href='<%= veranstaltung_link %>'>">
```

Abb. 12: Link zur Veranstaltung

Jede Spalte erhält einen zugehörigen „veranstaltung_link“ zu welchem, bei einem Klick, navigiert wird. Damit sich beim Hovern über eine Spalte, diese deutlicher von den anderen abhebt und damit es auch als anklickbare Spalte erkannt wird wurde folgender Code verwendet:

```
<style type="text/css">

.table tbody tr:hover td {
    background-color: #d2e6f4;
}

tr {
    cursor: pointer
}

</style>
```

Abb. 13: Stylesheet der Tabelle

Die Farbe der Spalte wird auf die oben gelistete(helles marineblau) geändert und der Mauszeiger ändert sich zu einer Hand mit ausgestrecktem Zeigefinger. Nun ist es erwartungskonform, was einen anklickbaren Link angeht. Die anklickbaren Spalten wurden bei den anderen Views nach dem gleichen Prinzip realisiert und werden aufgrund von Redundanzen nicht mehr ausführlich auf diese Hinsicht beschrieben.

3.3.3 listVeranstaltung

Diese View listet die einzelnen Teilnehmer einer Veranstaltung auf und ermöglicht es diese zu editieren und zu löschen. Dabei werden die Parameter der Form für den Löschvorgang oder aber für den Editervorgang genutzt, es wird mit Hilfe der JavaScript Funktion „submit_form“ bzw. „submit_form_delete“ die Form mit den zugehörigen Parametern an den „veranstaltung_controller“ gesendet.

```
function submit_form_delete(pnr,vnr){
    var value = document.getElementById(pnr+'').value;
    document.getElementById('vnr1').value = vnr;
    document.getElementById('pnr1').value = pnr;
    document.getElementById('teilnArt2').value = value;
    document.getElementById('delete').value = "true"
    $('#form_createTeilnahme_Zwo').submit()
}
```

Abb. 14: JavaScript Function “submit_form_delete”

Jede der Spalten mit der Teilnahmeart hat eine eindeutige ID erhalten, damit diese aus der Menge herausgesucht werden kann, um die richtigen Daten für die Form zu liefern.

Die Funktion „bearbeiten“ wird aufgerufen, wenn man einen nicht editierbaren Tabelleneintrag der Teilnahmeart anklickt. Dann wird die Sichtbarkeit vertauscht und es kann nun mit Hilfe der DropDownList Auswahl die Teilnahmeart verändert werden:

```

function bearbeiten(pnr){
    var noEdit = document.getElementById("as"+pnr);
    var Edit = document.getElementById("be"+pnr);
    noEdit.style['display'] = 'none';
    Edit.style['display'] = 'block';
}

```

Abb. 15: JavaScript Function „bearbeiten“

Um die Tabelleneinträge unterscheiden zu können, benötigt jeder Eintrag eine eindeutige Kennung. Hier wird dies durch das Zuteilen einer ID, bestehend aus einem Präfix und der „Pnr“ des Mitglieds(abb.16):

```

<% id = "be" + person.Pnr.to_s %>
<td name="check" id=<%= id%></td>

```

Abb. 16: ID-Zuweisung Tabelleneintrag

3.3.4 newVeranstaltung

Diese View bietet einem eine Form für die Erstellung und Eintragung einer neuen Veranstaltung in die Datenbank an.

3.4 Veranstaltung_controller

Im Controller für die Veranstaltungen werden neue Veranstaltungen und Teilnahmen mit den übergebenen Parametern erstellt, validiert, gespeichert oder gelöscht.

3.4.1 listTeilnahmen

Die Methode „ListTeilnahmen“ wird aufgerufen, wenn man die Teilnahmen des einzelnen Mitglieds anschauen möchte. Es wird nach dem Mitglied gesucht, die entsprechenden Teilnahmen werden aus der Datenbank gelesen. Falls es keine Teilnahmen gibt, bleiben die nachfolgenden Schritte ohne weitere Auswirkungen. Falls eine Teilnahme gefunden wird, werden die entsprechenden Daten, wie die dazugehörige Veranstaltung, die Veranstaltungsart, und die dazu passende ausgeschriebene Veranstaltungsart(zum Einblenden beim Hovern) gefunden und in das @veranstaltungen Array gepackt. So kann dieses logisch zusammenhängende Gesamtpaket in der View leicht und organisiert ausgelesen werden, ohne dass weitere Schritte in der View ausgeführt werden müssten.

3.4.2 createDeleteTeilnahme

Mit dieser Methode wird entweder eine neue Teilnahme erstellt, oder eine bereits vorhandene Teilnahme aus der Datenbank entfernt. Falls der Parameter „delete“ gesetzt ist, wird mitgeteilt, dass die Teilnahme, dessen Parameter man mitgibt, aus der Datenbank entfernt werden soll. Ist der „delete“ Parameter nicht gesetzt, so wird mit den gegebenen Parametern eine neue Teilnahme erstellt, überprüft und in die Datenbank eingetragen. Dadurch werden mit einer Form zwei Bereiche abgedeckt. Es wird überprüft, ob das Mitglied welches die Methode aufgerufen hat überhaupt die Rechte besitzt, um eine solche Änderung an der Datenbank vorzunehmen. Sollte dies der Fall sein, so wird nun unterschieden, ob eine Teilnahme erstellt oder gelöscht werden soll. Wenn die Teilnahme gelöscht werden soll, wird diese in der Datenbank gesucht und gelöscht. Wenn keine Teilnahme gefunden wird, erscheint eine Fehlermeldung. Beide Möglichkeiten verweisen anschließend auf dieselbe Seite, von der aus die Anfrage gestartet wurde.

3.4.3 listVeranstaltung

Diese Methode wird aufgerufen, nachdem man auf eine Veranstaltung geklickt hat. Man gelangt dann auf die View mit den Teilnahmen dieser Veranstaltung. Es wird die Veranstaltung aus der Datenbank geholt, die entsprechende Veranstaltungsart gesucht und alle Teilnahmen geladen. Zu

jeder Teilnahme wird die dazugehörige Person geladen. Es wird festgehalten, wie viele von welchen Mitglieder teilgenommen haben und welche Teilnahmearten diese aufweisen.
 @AnzahlRollen besteht aus [Rolle, Anzahl Eingeladen, Anzahl Anwesend, Anzahl Entschuldigt, Anzahl Unentschuldigt, Anzahl Gesamt] Jeder Mitgliederrolle ist ein solches Paket im Array zugeschoben. Dies erleichtert das Darstellen der Statistik enorm.

3.4.4 editVeranstaltung

Die Methode „EditVeranstaltungen“ wird aufgerufen wenn man unter „Weiteres“ auf „Veranstaltung“ klickt. Es werden die vorhandenen Veranstaltungen geladen und mit der entsprechenden Veranstaltungsart in ein Array gepackt.

3.4.5 newVeranstaltung

Die Methode „NewVeranstaltung“ wird aufgerufen, wenn der Bildschirm für die Erstellung einer neuen Veranstaltung aufgerufen wird. Es werden lediglich die Veranstaltungsarten für die DropDownList Auswahl der Teilnahmearten geladen.

3.4.6 createVeranstaltung

Mit den übergebenen Parametern wird eine neue Veranstaltung angelegt, validiert und in die Datenbank eingetragen. Da bei der Schulung („vid“ = 2) und bei der Gesellschafterversammlung („vid“ = 1) bereits bei der Erstellung der Veranstaltung, bestimmte Mitglieder eingeladen werden sollen, wird überprüft welche „vid“ die neue Veranstaltung aufweist. Ist es eine der beiden obengenannten, werden bestimmte Mitglieder eingeladen. Für die Art „Schulung“ sind die aktiven(„austrittsDatum“ = NULL) Personen ohne Schulungsdatum („schulungsDatum“ = NULL) einzuladen. Bei der Gesellschafterversammlung sind die aktiven Mitglieder, Gesellschafter und Partner gefragt. Die Personen werden aus der Datenbank gelesen und anschließend wird, zu jedem gefundenen Mitglied, eine Teilnahme für die gerade angelegte Veranstaltung erstellt. Die neu erstellten Teilnahmen haben die Teilnahmeart „eingeladen“.

3.5 Models

Die Models welche hierfür notwendig waren, lauten „teilnahme“, „veranstaltung“ und „teilnahmeArt“, diese sind simpel aufgebaut und erklären sich von selbst.

3.6 Routes

Die neuen Routes, welche für die Veranstaltungen benötigt werden, dienen dem Aufruf der entsprechenden Methode aus dem „veranstaltung_controller“.

```
match '/Verwaltung/Veranstaltungen/:Vnr/Ansicht' => 'veranstaltung#listVeranstaltung', :via => :GET
match '/Verwaltung/Veranstaltungen/:Vnr/Ansicht' => 'veranstaltung#createDeleteTeilnahme', :via => :POST
match '/Verwaltung/Veranstaltungen' => 'veranstaltung#editVeranstaltungen', :via => :GET
match '/Verwaltung/Veranstaltungen' => 'veranstaltung#createVeranstaltung', :via => :POST
match '/Verwaltung/OZBPerson/:Mnr/Teilnahmen' => 'veranstaltung#listTeilnahmen'
```

Abb. 17: Routes Veranstaltung

4 Sonderberechtigungsverwaltung (S. Stroh)

Mitglieder werden mit Sonderberechtigungen ausgezeichnet, um besondere Aufgaben in der Applikation durchführen zu können. Alle Berechtigungen haben einen eingeschränkten Zugriff auf die Bereiche der Applikation, außer der IT(Administrator). Um den Überblick über die vergebenen Sonderberechtigungen zu behalten, wurde eine übersichtliche Verwaltung dieser angelegt. Es ist auf einen Blick ersichtlich, wer welche Berechtigung besitzt und man kann diese bei Bedarf entfernen oder erweitern.

4.1 Ansichten

4.1.1 Tabelle Sonderberechtigungen

The screenshot shows a web-based application interface for managing member permissions. At the top, there is a navigation bar with links: 'Besser ohne / Zins', 'Meine Konten', 'Meine Daten', 'Protokolle', 'o/ZBlick', 'Mitglieder', 'Weiteres ▾', and 'Ausloggen'. Below the navigation bar, the main title is 'Verwaltung: Mitglieder'. Underneath, there is a sub-section titled 'Mitglieder mit Sonderberechtigungen'. A table lists one member with ID 13, name 'Kienle1, Tassilo1', and a green 'Ja' button in the 'IT' column, indicating they have been granted IT permissions. To the right of the table is a text input field labeled 'Neue E-Mail eintragen' with a small envelope icon. At the bottom of the table area, there are buttons for 'Zurück' (Back) and 'Neue Sonderberechtigung' (New Permission). A blue header bar at the very bottom of the page reads 'Geschäftsprozesse'.

Abb. 18: Tabelle Sonderberechtigungen

In der Ansicht werden nun alle Mitglieder mit Sonderberechtigungen gelistet. Man erhält hier die Möglichkeit den bereits in der Liste vorhandenen Mitgliedern durch das Klicken auf ein rotes „Nein“ die jeweilige Sonderberechtigung zu erteilen. Oben rechts erkennt man das Feld E-Mail, dieses wird benötigt um der neuen Sonderberechtigung eine separate E-Mail zu vergeben, unter welcher man z.B. als Admin der Seite angeschrieben werden kann. Bei leer lassen des E-Mail Feldes wird die von Mitglied standardmäßige E-Mail Adresse verwendet. Ist eine solche nicht vorhanden, so kann ohne E-Mail keine Sonderberechtigung erstellt werden. Beim Klick auf ein grünes „Ja“ wird nach einem Bestätigungsfenster die Sonderberechtigung vom jeweiligen Mitglied entfernt. Falls das Mitglied keine weiteren Sonderberechtigungen hat, wird dieses in dieser Tabelle nicht mehr gelistet. Beim Klicken auf den Namen der Person gelangt man zu der Ansicht mit den Berechtigungen des einzelnen Mitglieds.

4.1.2 Neue Sonderberechtigung

Um eine neue Sonderberechtigung für ein noch nicht gelistetes Mitglied zu erstellen, wird auf die Schaltfläche „Neue Sonderberechtigung“ geklickt. Daraufhin öffnet sich der folgende Bildschirm:

Verwaltung: Mitglieder

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

Neue Sonderberechtigung erstellen

Mnr.	Mitglied	Berechtigung	E-Mail
<input type="text" value="Mnr."/>	<input type="text" value="Mitglied eingeben"/>	<input type="button" value="Bitte auswählen"/>	<input type="text" value="E-Mail eingeben"/>

[Zurück](#) [Hinzufügen](#)

Abb. 19: Neue Sonderberechtigung

Es wird mit der Autovervollständigung im Mitgliedfeld und dem DropDownList der Berechtigungen eine neue Berechtigung für das eingegebene Mitglied erstellt. Auch hier gilt: bei leerem E-Mail Feld, wird die Standard E-Mail Adresse verwendet. Nach dem Hinzufügen gelangt man auf den Bildschirm mit den gelisteten Mitgliedern und deren Berechtigungen.

4.1.3 Liste der Geschäftsprozesse

Beim Klicken auf die Geschäftsprozesse gelangt man zur Übersicht aller in der Datenbank vorhandenen Geschäftsprozesse und deren zugehörigen Sonderberechtigung (Abb. 22). Die Geschäftsprozesse werden aus der Datenbank gelesen. Um einen neuen Geschäftsprozess anzulegen muss dies in der Datenbank selbst getätigkt werden, dann wird dieser in der Liste mit aufgelistet.

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

ID	Beschreibung	IT	RW	MV	ZE	Oea
1	Alle Mitglieder anzeigen	✓	✓	✓	✓	✓
2	Details einer Person anzeigen	✓	✓	✓	✓	✓
3	Mitglieder hinzufuegen	✓	✗	✓	✗	✗
4	asd	✓	✗	✓	✗	✗
5	Mitglieder loeschen	✓	✗	✓	✗	✗
6	Rolle eines Mitglieds zum Gesellschafter aendern	✓	✗	✓	✗	✗
7	Mitglied Administratorrechte hinzufuegen	✓	✗	✗	✗	✗
8	Kontenklassen hinzufuegen	✓	✓	✗	✗	✗
9	Kontenklassen bearbeiten	✓	✗	✗	✗	✗
11	Alle Konten anzeigen	✓	✓	✓	✓	✓
12	Details eines Kontos anzeigen	✓	✓	✓	✓	✓
13	Einlage/Entnahmekonten hinzufuegen	✓	✓	✗	✗	✗
14	Einlage/Entnahmekonten bearbeiten	✓	✓	✗	✗	✗
15	Zusatzentnahmekonten hinzufuegen	✓	✓	✗	✗	✗
17	Buergschaften anzeigen	✓	✓	✓	✓	✓
18	Buergschaften hinzufuegen	✓	✗	✗	✓	✗
19	Buergschaften bearbeiten	✓	✗	✗	✓	✗
20	Veranstaltung einsehen/bearbeiten	✓	✗	✓	✗	✓

[Zurück](#)

Abb. 20: Liste Geschäftsprozesse

So hat man auch den schnellen Überblick welche Sonderberechtigung derzeit welche Geschäftsprozesse vollziehen kann.

4.2 Attribute

Eine Sonderberechtigung enthält eine „ID“, die „Mnr“ für welches Mitglied diese gilt, eine E-Mail Adresse, unter welcher das Mitglied z.B. als Administrator erreicht werden kann, die Art der Berechtigung und die „SachPnr“ von wem diese Berechtigung angelegt bzw. zuletzt geändert wurde. Eine Sonderberechtigung kann nicht zweimal an dieselbe Person vergeben werden. Da ein Administrator alle Bereiche einsehen kann, wären für diesen die anderen Berechtigungen nicht von Nöten. Daher kann ein Mitglied mit der Sonderberechtigung „IT“ keine weiteren Sonderberechtigungen erhalten. Wenn ein Mitglied die Berechtigung „IT“ neu erhält, werden seine anderen Berechtigungen nötig. Eine Historisierung war nicht gewünscht, demnach werden Sonderberechtigungen bei einem Löschvorgang vollständig aus der Datenbank entfernt.

4.3 Sonderberechtigungs_Controller

4.3.1 listVeranstaltung

Die Methode lädt alle in der Datenbank aufgelisteten Geschäftsprozesse in ein Array, welches dann später in der entsprechenden View ausgelesen und dargestellt wird(Abb. 23).

4.3.2 createBerechtigungRollen

Hier wird zuallererst überprüft, ob eine Sonderberechtigung mit den übergebenen Parametern bereits in der Datenbank gelistet ist. Ist dies der Fall, so wird eine Fehlermeldung ausgegeben und nichts weiter unternommen. Falls keine solche Berechtigung vorhanden ist, wird geschaut ob eine E-Mail Adresse für die neue Berechtigung angegeben wurde, wenn ja wird diese für die Erstellung der Berechtigung verwendet. Wenn keine übergeben wurde, wird die Standard E-Mail des Mitglieds gesucht und für die Erstellung benutzt. Wenn auch diese nicht vorhanden ist, kann die neue Berechtigung nicht erstellt werden. Bei einer erfolgreichen Erstellung der Sonderberechtigung, wird man zu der Auflistung aller Sonderberechtigungen verwiesen und es wird eine Benachrichtigung angezeigt. Bei fehlerhafter Validierung verweilt man auf der Ansicht zum Erstellen einer neuen Sonderberechtigung und erhält ebenfalls eine Benachrichtigung über die fehlerhaften Parameter angezeigt.

4.3.3 deleteBerechtigungRollen

Es wird die Sonderberechtigung anhand der übergebenen ID gesucht und gelöscht. Anschließend wird auf der Seite verweilt und eine Benachrichtigung ausgegeben.

4.3.4 editRollen

Die Methode „editRollen“ wird aufgerufen, wenn man die Ansicht mit allen vergebenen Sonderberechtigungen aufruft. Es werden die Sonderberechtigungen und die dazugehörigen Personen gesucht und in einem Array abgelegt.

4.3.5 createSonderberechtigung

Diese Methode wird benötigt, wenn man die Ansicht zum Erstellen einer neuen Sonderberechtigung aufruft. Es werden die Personen geladen, welche für die Autovervollständigung gefragt sind. In diesem Fall können die Sonderberechtigungen nur Personen erhalten, welche noch nicht ausgetreten sind. D.h. eine Sonderberechtigung kann nur an aktive Mitglieder vergeben werden.

4.4 Views

4.4.1 editRollen

Diese View listet alle vergebenen Sonderberechtigungen auf, beim Hovern über die Abkürzungen werden die ausgeschriebenen Sonderberechtigungen angezeigt. Falls ein Mitglied keine bestimmte Sonderberechtigung besitzt, wird ein rotes „Nein“ in der Tabelle gelistet, das rote „Nein“ versendet beim Anklicken eine Form mit den Daten des Mitglieds zum Erstellen einer Sonderberechtigung an den Controller mit Hilfe der Javascriptfunktion „submit_form“:

```

function submit_form(pnr,berechtigung){
    if (confirm("Berechtigung hinzufügen?")){
        document.getElementById('pnr').value = pnr;
        document.getElementById('berechtigung').value = berechtigung;
        $('#editBerechtigungenRollen').submit()
    }
    else{
        alert("Berechtigung wurde nicht hinzugefügt")
    }
}

```

Abb. 21: Javascript Function „submit_form“

Es wird erst eine Bestätigung erwartet. Falls diese positiv ausfällt, wird die Form abgeschickt, falls nicht wird der „else“ Zweig ausgeführt und eine Meldung eingeblendet. Wenn das Mitglied eine Sonderberechtigung besitzt erscheint an der Stelle ein grünes „Ja“. Beim Anklicken ermöglicht dieses die Löschung der Berechtigung. Vorerst wird wie immer eine Bestätigung dieser Handlung erwartet. Achtung: Löscht ein Admin seine Berechtigungen fliegt er sofort aus der Sicht und wird als ein normales Mitglied ohne Sonderberechtigungen betrachtet. Dies kann dann nur durch Manipulation der Datenbank per Hand wieder rückgängig gemacht werden, oder ein weiterer Admin vergibt die Berechtigung erneut.

4.4.2 createSonderberechtigungen

Hier wird mit Hilfe einer Form, die Erstellung einer einzelnen Sonderberechtigung für ein einzelnes Mitglied, mit Hilfe der Auto vervollständigung, ermöglicht.

4.4.3 listGeschaeftsprozesse

Listet die in der Datenbank gefundenen Geschäftsprozesse auf. Wenn eine Sonderberechtigung einen Geschäftsprozess durchführen darf, wird dies als ein schwarzer Haken auf grünem Hintergrund angedeutet. Falls dieser Geschäftsprozess mit dieser Berechtigung nicht möglich ist, ist ein schwarzes Kreuz auf rotem Hintergrund abgebildet. Hier ein Beispiel für „IT“:

```
<%= prozess.IT ? '#fde5e5' %>> <i class=<%= prozess.IT ? "icon-ok" : "icon-remove" %>></i>
```

Abb. 22: Berechtigung Darstellung

Wenn IT diesen Prozess ausführen darf, werden die erste Farbe und das erste Icon gewählt, andernfalls wird das jeweils das Zweite gewählt.

4.5 Routes

Die neuen Routes, welche für die Sonderberechtigungen benötigt werden, dienen dem Aufruf der entsprechenden Methode aus dem „sonderberechtigung_controller“.

```

match '/Verwaltung/Rollen'          => 'sonderberechtigung#editRollen'
match '/Verwaltung/Geschaeftsprozesse' => 'sonderberechtigung#listGeschaeftsprozesse', :via => :GET
match '/Verwaltung/Rollen/NeueSonderberechtigung' => 'sonderberechtigung#createSonderberechtigung'
match '/Verwaltung/editRollen'       => 'sonderberechtigung#editBerechtigungenRollen', :via => :GET
match '/Verwaltung/editRollen'       => 'sonderberechtigung#createBerechtigungRollen', :via => :POST

```

Abb. 23: Routes Sonderberechtigungen

5 Bürgschaften (S. Stroh)

Ein Mitglied kann für ein anderes Mitglied bürgen. D.h. er verpflichtet sich bei geldlichen Schwierigkeiten des Bürgen, diesem finanziell unter die Arme zu greifen. Ein Mitglied kann einem anderen Mitglied nur eine Bürgschaft anbieten, jedoch kann ein Mitglied welches die Bürgschaft erhält, von mehreren eine sogenannte Teilbürgschaft erhalten, welche dann gemeinsam die Anleihe zum Teil oder vollständig abdecken. Die Bürgschaften sind historisiert, werden also immer nachvollziehbar hinterlegt. Es kann zu jedem Zeitpunkt nachvollzogen werden, wer die Bürgschaft geändert hat und was daran geändert wurde.

5.1 Ansichten

5.1.1 Aktuelle Bürgschaften

Die Ansicht Bürgschaften Aktuell zeigt die Bürgschaften, welche noch gültig sind und somit auch noch bearbeitet werden können. Die Bürgschaften sind in zwei Tabellen aufgeteilt. In der oberen Tabelle sind die Bürgschaften, welche die Person an andere Mitglieder vergeben hat gelistet. Die untere Tabelle beinhaltet die von anderen Mitgliedern erhaltenen Bürgschaften.

Kienle, Tassilo

The screenshot shows a software interface for managing pledges. At the top, there is a navigation bar with tabs: OZB-Konten, Details, Personaldaten, Kontaktdaten, Gesellschafterdaten, Bürgschaften (which is highlighted in blue), and Teilnahmen. Below the navigation bar, the title 'Bürgschaften Aktuell' is displayed. A table lists a single pledge made by 'Juhas, Anton' to another member. The table columns are: 'Vergeben an', 'ZE-Nr.', 'ZE-Betrag', 'Kurzbezeichnung', 'Betrag', 'Beginn', 'Ende', 'Laufzeit', and 'Ze-KtoNr'. The data row is: Juhas, Anton | D050922 | 6.000,00 | neuste | 100,00 | 05.03.2013 | 30.03.2013 | 4 | 10006. Below the table, a message states: 'Es sind derzeit keine erhaltenen Bürgschaften vorhanden.' At the bottom, there are three buttons: 'Zurück', 'Bürgschaften Historie' (highlighted in blue), and 'Bürgschaft hinzufügen'.

Abb. 24:Aktuelle Bürgschaften

5.1.2 Historisierte Bürgschaften

Die historisierten Bürgschaften, also die welche bereits gelöscht oder abgelaufen sind findet man, wenn man auf „*Bürgschaften Historie*“ klickt. Diese Schaltfläche ist nur für Mitglieder mit den entsprechenden Sonderberechtigungen sichtbar. Der Aufbau der Anzeige für die historisierten Bürgschaften, ist dem der aktuellen Bürgschaften nachempfunden. Die Anzeige ist ebenfalls in zwei Teile untergliedert, die obere Tabelle enthält die historisierten und aktuellen vergebenen Bürgschaften, die untere enthält die historisierten und aktuellen erhaltenen Bürgschaften. Beide Tabellen lassen sich nicht bearbeiten, da die Bürgschaften bereits abgeschlossen und Teil der Historisierung sind.

Kienle, Tassilo

OZB-Konten Details Personaldaten Kontaktdaten Gesellschafterdaten Bürgschaften Teilnahmen

Bürgschaften Historie

♦ Vergeben an	♦ ZE-Nr.	♦ ZE-Betrag	♦ Kurzbezeichnung	♦ Betrag	♦ Beginn	♦ Ende	♦ Laufzeit	♦ SachPnr
Stoll, Barbara	S071128	1.660,00	Teilbürgschaft	500,00	28.11.2007	31.12.2011	0	-
Juhas, Anton	D110125	2.300,00	-	100,00	05.03.2013	05.03.2013	2	13
Juhas, Anton	D050922	6.000,00	-	100,00	05.03.2013	09.03.2013	4	13
Juhas, Anton	D050922	6.000,00	neuste	100,00	05.03.2013	30.03.2013	4	13

♦ Erhalten von	♦ ZE-Nr.	♦ ZE-Betrag	♦ Kurzbezeichnung	♦ Betrag	♦ Beginn	♦ Ende	♦ Laufzeit	♦ SachPnr
Seidel, Wolfgang	D100430	4.000,00	test	100,00	10.03.2013	10.03.2013	0	13
Seidel, Wolfgang	D100331	1.200,00	test	100,00	10.03.2013	10.03.2013	0	13

Zurück

Abb. 25: Historisierte Bürgschaften

5.1.3 Neue Bürgschaft

Kienle, Tassilo

OZB-Konten Details Personaldaten Kontaktdaten Gesellschafterdaten Bürgschaften Teilnahmen

Neue Bürgschaft hinzufügen

Von:	<input type="text" value="Kienle, Tassilo"/>
An:	<input type="text" value="Mitglied auswählen"/>
ZE-Nr.:	<input type="text" value="Konto auswählen"/>
Beginn*:	<input type="text" value="10.03.2013"/>
Ende*:	<input type="text" value="10.03.2013"/>
Betrag*:	<input type="text" value="0"/>
Kurzbezeichnung:	<input type="text"/>

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

Abb. 26: Neue Bürgschaft

Um eine neue Bürgschaft anzulegen klickt man auf die Schaltfläche „*Bürgschaft hinzufügen*“. Eine neue Bürgschaft lässt sich an ein anderes Mitglied vergeben, der Name der Person, von der die Bürgschaft stammt bleibt unveränderbar. Es lässt sich auch keine Bürgschaft an sich selber vergeben, der eigene Name wird im DropDownList ausgegraut und nicht auswählbar dargestellt. Die Bürgschaft wird anhand der „*KtoNr*“ einem ZEKonto zugewiesen.

Die Auswahl der Konten erfolgt dynamisch. D.h. es wird erst eine Auflistung der Konten anzufinden sein, wenn ein Gläubiger gewählt worden ist. Die auswählbaren Konten sind dem Gläubiger zugehörig.

Der Betrag darf nicht kleiner als 0.01 sein. Bei einer erfolgreichen Speicherung gelangt man auf den Bildschirm mit den aufgelisteten aktuellen Bürgschaften. Nach dem erfolgreichen Hinzufügen der Bürgschaft wird man auf die Ansicht mit den aktuellen Bürgschaften weitergeleitet. Bei einer nicht erfolgreichen Speicherung verweilt man auf diesem Bildschirm und erhält die nicht passend ausgefüllten Felder angegeben.

5.1.4 Bürgschaft editieren

Um eine vorhandene Bürgschaft zu ändern oder diese zu löschen, reicht es auf die Spalte der gewünschten Bürgschaft zu klicken. So gelangt man in die Ansicht wo die Änderungen vorgenommen werden können. Die Schaltflächen für das Editieren und Löschen sind also ersetzt worden, durch die klickbare Spalte und die „*Löschen*“ Schaltfläche in der Editieren Ansicht. Beim Löschvorgang muss man diesen erst Bestätigen, bevor eine Bürgschaft „*gelöscht*“ wird. Um die Historisierung zu gewährleisten, werden die Bürgschaften nicht aus der Datenbank gelöscht. Es wird lediglich eine logische „*Lösung*“ aus der Ansicht „*Aktuelle Bürgschaften*“ vorgenommen. Mehr zur Historisierung im Teil 7.

The screenshot shows a web-based application interface for managing guarantees. At the top, there is a navigation bar with links: 'Besser ohne / Zins', 'Meine Konten', 'Meine Daten', 'Protokolle', 'oZBlick', 'Mitglieder', 'Weiteres', and 'Ausloggen'. Below the navigation bar, there is a secondary navigation menu with tabs: 'OZB-Konten', 'Details', 'Personaldaten', 'Kontaktdaten', 'Gesellschafterdaten', 'Bürgschaften' (which is currently selected and highlighted in blue), and 'Teilnahmen'. The main content area is titled 'Bürgschaft bearbeiten'. It contains several input fields for entering guarantee details:

Von:	Schmitz, Monika
An:	Schmitz, Christoph M.
ZE-Nr.:	D060321
Beginn*:	10.01.2007
Ende*:	31.12.2014
Betrag*:	€ 7070.0
Kurzbezeichnung:	Einzelbürgschaft

Below the input fields, a note states: "Felder die mit einem * markiert sind, müssen ausgefüllt werden." (Fields marked with an asterisk must be filled). At the bottom of the form, there are three buttons: 'Zurück' (Back), 'Löschen' (Delete), and 'Speichern' (Save).

Abb. 27:Bürgschaft editieren

5.2 Attribute

Eine neue Bürgschaft welche angelegt wird, erhält in die Spalte „*SachPnr*“ die Mitgliedsnummer des Mitglieds welches die Bürgschaft anlegt.“*SichAbDatum*“ und „*SichEndDatum*“ geben an, ab wann bis wann die Bürgschaft gültig ist. Ab dem Ende wird keine finanzielle Absicherung mehr garantiert. Die Attribute „*GueltigVon*“ und „*GueltigBis*“ werden für die Historisierung benötigt. Es sind nur die Bürgschaften aktuell und von Gültigkeit, dessen „*GueltigBis*“-Attribut nicht in der Vergangenheit liegt.

5.3 *Buergschaften_Controller*

5.3.1 index

Die Methode „*index*“ wird aufgerufen, wenn man auf den Reiter „*Bürgschaften*“ klickt. Es werden nur die aktuellen Bürgschaften geladen. Zwei Arrays werden angelegt, eines mit den vergebenen Bürgschaften, eines mit den erhaltenen Bürgschaften. Anschließend werden zu jeder Bürgschaft das entsprechende ZE-Konto und die beteiligte zweite Person gefunden. Die logisch zusammenhängenden Daten werden als gemeinsames Paket in ein Array gepackt. Dieses Array sieht dann folgendermaßen aus: [[ZE-Konto1, Person1], [ZE-Konto2, Person2],...]

5.3.2 edit und editB

Die beiden Methoden werden aufgerufen, je nachdem man welche Bürgschaft bearbeiten will. Sei es die Bürgschaft in der man als Bürge beteiligt ist, so wird „*editB*“ aufgerufen. Darin wird die entsprechende Bürgschaft geladen und bei Nichtexistenz wird ein Fehler gemeldet. Da alle Bürgschaften historisiert werden, muss darauf geachtet werden, dass die gesuchten Bürgschaften noch gültig sind.

5.3.3 create

Diese Methode wird benötigt um eine neue Bürgschaft anzulegen. Da ein Bürge demselben Gläubiger eine einzige Bürgschaft bieten kann, wird die neue Bürgschaft nur dann angelegt, wenn derzeit keine aktuelle Bürgschaft mit den beiden Personen, in den obigen Rollen, in der Datenbank besteht. Eine historisierte Bürgschaft, also eine die nicht mehr gültig ist, darf vorhanden sein. Beim Erstellen der Bürgschaft wird mit Hilfe der übergebenen „*KtoNr*“ des Bürgen, das dazugehörige ZEKonto gesucht und die „*ZENr*“ in die neue Bürgschaft übernommen. Es wird bei jeder Bürgschaft die „*SachPnr*“ des Erstellers protokolliert.

5.3.4 update

Die Methode „*update*“ sucht die zu aktualisierende Bürgschaft aus der Datenbank, dupliziert diese und ändert den Wert der Gültigkeit. Die neue Bürgschaft wird die alten Werte mit den neuen Änderungen erhalten, wohingegen die alte Bürgschaft in die Datenbank abgelegt wird, mit einem neuen Ende Datum. Die Mitgliedsnummer von der Person, welche die Änderungen vorgenommen hat, wird vermerkt.

5.3.5 delete

Die „*delete*“-Methode löscht die Bürgschaft nicht komplett, sondern ändert nur das Ende Datum, die Bürgschaft bleibt weiterhin in der Tabelle gelistet. Auch hier wird die Mitgliedsnummer der Person vermerkt, welche die Änderungen durchgeführt hat.

5.3.6 listHisto

Beim Aufruf der historisierten Bürgschaften, werden erst mal alle erhaltenen und vergebenen Bürgschaften geladen, welche bereits abgelaufen sind. D.h. diese sind nicht mehr aktuell und bereits ein Teil der Historisierung aller Bürgschaften. Diese dürfen nicht mehr verändert werden.

5.4 Routes

Die neuen Routes, welche für die Bürgschaften benötigt werden, dienen dem Aufruf der entsprechenden Methode aus dem „buergschaft_controller“.

```
match '/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/AendernB' => 'buergschaft#editB', :via => :GET
match '/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/AendernB' => 'buergschaft#updateB', :via => :PUT
match '/Verwaltung/OZBPerson/:Mnr/Buergschaften/:Pnr_B/LoeschenB' => 'buergschaft#deleteB', :via => :POST
match '/Verwaltung/OZBPerson/:Mnr/Buergschaften/AeltereBuergschaften' => 'buergschaft#listHisto'
```

Abb. 28: Routes Bürgschaften

5.5 Views

5.5.1 _form

Dies ist eine der wichtigsten Views der Bürgschaften, durch diese werden die Erstellung, Aktualisierung und Löschung einer Bürgschaft dargestellt. Die Bürgschaft kann vergeben worden sein. Dann gelangt man in den Bearbeitungsansicht und die Person welche man ausgewählt hat ist der Bürge. Die angeklickte Bürgschaft kann ebenfalls erhalten worden sein, dann ist die Person der Gläubiger. Dadurch, dass eine View für mehrere Anzeigen verwendet wird, ist sie auch etwas verworrender aufgebaut.

Es wird immer unterschieden, welche der drei Möglichkeiten hier angezeigt werden soll. Entweder dient sie als Ansicht für eine „neue“, eine „vergebene“ oder eine „erhaltene“ Bürgschaft.

Bei einer neuen ist @action = „new“, dann wird die Überschrift für die Erstellung angezeigt. Es wird der Name des Bürgen „deaktiviert“, sodass man von dem Mitglied nur von seinem Namen aus Bürgschaften vergeben kann und von Niemandem sonst.

Die Auswahl des ZE-Kontos wird nun über die „KtoNr“ getätigt(Abb. 25):

```
var ozbUZeKonto = new Array();
<% for konto in @zeKonten -%>
  ozbUZeKonto.push(new Array(<%= konto[0].Mnr %>, <%= konto[0].KtoNr %>));
<% end -%>

function updateZekonto() {
  pnr_g = document.getElementById('buergschaft_Mnr_G').value;
  ktonr_g = document.getElementById('buergschaft_ZENr').options;

  ktonr_g.length = 1;
  $.each(ozbUZeKonto, function (index, konto) {
    if (konto[0] == pnr_g) {
      ktonr_g[ktonr_g.length] = new Option(konto[1], konto[1]);
    }
  });
}
```

Abb. 29: Dynamische Collection Javascript

Mit der in Abb.25 dargestellten Javascriptfunktion wurde eine dynamische Auswahl der „OzbKonten“ ermöglicht. Wenn der „collection_select“ des Gläubigers verändert wird. Dann wird die Funktion „updateZekonto“ auf gerufen. Diese sucht aus den vorher erstellten, „Mnr“ und „KtoNr“, Pakten die dem ausgewählten Bürgen zugehörigen Kontonummern heraus und fügt diese dem „collection_select“ der KtoNr hinzu. Wenn ein Gläubiger kein OzbKonto besitzt kann keine neue Bürgschaft angelegt werden.

Die Variable @edit gibt an, ob die DropDownList-Auswahl editierbar sein soll oder nicht. Beim Aktualisieren einer erhaltenen Bürgschaft, ist der Name des Mitglieds unter dem Gläubiger aufgeführt. Wenn eine Bürgschaft an gelegt wurde, können die daran beteiligten Personen nicht mehr geändert werden. Man kann die Höhe der Summe, die Daten, das ZeKonto und die Kurzbezeichnung ändern. @selected gibt an welches Mitglied in der Auswahl der Personen vorausgewählt sein muss, da auch hier die Gläubiger und Bürgen je nach Art der Bürgschaft an beiden Positionen auftauchen könnten, muss darauf geachtet werden, ob es eine erhaltene oder vergebene Bürgschaft ist.

5.5.2 edit

Diese View wird durch Klicken auf eine erhaltene Bürgschaft aufgerufen. Weiterleitung an „form“. Wichtig für den Aufruf und Erstellung der Variablen im Controller.

5.5.3 editB

Diese View wird durch Klicken auf eine vergebene Bürgschaft aufgerufen. Weiterleitung an „form“. Wichtig für den Aufruf und Erstellung der Variablen im Controller.

5.5.4 listHisto

Es werden alle vergebenen und erhaltenen Bürgschaften, jeweils getrennt voneinander, aufgelistet. Die Bürgschaften können historisiert sein oder auch aktuell. @buergschaften wird zwei Mal durchlaufen, an der Stelle [0] befinden sich die vergebenen Bürgschaften. Ist diese leer wird ein Ersatztext anstelle der Tabelle angezeigt. Genauso verhält es sich für die Stelle [1] mit den erhaltenen Bürgschaften. Je nachdem ob @count eine 0 oder eine 1 enthält, befindet man sich beim ersten oder beim zweiten Durchlauf, danach werden auch die Texte ausgerichtet:

```
<%= @count == 0 ? "Vergeben an" : "Erhalten von" %></th>
```

Abb. 30: Textauswahl

Auch hier werden die logisch zusammenhängenden Objekte aus einem Datenpaket ausgelesen und angezeigt:

```
<% buergschaften.each do |buergschaft, zeKonto, person| %>
```

Abb. 31: Zusammenhängende Pakete

Falls beide Tabellen leer sind wird ebenfalls ein Ersatztext anstelle beider Tabellen erscheinen:

```
<h2 align="center"><small>Es sind derzeit keine historisierten Bürgschaften vorhanden.</small></h2>
```

Abb. 32: Text „keine Bürgschaften“

6 RoR unter Windows mit Xampp (S. Stroh)

Wie man RoR unter Windows installiert soll hier nicht erläutert werden, dies war nicht weiter schwierig. Allerdings gibt es einige Hürden welche in Verbindung mit Xampp auftauchen. Da hier mit Xampp gearbeitet wird, sind andere Server Gems nicht notwendig. Daher kann aus dem Gemfile die Zeile „gem 'unicorn'“ auskommentiert werden. Dieses Gem würde unter Windows sowieso nicht laufen, es ist für Unix ausgelegt.

6.1 Port besetzt

Es kann vorkommen dass irgendein anderes Programm die Ports 80 und 443, welche Xampp standardmäßig nutzt, belegt. Dann startet Xampp den Server nicht und gibt eine Fehlermeldung aus. Hier hilft es die belegten Ports auf freie zu ändern. Man öffnet die Datei „/apache/conf/httpd.conf“ und ändert die darin auftauchenden Port Angaben von 80 auf irgendeinen freien Port um, z.B. 1234.

Am besten geht dies durch Suchen und Ersetzen. Es werden dabei keine anderen Einstellungen geändert.

Die andere Datei welche geändert werden sollte ist „\apache\conf\extra\httpd-ssl.conf“. Darin sollten alle Portangaben von 443 auf einen freien Port geändert werden. Phpmyadmin wird dann unter der Adresse „127.0.0.1:3000/phpmyadmin“ nicht mehr erreichbar sein , daher muss „http://127.0.0.1:1234/phpmyadmin“ der Port verändert werden.

6.2 Localhost wird nicht gefunden

Es ist vorgekommen, dass man beim Starten der Applikation mit „localhost:3000“ den Fehler erhält, dass localhost nicht erreicht werden kann. Eine Abhilfe schafft hier der Aufruf der Applikation mit „127.0.0.1:3000“. Sollte allerdings die Datenbank nicht erreicht werden, so muss in der Datei „ozbapp\config\database.yml“ der host von „localhost“ auf „127.0.0.1“ geändert werden. Danach sollte alles gut laufen.

7 Historisierung (N. Usow)

7.1 Beschreibung

Um lückenlos Änderungen oder Einfügungen in der Applikation nachvollziehen zu können, wird in vielen geschäftskritischen Teilen der Applikation eine zeitliche Historisierung der Datensätze vorgenommen. Auch wird der Benutzer festgehalten, welcher die Änderungen an dem Datensatz durchgeführt hat. Das Kernkonzept der Historisierung basiert dabei auf dem Duplizieren von bestehenden Datensätzen, welchen dann Historisierungseigenschaften durch eine Aktualisierung zugewiesen werden.

Des Weiteren sei noch anzumerken, dass sich die Historisierung nicht destruktiv auf vorhandene Datensätze auswirkt d.h. es wird zuerst nur auf der angelegten Kopie des bestehenden Datensatzes gearbeitet, das Original bleibt somit vorerst unberührt. Dieses Verhalten wird mit Hilfe von Transaktionen unterstützend umgesetzt.

Durch die durchgeführte Datenbankmigration ist ein Großteil der bestehenden Historisierungslogik in den Models und die Triggerlogik in den Controllern unbrauchbar geworden. Die Gründe hierfür sind Änderungen an den bestehenden Primärschlüsseln, auf die die vorhergehende Historisierungslogik stark aufgebaut hatte. Zum Anderen sind auch in bestehenden Tabellen Historisierungsattribute durch die Migration neu mitaufgenommen worden. Durch diese Änderungen an der Datenbankstruktur ergaben sich sehr schwerwiegende Folgen für den bestehenden Code, da sehr große Teile der OZB-Applikation (sämtliche Controller mit Historisierungslogik inkl. der Models) umgeschrieben oder gänzlich um die Historisierungslogik neu erweitert werden mussten.

Meine zusätzliche Aufgabe bestand darin den Historisierungsvorgang auf die durchgeführte Datenbankmigration anzupassen und die neu aufgenommenen Attribute mit in den Historisierungsvorgang aufzunehmen und zugleich neue Historisierungslogik hinzuzufügen.

7.2 Attribute der Historisierung

Die Historisierung besteht aus der Sicht des Datenmodells aus den drei wesentlichen Attributen „GueltigVon“ und „GueltigBis“. Mit nur diesen Attributen ist es möglich den vollständigen Ablauf von getätigten Änderungen eines Benutzers zu loggen und den Verantwortlichen im Attribut „SachPnr“ festzuhalten.

Die beiden Attribute „GueltigVon“ und „GueltigBis“ sind vom SQL-Datentyp Datetime, welcher es ermöglicht neben des aktuellen Datums auch die Zeit bis auf die Sekunden genau zu speichern. Die Syntax des Datums ist dabei die folgende: „YYYY-MM-DD HH:MM:SS“.

„GueltigVon“ gibt dabei an, wann dieser Datensatz angelegt wurde, wo hingegen „GueltigBis“ entweder das bereits historisierte Ende angibt, oder ein nach oben offenes Datum darstellt.

Vor der Datenbankmigration stellten die beiden Attribute „Pnr“ und „GueltigVon“ einen zusammengesetzten Primärschlüssel dar. Der neue Primärschlüssel im produktiven System besteht jetzt aus den Attributen „Pnr“ und „GueltigVon“.

Das Attribut „SachPnr“ (Sachbearbeiter-Personennummer) gibt an von welchem aktuell eingeloggten Benutzer die Änderungen durchgeführt worden sind. Die „SachPnr“ ist vom SQL-Datentyp Integer, sie stellt eine Referenz auf eine gültige Person aus der Tabelle „person“ dar.

7.2.1 Vollständige Historisierung

In den folgenden geschäftskritischen Datenbank-Tabellen wird eine *vollständige* Historisierung, sprich eine Duplikierung des Original-Datensatzes, durchgeführt:

- adresse
- bankverbindung
- buergschaft
- eekonto
- foerdermitglied
- gesellschafter
- mitglied
- ozbkonto
- partner
- person
- student
- zekonto

7.2.2 Teilweise Historisierung

In der Tabelle „telefon“ und „ozbperson“ wird nur *teilweise*, ohne Erstellung eines Duplikates, historisiert. Hier wird lediglich ein SQL-Update auf einem bestehenden Datensatz durchgeführt, welches ein Attribut auf einen aktualisierten Wert setzt.

Bei Änderungen in der Tabelle „telefon“ wird nur die Nummer des Sachbearbeiters („SachPnr“) festgehalten. Telefon-Nummern stellen in der Regel keine sicherheitskritischen Daten dar, deswegen wird hier auf die Speicherung des „GueltigVon“ und „GueltigBis“-Zeitstempels komplett verzichtet.

In der Tabelle „ozbperson“ wird hingegen nur das Passwort-Änderungsdatum („PWAendDatum“) historisiert, dieses wird jeweils beim Ändern des Passworts auf das aktuelle Datum gesetzt.

7.3 Beispielvorgang einer Historisierung

Es wird nun der Vorgang einer *vollständigen* Historisierung am Beispiel der Kontaktdata einer Person in der Tabelle „adresse“ visualisiert. Das Verständnis des Historisierungsvorgangs ist essentiell für die darauffolgende Implementierung.

Dieser Vorgang findet auch analog für alle anderen historisierenden Tabellen statt.

Im folgenden Beispiel werden zwei aufeinanderfolgende Änderungen an der Straße und am Vermerk durchgeführt.

Der erste Datensatz stellt in diesem Beispiel das Original- bzw. den Ausgangsdatensatz für die erste Änderung dar. Änderungen werden durch eine blaue Umrandung dargestellt.

Pnr	GueltigVon	GueltigBis	Strasse	Nr	PLZ	Ort	SachPnr	Vermerk
13	2013-02-18 16:39:02	9999-12-31 23:59:59	Landhausstr.	92	70190	Stuttgart	NULL	

Abb. 33: Original-Datensatz in der Tabelle Adresse welcher verändert wird

Das Ändern der Straße bewirkt zunächst, dass der erste Datensatz dupliziert wird. Auf diesem Duplikat wird dann die Straße geändert und im Sinne der Historisierung die Nummer des aktuell eingeloggten Sachbearbeiters „SachPnr“ gesetzt. Fortführend wird noch im Duplikat das „GueltigVon“-Datum auf das aktuelle Datum gesetzt und das „GueltigBis“-Datum auf das Ende des Zeithorizonts aktualisiert. Hier wird präventiv und zum Zwecke der besseren Lokalisierung das

Datum „9999-12-31 23:59:59“ gesetzt. Dadurch ist erkennbar, dass dieser Datensatz der aktuell Gültige ist.

Für eine erfolgreiche Historisierung wird nun noch das Original als ungültig erklärt, dies geschieht durch das Aktualisieren des „GueltigBis“-Datums auf das aktuelle Datum. Es findet hier also eine Art des Dreiecktausches der Datumsangaben zwischen Original und Kopie statt.

Pnr	GueltigVon	GueltigBis	Strasse	Nr	PLZ	Ort	SachPnr	Vermerk
13	2013-02-18 16:39:02	2013-02-20 11:45:04	Landhausstr.	92	70190	Stuttgart	NULL	
13	2013-02-20 11:45:04	9999-12-31 23:59:59	Landhausstraße	92	70190	Stuttgart	13	

Abb. 34: Erster Datensatz wurde historisiert. Das Straße-Attribut wurde geändert.

Bei einer zweiten Änderung an den Kontaktdaten wird nun zuerst nach einem Datensatz mit einem gültigen „GueltigBis“-Datum (9999-12-31 23:59:59) gesucht, dieser wird dann für die zweite Änderungsiteration als Original- bzw. Ausgangsdatensatz deklariert.

Pnr	GueltigVon	GueltigBis	Strasse	Nr	PLZ	Ort	SachPnr	Vermerk
13	2013-02-18 16:39:02	2013-02-20 11:45:04	Landhausstr.	92	70190	Stuttgart	NULL	
13	2013-02-20 11:45:04	2013-02-20 11:45:54	Landhausstraße	92	70190	Stuttgart	13	
13	2013-02-20 11:45:54	9999-12-31 23:59:59	Landhausstraße	92	70190	Stuttgart	13	Erdgeschoss

Abb. 35: Zweiter Datensatz wurde historisiert. Das Vermerk-Attribut wurde geändert.

7.4 Implementierung der Historisierung

Die Implementierung unterscheidet sich in einigen Abläufen von dem oben beschriebenen Historisierungsvorgang, sie führt aber zum exakt gleichen Ergebnis. Dies ist damit zu erklären, da in der Praxis das bloße Duplizieren eines Datensatzes sofort zu einer Primärschlüsselverletzung führen würde. Die Attribute „GueltigVon“ und „Pnr“ sind zu einem zusammengesetzten Primärschlüssel definiert, im Falle eines Duplizierens würde es hier zum Fehler kommen, da der Datensatz eins-zu-eins doppelt vorliegen würde. Es wird daher zuerst eine Kopie des Original- bzw. Ausgangsdatensatzes aus der Datenbank geholt und nur im Speicher vorgehalten, erst nach der erfolgreichen Änderung und Speicherung des Originals wird auch die Kopie in historisierter Form persistiert.

Der Historisierungsvorgang läuft in der aufzurufenden Methode als eine Transaktion ab. Mögliche Fehler oder eine Unvollständigkeit des Objekts während des Speicherns, würden somit automatisch zu einem Rollback führen, welches auch dem Benutzer mit einer Fehlermeldung quittiert werden würde.

7.4.1 Automatische Historisierung

Um eine möglichst vollautomatische Historisierung zu erreichen, so dass der Programmierer diese nicht mehr selbst jedes Mal neu implementieren oder benutzen muss, stellt Rails in den Models verschiedene eventbasierte Callback-Methoden zur Verfügung. Diese Methoden benachrichtigen den Programmierer *bevor* eine bestimmte Aktion ausgeführt wird, sodass darauf gezielt reagiert werden kann.

So kann zum Beispiel abgefangen werden, ob ein Objekt in die Datenbank gespeichert wird oder die Speicherung erfolgreich abgeschlossen ist, damit lassen sich vor oder nach der Speicherung noch eigene Aktionen oder Logiken ausführen.

Die Callback-Methoden „before_update“ und „after_update“ sind dabei der Schlüssel für eine vollautomatische Historisierung. Diese Methoden wurden daher in allen Models implementiert die eine Historisierung der Daten vornehmen, siehe Tabellen unter „Vollständige Historisierung“.

Die Implementierung findet hier beispielhaft für das Adresse-Model statt, wie aber bereits erwähnt ist diese auch äquivalent für alle anderen historisierenden Models umgesetzt worden.

7.5 Implementierung

Die Implementierung fand in der folgenden Datei statt:

- app/models/adresse.rb

7.5.1 Callback: before_update

Diese Callback-Methode wird automatisch getriggert falls irgendwo im Controller versucht wird ein Model-Objekt (im diesen Fall ein Adresse-Objekt) zu speichern. Es ist daher hier genau der richtige Zeitpunkt eine Kopie aus der Datenbank zu erstellen und diese im Speicher in der Instanzvariable „@copy“ zu halten. Auf dieser Kopie können nun vor der getriggerten Speicherung des Originalobjekts die Historisierungseigenschaften wie „GueltigVon“, „GueltigBis“ und die „SachPnr“ gesetzt werden.

```
# Instanzvariable für die Kopie
@@copy = nil

# Callback-Methode. Wird automatisch getriggert bevor ein Adresse-Objekt in die Datenbank persistiert wird
before_update do
  # Hat das zu speichernde Adresse-Objekt ein gültiges "GueltigBis"-Datum
  if (self.GueltigBis == "9999-12-31 23:59:59")
    # Holt aus der DB das letzte gültige Adresse-Objekt zu einer Person
    @@copy      = Adresse.get(self.Pnr)
    # Dupliziert das Objekt aus der DB. Es wird hier noch nicht gespeichert!
    @@copy      = @@copy.dup
    @@copy.Pnr  = self.Pnr
    # Setzt in der Kopie das "GueltigVon" aus dem Original-Objekt
    @@copy.GueltigVon = self.GueltigVon
    # In der Kopie wird "GueltigBis" aufs jetzige Datum gesetzt
    @@copy.GueltigBis = Time.now

    # Im Original wird "GueltigVon" auf das jetzige Datum gesetzt
    self.GueltigVon = Time.now
    # Im Original wird "GueltigBis"-Datum auf ein Zukünftiges gesetzt
    self.GueltigBis = Time.zone.parse("9999-12-31 23:59:59")
  end
end
```

Abb. 36: Callback „before_update“

7.5.2 Callback: after_update

Diese Callback-Methode wird erst getriggert nach dem das veränderte Originalobjektpersistiert wurde, daher ist hier der optimale Zeitpunkt die Kopie nun auch in die Datenbank zu speichern. Zwecks der Abwärtskompatibilität wird bei der Speicherung der Kopie aber die Validierung ausgeschaltet, da es vorkommen kann, dass sich ehemals in der Datenbanktabelle NULL-Werte befunden haben, diese aber jetzt eventuell nicht mehr durch die Validierung kommen würden, aber dennoch trotzdem einen gültigen Datensatz aus historischer Sicht darstellen.

```
# Callback-Methode. Wird automatisch getriggert nachdem ein Adresse-Objekt in die Datenbank persistiert wurde
after_update do
  # Konnte in der "before_update"-Methode zuvor überhaupt eine gültige Kopie bezogen werden
  if !@@copy.nil?
    # Speichert die Kopie in die DB.
    # Validierungen werden zwecks Abwärtskompatibilität deaktiviert,
    # da in der DB auch alte NULL-Werte vorhanden sein könnten
    @@copy.save(:validation => false)

    # Vernichtet die Kopie. Ansonsten würde eine Endlosrekursion passieren,
    # da @@copy.save theoretisch wieder den Callback "before_update" triggern würde
    @@copy = nil
  end
end
```

Abb. 37: Callback: „after_update“

7.5.3 Adresse.get

Diese Methode stellt lediglich eine Hilfsmethode für den „*before_update*“-Callback dar. Es wird ein gültiges (9999-12-31 23:59:59) Adresse-Objekt zu einer Personennummer aus der Datenbank geholt, welches später programmatisch dupliziert werden kann.

```
# Bezieht ein gültiges Adresse-Objekt aus der DB für eine Person
def Adresse.get(pnr, date = Time.now)
  Adresse.where(:Pnr => pnr).where(["GueltigVon <= ?", date]).where(["GueltigBis > ?", date]).first
end
```

Abb. 38: Methode „Adresse.get“

8 Mitgliederverwaltung (N. Usow)

8.1 Beschreibung

Die Verwaltung der Mitglieder findet in der Regel über zwei Wege statt. Ein eingeloggter Benutzer ohne jegliche Sonderberechtigungen kann jederzeit seine Personaldaten und sein Passwort unter „Meine Daten“ ändern. Verfügt aber ein Benutzer darüber hinaus über erweiterte Sonderberechtigungen, so kann er auch die Daten anderer Benutzer einsehen und bei Bedarf abändern. Unter anderem ist dann auch die Bearbeitung der Kontaktdaten, Gesellschafterdaten (Rollenverwaltung) und der Sonderberechtigung möglich.

Des Weiteren lassen sich neue Mitglieder anlegen, Mitglieder historisiert löschen und alle bestehenden Mitglieder sortiert einsehen. Um ein effizienteres Suchen der Mitglieder zu ermöglichen, steht nun darüber hinaus ein interaktiver AJAX-Dialog zur Schnellsuche von Mitgliedern zur Verfügung, sodass das manuelle Suchen in Tabellen gänzlich Entfallen kann.

8.2 Serverseitige Sortierung der Mitglieder

Die bestehende Sortierung der Mitglieder erfolgte bisher über das Javascript jQuery-Plugin „tablesorter“. Diese Sortermethode inkl. des Plugins hatte aber den Nachteil, dass die Sortierung nur clientseitig für die *aktuell* angezeigte Seite angewendet wurde. Es fand also kein Sortieren *aller* Mitglieder statt, sondern nur der angezeigten Mitglieder auf der aktuellen Seite, somit ging auch beim Wechseln auf eine andere Seite die bestehende Sortierung verloren.

Eine Abhilfe schafft hier eine serverseitige Sortierung, die auch über mehrere Seiten hinweg ein konstantes und stabiles Sortieren zur Verfügung stellt. Dabei werden im Unterschied zum clientseitigen Sortieren, die Datensätze bereits auf dem Server mit einem angegebenen Seiten-Offset sortiert und schlussendlich in der OZB-Applikation ausgegeben. Die Steuerung des Offsets erfolgt dabei über den GET-Parameter „page“.

Mitglied-Nr.	Name, Vorname	Rolle	Aufnahmedatum	Austrittsdatum
5	Seidel, Wolfgang	Gesellschafter	21.01.2005	
6	Juhas, Anton	Gesellschafter	21.01.2005	
7	Ochs, Armin	Gesellschafter	21.01.2005	
8	Peckmann, Ulrich	Gesellschafter	21.01.2005	
9	Sander, Uwe	Gesellschafter	21.01.2005	31.12.2010

Abb. 39: Stabile serverseitige Sortierung nach Name

8.2.1 Parameter zur Steuerung der Sortierung

Da die Sortierung nun serverseitig verläuft muss das Verhalten der Sortierung auf den Server mit sogenannten GET-Parametern gesteuert bzw. beeinflusst werden. Die Steuerparameter werden dabei an die bestehende URL angehängt. Somit kann dann der Controller anhand der übergebenen GET-Parameter auf eine bestimmte Aktion reagieren und auf den SQL-Server eine Operation triggern.

Die Wertepaare von GET-Parametern werden gemäß dem HTTP-Standard durch das &-Zeichen voneinander getrennt. Die angehängten Daten sind somit ein Teil der URL und können dadurch beim Speichern oder bei einer Weitergabe des Links weiterhin persistent erhalten bleiben.

Ein Wertepaar eines GET-Parameters hat dabei die folgende Syntax:

bezeichnung1=wert1&bezeichnung2=wert2

Im folgenden Beispiel findet eine Parametrisierung für eine absteigende Sortierung nach dem Namen auf der zweiten Seite statt.



Abb. 40: Zur Sortierung werden GET-Parameter an die URL angehängt, siehe rote Umrandung

Um eine konstante Sortierung über mehrere Seiten hinweg zu berücksichtigen, werden des Weiteren auch in die Pagination (Seitenumschaltung) die Parameter übergeben. Beim Wechseln auf eine andere Seite werden somit auch die Parameter automatisch angepasst und weiter an die nächste Seite durchgereicht. Dies lässt sich durch ein Hovern mit der Maus über die Pagination überprüfen.

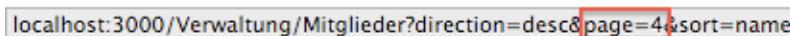


Abb. 41: Durchreichung der Parameter in die Pagination, siehe rote Umrandung

8.2.2 GET-Sortierparameter

Zur Sortierung der Mitgliederverwaltung kommen dabei die folgenden GET-Parameter inklusive der möglichen Werte zum Einsatz:

- *direction*
 - Mögliche Werte sind „*asc*“ und „*desc*“. „*asc*“ stellt dabei die aufsteigende und „*desc*“ absteigende Sortierung dar.
- *page*
 - Mögliche Werte sind positive ganze Zahlen. „*page*“ gibt an auf welcher Seite die Sortierung vorgenommen wird, anhand der Seitenzahl lässt sich der SQL-Offset berechnen (*page* * *anzuzeigende Mitglieder pro Seite*). Dadurch kann gezielt nur ein bestimmter Teilausschnitt aus der Datenbank selektiert werden.
- *sort*
 - Mögliche Werte sind „*mnr*“, „*name*“, „*rolle*“, „*aufnahmedatum*“ und „*austrittsdatum*“. „*sort*“ steuert nach welchem Attribut in der Datenbank sortiert werden soll.

8.2.3 Implementierung

Die Implementierung der Sortierung ist abstrakt gehalten, somit können zukünftig andere Programmierer diese Funktionen mit ihren eigenen Parametern wiederverwenden.

Die Implementierung fand jeweils in den folgenden Dateien statt:

- app/controllers/verwaltung_controller.rb
- app/helpers/application_helper.rb
- app/views/verwaltung/listOZBPersonen.html.erb

Im Controller sind zwei Helper-Methoden „sort_column“ und „sort_direction“ angelegt worden. Diese sorgen dafür, dass in der View entweder nach den Standardparametern oder nach einem selbst festgelegtem Sortierparameter sortiert wird. Falls keine Parameter angegeben sind, wird dabei „asc“ für eine aufsteigende Sortierrichtung und „mnr“ (Mitgliednummer) für die zu sortierende Spalte in der Datenbank als Standardparameter benutzt.

```
# Zugriffsfreigabe der Methoden aus der View
helper_method :sort_column, :sort_direction, :rollen_bezeichnung

private
# Default Sortierspalte.
# Ist keine vorhanden wird standardmäßig nach der Mitgliednummer sortiert
def sort_column
  params[:sort] || "mnr"
end

# Default Sortierrichtung
def sort_direction
  # Ist keine Sortierrichtung vorhanden wird nach aufsteigend (asc) sortiert
  %w[asc desc].include?(params[:direction]) ? params[:direction] : "asc"
end
```

Abb. 42: sort_column und sort_direction in verwaltung_controller.rb (Controller)

Des Weiteren werden der Methode „listOZBPersonen“, die zum Anzeigen der Mitglieder verantwortlich ist, die Sortierspalte und –richtung als Parameter übergeben. Somit können die Mitglieder schon direkt aus der Datenbank mit den übergebenen Sortierparametern geholt. Die Berechnung des Offsets für die Datenbank wird automatisch von der hinzugefügten Methode „paginate“ realisiert, diese erwartet als Eingabeparameter nur die aktuelle Seite und die Anzahl der Mitglieder, in diesem Fall 25, die pro Seite angezeigt werden sollen.

```
# Auflistung aller Mitglieder
def listOZBPersonen
  if currentUserAdmin?
    @OZBPersonen = OZBPerson
      .joins(:Person)
      .order(sort_column + " " + sort_direction) # Übergebene Sortierspalte und -richtung
      .where("GueltigBis LIKE '%9999%'")
      .paginate(:page => params[:page], :per_page => 25) # Aktuelle Seite und anzuzeigende Mitglieder
  else
    redirect_to "/MeineKonten"
  end
end
```

Abb. 43: listOZBPersonen in verwaltung_controller.rb (Controller)

Um einen möglichst redundanzfreien Code zu schreiben und eine allgemein gültige Methode zur Verfügung zu stellen, wird außerdem eine globale „sortable“ Helper-Methode angelegt. Diese Methode kann so theoretisch aus jeder View heraus verwendet werden. Sie generiert einen HTML-Link mit den übergebenen Sortierparametern und gibt den Link als einen String zurück.

```
# Spaltensortierung in einer Tabelle. NU
def sortable(column, title = nil)
  # Verschönert den Link-Namen
  title ||= column.titleize

  # Sortierrichtung. Falls die gleiche Spalte nochmals sortiert wird, dann wird auf die andere Sortierrichtung umgeschaltet
  # Standard-Sortierrichtung ist ASC
  direction = column == sort_column && sort_direction == "asc" ? "desc" : "asc"

  # Erstellt ein a-Link mit Spaltenparameter und Sortierrichtung und gibt diesen zurück
  link_to title, :sort => column, :direction => direction
end
```

Abb. 44: sortable in application_helper.rb (Helper)

Durch die zuvor angelegte „sortable“ Helper-Methode kann nun in der View die Ausgabe des Links erfolgen. Dazu wird als erster Parameter die zu sortierende Spalte in der Datenbank angegeben, und als zweiter Parameter die Bezeichnung des Links sowie er auch in der View dargestellt wird.

```
<table id="personen" class="table table-bordered table-striped">
<thead>
<tr>
  <th><%= sortable "mnr", "Mitglied-Nr." %></th>
  <th><%= sortable "name", "Name, Vorname" %></th>
  <th><%= sortable "rolle", "Rolle" %></th>
  <th><%= sortable "aufnahmedatum", "Aufnahmedatum" %></th>
</tr>
</thead>
```

Abb. 45: Ausgabe von sortable in listOZBPersonen.html.erb (View)

8.3 Schnellsuche

Die vorhergehende Suche nach Mitgliedern war bisher nur über das Blättern der einzelnen Seiten in der Mitgliederübersicht möglich, gerade bei einer Vielzahl von Mitgliedern stößt dieses manuelle Durchklicken und Abgleichen schnell an seine Grenzen. Es bestand also keine benutzerfreundliche Variante und effiziente Möglichkeit Mitglieder anhand des Vor- und Nachnamens im System aufzufinden.

Die nun neuimplementierte Suche setzt dabei auf interaktives AJAX. Dabei werden schon beim Tippen des Vor- oder Nachnamens Suchvorschläge von im System eingetragenen Benutzern aufgeführt. Das interaktive Einblenden von neu gefundenen Mitgliedern, sowie das Übertragen der Tastenanschläge, wird mit Hilfe des jQuery-UI Frameworks und dem dazu gehörigen Modul „autocomplete“ umgesetzt.

Des Weiteren ist es das Ziel eine möglichst allgemeine und abstrakte Referenzimplementierung einer Schnellsuche zu erstellen, damit diese auch in weiteren Teilen der Applikation ohne großen Aufwand übernommen und angewendet werden kann.

Das Tippen der Zeichenfolge „schm“ liefert alle Benutzer die irgendwo im Vor- oder Nachnamen die gleiche Zeichenfolge aufweisen z.B. „Schmidt“. Beim Klick auf ein Suchergebnis wird man zur Detailansicht der Person weitergeleitet wo man dann weitere personenbezogene Daten oder die Kontenverwaltung einsehen kann.

The screenshot shows a user interface for searching members. At the top, there are four buttons: "Alle Mitglieder anzeigen", "Schnellsuche" (which is highlighted in blue), "Neues Mitglied hinzufügen", and "Sonderberechtigungen". Below these buttons is a text input field containing the text "schm". A dropdown menu lists five search results: "Schmitz, Christoph M.", "Schmitz, Monika" (which is highlighted with a grey background), "Schmitz, Anna", "Schmidtke, Alexander", and "Adam-Schmidtke, Kerstin".

Abb. 46: Benutzer die irgendwo im Vor- oder Nachnamen die Zeichenfolge „schm“ enthalten

8.3.1 Implementierung

Die Implementierung fand jeweils in den folgenden Dateien statt:

- app/controller/verwaltung_controller.rb
- app/views/verwaltung/searchOZBPerson.html.erb

Die Such-Methode im Controller hat nur eine primäre Aufgabe: Sie liefert alle Personen, ohne doppelte Einträge, aus der Datenbank zurück und speichert diese in die öffentliche Variable „@DistinctPersonen“.

Diese Methode wird nur einmalig beim Laden der Seite getriggert, die Mitglieder-Objekte werden danach dauerhaft ins JSON-Format (JavaScript Object Notation) überführt. Dort stehen sie dann für jQuery für die weitere Verarbeitung zur Verfügung.

```
# Holt alle Personen aus der DB, ohne doppelte Einträge
def searchOZBPerson
  @DistinctPersonen = Person.find(:all, :select => "DISTINCT Pnr, Name, Vorname")
end
```

Abb. 47: searchOZBPerson in verwaltung_controller.rb (Controller)

Die restliche Implementierung findet nun in Javascript in der View statt. Dafür wird zuerst ein HTML-Inputfeld angelegt und zwingend mit der ID „suchFeld“ ausgezeichnet. Anhand der ID kann jQuery später das Suchfeld ansprechen um die Tastaturanschläge abzufangen und um die Ergebnisse zu präsentieren.

```
<p>Bitte geben Sie die zusuchende Person ein:</p>
<div>
  <%= text_field_tag "", "", :id => "suchFeld" %>
</div>
```

Abb. 48: Suchfeld in searchOZBPerson.html.erb (View)

Die zuvor abgefragten Personen werden nun in die JSON-Notation übertragen, so dass die Personen auch von Javascript bzw. jQuery gelesen werden können.

```
// Ruby Objekte werden ins JSON-Format überführt
var names = [
  => @DistinctPersonen.each do |person| %>
  {
    value: "<%= person.Pnr.to_s %>",
    label: "<%= person.Name.to_s + ', ' + person.Vorname.to_s %>"
  },
  => end %>
];

```

Abb. 49: Überführung der Ruby-Objekte nach JSON in searchOZBPerson.html.erb (View)

Um das vorher festgelegte Suchfeld zu einem Autocomplete-Feld zu transformieren wird auf dessen ID die „autocomplete“-Methode aufgerufen. Zusätzlich wird die Quelle und die kürzeste Zeichenkettenlänge festgelegt. In diesem Fall reicht ein Buchstabe als Eingabe aus, um eine Suchoperation zu triggern.

Außerdem wird das Verhalten bei einem Klick auf einen Eintrag definiert, hier passiert bei einem Klick auf einen Suchtreffer eine Weiterleitung zu der gewünschten Personenseite. An dieser Stelle können andere Programmierer ihr eigenes gewünschtes Programmverhalten implementieren.

```
// Das Suchfeld wird mit der Autocomplete-Funktion erweitert und bekommt die JSON-Personen übergeben
$("#suchFeld").autocomplete({
  minLength: 1,
  source: names,
  select: function(event, ui) {
    $("#suchFeld").val(ui.item.label);
    window.location.href = "/Verwaltung/OZBPerson/" + ui.item.value + "/Konten";
    return false;
  }
});
```

Abb. 50: HTML-Inputfeld wird zur Suche transformiert in searchOZBPerson.html.erb (View)

In der „data“-Methode wird das Verhalten beim Auffinden einer Person bestimmt, hier wird ein neues Listenelement mit dem gefundenen Vor- und Nachnamen und einem Link zurückgegebenen. Dieser wird dann anschließend an die bestehende Liste mit den Suchergebnissen hinten angehängt.

```
// Hängt neue Suchergebnisse in Form eines HTML-Listenobjekts an den Dialog an
.data("autocomplete")._renderItem = function(ul, item) {
  return $(<li></li>)
    .data("item.autocomplete", item)
    .append("<a href='/Verwaltung/OZBPerson/" + item.value + "/Konten'>" + item.label + "</a>")
    .appendTo(ul);
};
```

Abb. 51: Suchergebnisse werden als eine Liste präsentiert in searchOZBPerson.html.erb (View)

8.3.2 Wiederverwendung der Implementierung

Durch die abstrakte Implementierung konnte der Code mit minimalsten Änderungen auch erfolgreich in der Veranstaltungsverwaltung von Herrn Stroh eingesetzt werden.

Mnr.	Mitglied	Anwesenheit
61	peter	u
	Hamann, Peter	
Gesellsch.	Herrmann, Peter-Julius	04.2012, BZ-Stuttgart Ost
	Kleinert, Peter	

Abb. 52: Weiterer Einsatz der Schnellsuche in der Veranstaltungsverwaltung

8.4 Benutzerbezogene Berechtigungen

8.4.1 Beschreibung

Neben der globalen Rechtevergabe existiert noch zusätzlich eine benutzerbezogene Rechtevergabe. Diese ermöglicht es für einen ausgewählten Benutzer verschiedene Sonderberechtigungen zu setzen. Der administrative Einflussbereich eines Benutzers kann somit erweitert werden. Die Art der Berechtigung wurde dabei als ein Geschäftsprozess für eine definierte Benutzergruppe vorab festgelegt.

Die Berechtigungen können für einen bestimmten Benutzer dynamisch gesetzt oder gelöscht werden.

The screenshot shows a user profile page with tabs at the top: OZB-Konten, Details, Personaldaten, Kontaktdaten, Gesellschafterdaten (which is selected), Bürgschaften, and Teilnahmen. On the left, under 'Rolle bearbeiten', there are fields for 'Rolle' (set to 'Gesellschafter'), 'FA Steuernummer' (redacted), 'FA Lfd. Nr.' (14), and 'Wohnsitzfinanzamt' (redacted). On the right, under 'Aktuelle Berechtigungen', there is a table with columns 'Berechtigung', 'Email', and 'Aktionen'. It lists 'Öffentlichkeitsverwaltung (OeA)' and 'Projekteverwaltung (ZE)'. Each row has a red 'Löschen' (Delete) button. A blue 'Berechtigung hinzufügen' (Add permission) button is at the bottom.

Abb. 53: Benutzer mit den Rechten Öffentlichkeits- und Projekteverwaltung

Die Geschäftsprozesse inkl. der Gruppenberechtigungen liegen in der Datenbank in einer Art Wahrheitstabelle vor.

Einem Geschäftsprozess kann dabei eine Gruppenberechtigung in Form von einer „0“ oder „1“ zugewiesen werden. Eine „1“ definiert dabei ein aktives bzw. erlaubtes Recht, die „0“ hingegen ein Verbot.

ID	Beschreibung	IT	MV	RW	ZE	OeA
1	Alle Mitglieder anzeigen	1	1	1	1	1
2	Details einer Person anzeigen	1	1	1	1	1
3	Mitglieder hinzufuegen	1	1	0	0	0
5	Mitglieder loeschen	1	1	0	0	0
6	Rolle eines Mitglieds zum Gesellschafter aendern	1	1	0	0	0
7	Mitglied Administratorrechte hinzufuegen	1	0	0	0	0

Abb. 54: Auszug aus den Geschäftsprozessen inkl. der Gruppenberechtigungen

Es existieren insgesamt fünf Gruppenberechtigungen:

- IT – Administrationsrecht
- MV – Mitgliederverwaltung
- RW – Finanzenverwaltung
- ZE – Projekteverwaltung
- OeA – Öffentlichkeitsverwaltung

8.4.2 Dynamische Rechteanzeige für „Berechtigung hinzufügen“

Um das Hinzufügen einer Berechtigung transparenter und ergonomisch einfacher zu gestalten, werden die Rechte für eine Sonderberechtigung angezeigt, somit erspart man sich das manuelle Durchforsten in der Datenbank und hat sofort einen Überblick der Möglichkeiten. Bei der Auswahl eines Rechts aus der Berechtigungs-Auswahlbox werden und dynamisch die erlaubten Aktionen (Geschäftsprozesse) für diese Berechtigung aufgeführt. Somit erhält der Anwender eine bequeme Möglichkeit zu einer Gruppe die möglichen Rechte einzusehen und entsprechend zu verteilen. Des Weiteren soll verhindert werden, dass einem Mitglied mehrfach das selbe Recht zugeteilt wird. Dies wird jetzt vor dem Eintragen einer Sonderberechtigung mit der Datenbank abgeglichen und im Fehlerfall dem Anwender mit einer Meldung quittiert.

The screenshot shows a user interface for managing permissions. On the left, there's a form with fields for 'Berechtigung' (Permission) and 'Email'. The 'Berechtigung' field has a dropdown menu showing 'Finanzenverwaltung'. The 'Email' field contains 'test@user.de'. On the right, a list titled 'Rechte für Finanzenverwaltung' is shown as an HTML list:

- Alle Mitglieder anzeigen
- Details einer Person anzeigen
- Kontenklassen hinzufügen
- Alle Konten anzeigen
- Details eines Kontos anzeigen
- Einlage/Entnahmekonten hinzufügen
- Einlage/Entnahmekonten bearbeiten
- Zusatzentnahmekonten hinzufügen
- Bürgschaften anzeigen

Abb. 55: Dem Benutzer wird das Recht Finanzenverwaltung vergeben. Rechts werden die möglichen Rechte als HTML-Liste für diese Gruppe angezeigt



Abb. 56: Falls die selbe Berechtigung bereits im System vorhanden ist

8.4.3 Implementierung der dynamischen Rechteanzeige

Die Implementierung der dynamischen Rechteanzeige fand jeweils in den folgenden Dateien statt:

- app/controller/verwaltung_controller.rb
- app/views/verwaltung/editBerechtigungen.html.erb
- app/assets/stylesheets/application.css

Bevor eine Sonderberechtigung ins System eingepflegt wird, wird zuerst in der Datenbank geschaut, ob es diese Berechtigung für diesen Benutzer und der Email bereits gibt. Ist die Sonderberechtigung noch nicht vorhanden so wird sie dem Benutzer hinzugefügt.

```
# Transaktion
ActiveRecord::Base.transaction do
  # Evt. bestehende Berechtigung holen
  @sonderberechtigung = Sonderberechtigung.where(
    "Mnr = ? AND Email = ? AND Berechtigung = ?",
    params[:Mnr],
    params[:email],
    params[:berechtigung]
  )

  # Wenn Berechtigung noch nicht existiert
  if @sonderberechtigung.empty?
    @OZBPerson = OZBPerson.find(params[:Mnr])
    @Person    = Person.get(@OZBPerson.Mnr)
  end
end
```

Abb. 57: Existiert das Recht schon in verwaltung_controller.rb (Controller)

Das Erzeugen der HTML-Liste für die Rechteanzeige wird in der „display_berechtigungen_for“-Methode ausgelöst. Diese erwartet als Eingabe-Parameter die Liste aller Geschäftsprozesse und die Gruppe aus der Berechtigungen-Auswahlbox z.B. Finanzenverwaltung. Basierend darauf wird eine HTML-Liste generiert und schlussendlich zurückgegeben.

```
# Generiert für eine Benutzergruppe die Berechtigungs-Liste
def display_berechtigungen_for (geschaeftsprozesse, gruppe)
  if !geschaeftsprozesse.nil?
    html = "<ul>"
    geschaeftsprozesse.each do |gp|
      if gp.send(gruppe)
        html << "<li>" << gp.Beschreibung << "</li>"
      end
    end
    html << "</ul>"
    html.html_safe # Das HTML soll nicht maskiert werden
  end
end
```

Abb. 58: Generiert eine HTML-Liste zu der Berechtigungsgruppe in editBerechtigungen.html.erb (View)

Die oben implementierte Funktion kann dann für alle Berechtigungsgruppen aufgerufen werden, sie erzeugt jeweils fünf HTML-Listen mit allen Geschäftsprozessen die zu der jeweiligen Gruppe passen.

```
<div class="span6" id="beschreibung-berechtigung">
  <h4></h4>
  <div id="MV">&lt;= display_berechtigungen_for(@Geschaeftsprozesse, "MV") %></div>
  <div id="IT">&lt;= display_berechtigungen_for(@Geschaeftsprozesse, "IT") %></div>
  <div id="RW">&lt;= display_berechtigungen_for(@Geschaeftsprozesse, "RW") %></div>
  <div id="ZE">&lt;= display_berechtigungen_for(@Geschaeftsprozesse, "ZE") %></div>
  <div id="OeA">&lt;= display_berechtigungen_for(@Geschaeftsprozesse, "OeA") %></div>
</div>
```

Abb. 59: Gibt die generierten HTML-Listen aus in editBerechtigungen.html.erb (View)

Um die initiale Anzeige aller HTML-Listen zu unterdrücken wird der umschließende Kontainer (DIV-Tag) der Listen im CSS-Stylesheet als nicht sichtbar markiert.

```
#beschreibung-berechtigung div {
  display: none;
}
```

Abb. 60: application.css (CSS-Stylesheet). Markiert den Berechtigungs-Kontainer initial als nicht sichtbar

Die Umschaltlogik der HTML-Listen wird in Javascript bzw. jQuery implementiert. Dabei wird zuerst eventbasiert geschaut ob sich ein Eintrag in der Auswahlbox geändert hat. Ist dies der Fall wird die bestehende Liste mit der „*hide*“-Methode versteckt und die neue Liste, die vorher mit CSS als nicht sichtbar markiert wurde, nun als sichtbar mit der „*show*“-Methode ausgezeichnet. Bei jeder Änderung in der Auswahlbox wird somit zuerst eine Aus- und dann Einblendung durchgeführt. Wird in der Liste in den Initialzustand gewechselt (Auswahlbox: „Bitte auswählen“), so wird die komplette Liste wieder ausgeblendet.

```
$(document).ready(function() {
    // Gab es eine Änderung in der Auswahlbox
    $("#berechtigung").change(function(){
        // Entferne die alte HTML-Liste
        $("#beschreibung-berechtigung div").hide();

        // Hole Wert für neu anzugezendes Recht
        var value = $("#berechtigung").val();
        $("#"+ value).show();

        // Gültiges Recht gewählt
        if (value != ""){
            var beschreibung = $("#berechtigung :selected").text();
            $("#beschreibung-berechtigung h4").show().text("Rechte für " + beschreibung);
        }
        // "Bitte auswählen"
        else{
            $("#beschreibung-berechtigung h4").hide();
        }
    });
});
```

Abb. 61: Ein- und Ausblenden der HTML-Liste mit Berechtigungen in editBerechtigungen.html.erb (View)

8.5 Mitglied hinzufügen

Das Hinzufügen von neuen Mitgliedern war bisher nicht komfortabel gelöst, da die Eingaben aller personenbezogenen Daten über mehrere unabhängige Tabs verteilt waren. Man musste sich somit durch mehrere Tabs durchklicken um einen Benutzer anzulegen. So mussten auch die Personal- und Kontaktdaten, die Rolle und das Passwort unabhängig voneinander angegeben werden, obwohl sie in ihrer Gesamtheit einen kompletten und eigenständigen Geschäftsprozess darstellen. Dies war insofern problematisch, da die Daten auch über mehrere verschiedene Datenbank-Tabellen hinweg gespeichert werden. Diese mussten somit bis zu einer erfolgreichen Eingabe und Validierung in einer Art Zwischenspeicher (Session bzw. Cookie) gehalten werden, da sonst beim Wechseln auf einen anderen Eingabe-Tab die bisher getätigten Daten verloren gegangen wären. Des Weiteren war es bisher nicht möglich alle Validierungsfehler der Eingabefelder anzuzeigen, sondern nur die des aktuellen Tabs. Aus ergonomischer Sicht war diese Art der Dateneingabe nicht besonders benutzerfreundlich und auch technisch sehr komplex umgesetzt. Um dem Benutzer eine vereinfachte und übersichtliche Form der Dateneingabe zu ermöglichen wurde daher beschlossen alle Eingabefelder auf nur eine Seite zusammenzuführen. Dadurch soll es nun dem Benutzer möglich sein alle Felder auf einmal zu überblicken und mögliche Fehler bei der Eingabe als eine Gesamtheit angezeigt zu bekommen.

The screenshot shows a user registration form with four tabs at the top: 'Personaldaten', 'Kontaktdaten', 'Rolle', and 'Logindaten'. The 'Personaldaten' tab is currently active. Below it, there are three input fields: 'Name' (Hans), 'Vorname' (Dampf), and 'Geburtsdatum' (01.02.1987).

Personaldaten	Kontaktdaten	Rolle	Logindaten
*Name: <input type="text" value="Hans"/>			
*Vorname: <input type="text" value="Dampf"/>			
Geburtsdatum: <input type="text" value="01.02.1987"/>			

Abb. 62: Ehemalige Form der Dateneingabe. Die Eingabefelder waren unabhängig über mehrere Tabs verteilt

Das neue Layout sieht dabei vor die einzugebenden Daten logisch in zwei Spalten zu präsentieren. Die Unterteilung geschieht dabei zuerst zeilenweise in Personal- und Kontaktdaten, und danach in Rolle und Logindaten. Zum Anderem können nun im Falle eines Fehlers Korrektureingaben übersichtlicher und schneller getätigter werden.

Personaldaten		Kontaktdaten	
*Name:	Hans	*Email:	<input type="text" value="hans.dampf@gmail.com"/>
*Vorname:	Dampf	Tel.:	<input type="text"/>
Geburtsdatum:	10.02.1983	Mobil:	01621212399
*Antragsdatum:	07.02.2013	Fax:	<input type="text"/>
Aufnahmedatum:	15.02.2013	Straße:	Wasserstr.
Austrittsdatum:	29.02.2020	Hausnummer:	1337
		PLZ:	76149
		Ort:	Karlsruhe
		Vermerk:	Kellergeschoss
Rolle		Logindaten	
*Rolle:	<input type="text" value="Mitglied"/>	*Passwort:	<input type="password" value="*****"/>
RV Datum:	<input type="text" value="14.02.2013"/>		

Abb. 63: Neue übersichtliche Form der Dateneingabe

8.5.1 Implementierung

Die Implementierung fand in der folgenden Datei statt:

- app/views/verwaltung/newOZBPerson.html.erb

Am Controller der View mussten keine Änderungen gemacht werden, da die Hintergrundlogik zum Einfügen der Daten nicht verändert werden musste. Die Implementierung bzw. Zusammenführung fand nur in der View statt. Der Code der Zusammenführung der Felder ist über mehrere Seiten verteilt, es wird im Folgenden daher nur ein kleines zusammengefasstes Beispiel wiedergegeben.

```
<!-- Kontainer für Personal- und Kontaktdaten -->
<div class="row-fluid">
  <!-- Personaldaten-Spalte -->
  <div class="span6">
    <h4>Personaldaten</h4>
    <table class="table-hover" width="100%" cellpadding="10" cellspacing=""> ...
      </table>
  </div>

  <!-- Kontaktdaten-Spalte -->
  <div class="span6">
    <h4>Kontaktdaten</h4>
    <table class="table-hover" width="100%" cellpadding="10" cellspacing=""> ...
      </table>
  </div>
</div>
```

Abb. 64: Zusammengeführte Personal- und Kontaktdaten umgeben von einem Zeilenkontainer

8.5.2 Weitere Änderungen in „Mitglied hinzufügen“

- Das „Vermerk“-Feld wurde aufgrund des logischen Zusammenhangs von den Personaldaten in die Kontaktdaten übertragen. Der Vermerk beschreibt dabei einen Adresszusatz z.B. Kellergeschoss. Vor der Datenbank-Migration war der Vermerk in der Tabelle „*person*“ angegeben, ist aber im Zuge der Migration in die Tabelle „*adresse*“ überführt worden. Es mussten daher in sämtlichen Controllern und Models die den Vermerk ausgaben, speicherten oder neu angelegt haben, auf die neue Tabelle „*adresse*“ angepasst werden.
- Jegliche Datumsangaben wurden bisher in der Form YYYY-MM-DD ausgeben, sprich direkt aus der Datenbank übernommen. Die Umstellung in ein anderes Datumsformat z.B. nach DD.MM.YYYY konnte hier bequem mit der Methode „*strftime*“ durchgeführt werden. Im nachfolgenden wird ein Beispiel der Methode dargestellt. Zu beachten ist, dass die Methode nur auf einem gültigem Objekt aufgerufen werden kann. Es ist daher vor der Konvertierung zwingend erforderlich das Objekt auf seine Gültigkeit zu überprüfen.

```
<tr>
  <th align="left">Geburtsdatum:</th>
  <td>&lt;? text_field_tag :gebDatum, (!@Person.Geburtsdatum.nil?) ? @Person.Geburtsdatum.strftime("%d.%m.%Y") : nil ?&gt;</td>
</tr>
```

Abb. 65: Überprüfung auf ein gültiges Objekt, und schlussendliche Konvertierung nach DD.MM.YYYY

- Felder bei denen Datumseingaben gemacht werden können, waren nicht durchgehend mit der jQuery-UI-Kalender-Erweiterung belegt. Dies wurde nun auf alle Datumsfelder angewendet. Um des Weiteren eine bequeme Möglichkeit von Datumseingaben zu schaffen die in der Zukunft oder Vergangenheit liegen, ist nun auch zusätzlich eine Umschaltung zwischen den Jahren und Monaten möglich.

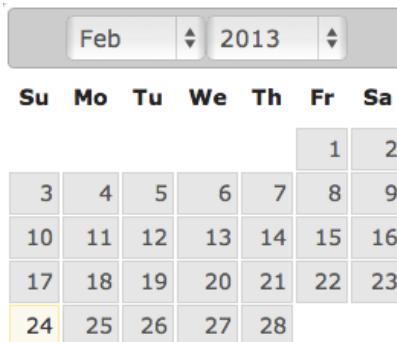


Abb. 66: Die Auswahl der Jahre und Monate ist bequem per Auswahldialog möglich

Die Implementierung der Umschaltung ist denkbar einfach. Dazu werden jegliche Felder, die eine Datumseingabe zulassen, um die Attribute „*changeMonth*“ und „*changeYear*“ erweitert.

```
$(function() {
  $("#gebDatum").datepicker({
    dateFormat: 'dd.mm.yy',
    changeMonth: true,
    changeYear: true
});
```

Abb. 67: Erweitert den Kalender um die Umschaltung zwischen den Jahren und den Monaten

- Email-Adressen werden neben der Datenbank-Tabelle „*ozbperson*“ nun auch in der Tabelle „*person*“ gespeichert und auch von dort in sämtlichen Ausgaben ausgelesen. Die Speicherung in der Tabelle „*ozbperson*“ erfolgt nur noch aus Kompatibilitätsgründen zu dem Authentifizierungs-Gem „*Devise*“, welches die Email-Adresse zwingend benötigt.

9 Kontenverwaltung (N. Usow)

9.1 Beschreibung

Die Kontenverwaltung ist einer der zentralen Punkte innerhalb der OZB-Applikation, hier lassen sich neue EE- und ZE-Konten anlegen, einsehen, ändern und löschen. Des Weiteren lässt sich der Kontoklassenverlauf einsehen und nun auch die Kontoauszug-Funktion benutzen.

Es existieren zwei Arten der Kontendarstellung. Die erste Variante ist für jeden Benutzer ohne jegliche Sonderberechtigungen sichtbar. Sie ist direkt in der Hauptnavigation unter „Meine Konten“ erreichbar. Die andere Art ist die Kontendarstellung unter der Mitgliedsverwaltung. Dort können Benutzer mit den richtigen Sonderrechten auch die Konten anderer Benutzer verwalten.

9.2 Horizontales Scrollen

Gerade beim ZE-Konto ist es besonders wichtig die große Fülle an Informationen kompakt aber dennoch komplett und ohne Informationsverlust zu präsentieren. Die vorangegangene tabellarische Darstellung hatte den Nachteil gehabt, dass sie fix auf 940px angepasst war. Informationen mussten deshalb weglassen werden, da sie ansonsten nicht leserlich innerhalb dieser Breite gepasst hätten. Um eine bessere und benutzerfreundlichere Darstellungsform zu erreichen, wird nun daher auf horizontales Scrollen innerhalb der Tabelle gesetzt. So können zum Beispiel beim ZE-Konto die 15 Spalten ohne jegliche Stauchung oder Weglassung untergebracht werden.

Um das horizontale Verhalten umzusetzen wurde früher auf Frames gesetzt, damit konnte innerhalb einer Seite noch eine Seite inkludiert werden. Dieses Verfahren ist aber aus heutiger Betrachtung technisch als auch aus der Sicht der Standardkonformität nicht mehr zeitgemäß. Frames haben den Nachteil, dass sie meistens zwei verschiedene und getrennte Implementierungen der Seiten brauchen, jeweils den Hauptframe in den dann der Unterframe eingebunden wird. Des Weiteren musste dann noch die Synchronisation zwischen Haupt- und Unterframe hergestellt werden. Um ein *frameähnliches* Verhalten zu simulieren, ohne jedoch Frames zu verwenden, kann man mittlerweile auf moderne CSS(Cascading-Stylesheets)-Eigenschaften zurückgreifen. Diese transformieren zur Laufzeit ausgewiesene Teile der Webseite auf eine gewünschte Anzeige.

Letzte Kontenbewegung	Laufzeit Jahre	Modus	Tilgung	Sparen	RDU	KDU	kalkulierte Leihpunkte	tatsächliche Leihpunkte	Aktion
01.03.2012	4 Monate	M	71,25	33,72	5,34	0,00	0	0	KKL-Verlauf Ändern L
01.03.2012	5 Monate	M	110,00	116,72	9,35	0,00	0	0	KKL-Verlauf Ändern L
01.03.2012	3 Monate	M	88,89	92,35	5,78	0,00	0	0	KKL-Verlauf Ändern L

Abb. 68: Horizontales Scrollen innerhalb der tabellarischen ZE-Konto-Darstellung

Um das horizontale Scrollen zu realisieren muss zuerst die bestehende HTML-Kontotabelle mit einem Kontainer-Element umgeben werden. Die Kontotabelle wird somit in das Kontainer-Element (DIV-Tag) eingebettet. Durch die Inline-CSS-Auszeichnung „`style="overflow: auto; width: 940px"`“ des Kontainers wird es ermöglicht, dass sich darin eingebettete Inhalte nicht mehr stauchen sondern komplett in der Breite ausdehnen dürfen, dies wird mit „`overflow: auto`“ erreicht.

```
<!-- Umschliessender Kontainer -->
<div style="overflow: auto; width: 940px">
  <table class="table table-striped tablesorter konto-list-table">...
    </table>
</div>
```

Abb. 69: Tabelle wird in ein Kontainer-Element inkludiert

Die Ergänzungen mit dem horizontalen Scrollen fanden in den folgenden Dateien statt:

- app/views/willkommen/_index_table_ee.html.erb
- app/views/willkommen/_index_table_ze.html.erb
- app/views/ozb_konto/_index_table_ee.html.erb
- app/views/ozb_konto/_index_table_ze.html.erb

9.3 Kontendarstellung

Die Darstellung der EE- und ZE-Konten war bisher nicht optimal gelöst, da sie einige unnötige Attribute enthielt und einige auch gänzlich fehlten. Die EE- und ZE-Konten wurden daher um Attribute ergänzt und die Bezeichnungen bereinigt. Im Nachfolgenden wird auf die Darstellung der Konteninhalte und der passenden Formatierung eingegangen. Um eine möglichst ergonomische Darstellung der Inhalte zu erreichen werden diese entsprechend ihrem Inhalt links-, rechtsbündig oder zentriert ausgerichtet. Des Weiteren wurde bei Zahleninhalten durchgehend ein Tausender trennpunkt eingeführt, um noch die Zahlenwerte aus der Datenbank dem deutschen Format anzupassen, wurde der Dezimalpunkt durch das Dezimalkomma ersetzt und eine Rundung auf zwei Nachkommastellen implementiert.

9.3.1 Neue EE-Konto-Anzeige

Neue Anzeige	Darstellungsform
EE-Konto	zentriert
Letzte Kontobewegung	zentriert
Währungssaldo	rechtsbündig, Dezimalkomma, Tausenderpunkt
Punktesaldo	rechtsbündig, Ganzzahl, Tausenderpunkt
Einrichtungsdatum	zentriert
Dispo-Limit	rechtsbündig, Dezimalkomma, Tausenderpunkt
Konto-Nr.	linksbündig
BLZ	zentriert
Kreditinstitut	linksbündig

9.3.2 Neue ZE-Konto-Anzeige

Neue Anzeige	Darstellungsform
ZE-Konto	zentriert
ZE-Nr.	zentriert
ZE-Betrag	rechtsbündig, Dezimalkomma, Tausenderpunkt
Rest-ZE	rechtsbündig, Dezimalkomma, Tausenderpunkt
Punktesaldo	rechtsbündig, Ganzzahl, Tausenderpunkt
Letzte Kontobewegung	zentriert
Laufzeit Jahre	zentriert
Modus	zentriert
Tilgung	rechtsbündig, Dezimalkomma, Tausenderpunkt
Sparen	rechtsbündig, Dezimalkomma, Tausenderpunkt
RDU	rechtsbündig, Dezimalkomma, Tausenderpunkt
KDU	rechtsbündig, Dezimalkomma, Tausenderpunkt
kalkulierte Leihpunkte	rechtsbündig, Ganzzahl, Tausenderpunkt
tatsächliche Leihpunkte	rechtsbündig, Ganzzahl, Tausenderpunkt

Die Ergänzungen an den Kontendarstellungen fanden in den folgenden Dateien statt:

- app/views/willkommen/_index_table_ee.html.erb
- app/views/willkommen/_index_table_ze.html.erb
- app/views/ozb_konto/_index_table_ee.html.erb
- app/views/ozb_konto/_index_table_ze.html.erb
- app/assets/stylesheets/application.css

Die rechtsbündige und zentrierte Darstellung wird mit CSS-Klassen umgesetzt, diese sind in der Stylesheet-Datei „application.css“ in Form von Klassen hinterlegt worden und müssen nun nur noch auf die Tabellenzeilen angewendet werden.

Die Rundung auf zwei Nachkommastellen, die Tausendertrennzeichen und das Dezimalkomma können bequem mit der Methode „number_with_precision“ und den Symbol-Parametern „:precision“ (Nachkommastellen) und „:delimiter“ (Tausendertrennzeichen) gesetzt werden. Das Setzen des Dezimalkommas wird automatisch durch die deutsche Lokalisierung der Applikation von Ruby vorgenommen.

```
<td class="zentriert"><a href="/Darlehensverlauf/&= ee_konto.KtoNr %>/EE">&= ee_konto.KtoNr %></a></td>
<td class="zentriert">&= ee_konto.ozb_konto.saldoDatum.strftime("%d.%m.%Y") %></td>
<td class="rechtsbuendig">&= number_with_precision(ee_konto.ozb_konto.wSaldo, :precision => 2, :delimiter => ".") %></td>
<td class="rechtsbuendig">&= number_with_precision(ee_konto.ozb_konto.pSaldo, :precision => 0, :delimiter => ".") %></td>
<td>&= ee_konto.ozb_konto.ktoEinrDatum.strftime("%d.%m.%Y") %></td>
<td class="rechtsbuendig">&= number_with_precision(ee_konto.kreditlimit, :precision => 2, :delimiter => ".") %></td>
```

Abb. 70: Ausrichtung inkl. der Formatierung der Zahlenwerte mit „number_with_precision“

9.3.3 Weitere Änderungen an der Kontendarstellung

- Die Anzeige jeglicher Datumsinhalte wurde mit der Methode „strftime("%d.%m.%Y")“ auf das Format DD.MM.YYYY angepasst.
- Der Kontoauszug-Button wurde entfernt. Es wurde stattdessen eine Verlinkung zum Darlehensverlauf eines Kontos gesetzt. Von dort aus kann dann auf den Kontoauszug zurückgegriffen werden.
- Kalkulierte Leihpunkte und tatsächliche Leihpunkte sind nach der Datenbankmigration mit in die ZE-Kontenübersicht aufgenommen worden. Diese werden aus der Datenbank-Tabelle „zekonto“ ausgelesen und entsprechend dargestellt.
- Die Änderungen an den Kontendarstellungen (umbenennen, hinzufügen und entfernen von Attributen) wurde auch in den dazugehörigen Formularen zum Editieren und Hinzufügen von neuen EE- und ZE-Konten durchgeführt.
- Die Spalte Einrichtungsdatum in der Kontendarstellung wird für Benutzer ohne Sonderberechtigungen nicht angezeigt. Dies wird zuvor mit einer Abfrage des aktuellen Rechts des Benutzers abgefragt. Ist das benötigte Recht nicht vorhanden, so wird die Spalte ausgeblendet.

9.4 Letzte Bank und Bankverbindung beim EE-Konto

Das Eintragen bzw. Anlegen einer Bankverbindung bei einem EE-Konto war in dem vorhergehenden Zustand nicht besonders benutzerfreundlich gestaltet. Beim Anlegen eines neuen EE-Kontos musste jedes Mal die Bankverbindung neu hinterlegt werden. Gerade bei diesem geschäftskritischen Prozess und den sensiblen Daten sollte die Fehleranfälligkeit auf ein Minimum reduziert werden. Es wird jetzt daher die zuletzt benutzte Bankverbindung zu einem EE-Konto ausgelesen und dem Benutzer vorausgefüllt präsentiert, sodass dieser diese nicht mehr manuell eintippen muss. Sollte der Benutzer noch gar keine Bankverbindung hinterlegt haben, so bleiben die Bankverbindungsfelder leer und er kann eine eigene Bankverbindung seiner Wahl eintragen.

Bankverbindung

Bank Konto-Nr.*:	<input type="text" value="XXXXXX"/>
BLZ*:	<input type="text" value="64050000"/>
Name der Bank:	<input type="text" value="KSK Reutlingen"/>
BIC:	<input type="text"/>
IBAN:	<input type="text"/>

Abb. 71: Dieser Benutzer hat bereits ein EE-Konto mit einer Bankverbindung hinterlegt, daher werden nun die Felder vorausgefüllt

9.4.1 Implementierung

Die Implementierung fand in den folgenden Dateien statt:

- app/controllers/ozb_konto_controller.rb
- app/views/ozb_konto/_form_ee.html.erb

Vor dem Laden der View wird zuerst im Controller abgefragt ob der Benutzer bereits eine Bankverbindung besitzt, dabei wird die letzte Bankverbindung ausgewählt. Dies wird durch die absteigende Sortierung nach den „GueltigVon“-Attribut sichergestellt. Ist ein Wert vorhanden, so wird er in die lokale Variable „result“ übertragen. Daraufhin erfolgt dann sofort eine Überprüfung, ob zu dieser Bankverbindung auch noch eine gültige Bank existiert. Die Abgleichung zwischen der Bank und der Bankverbindung erfolgt dabei durch die Bankleitzahl.

```
def new
  @action      = "new"
  @ozb_konto  = OzbKonto.new
  @kontotyp   = params[:kontotyp]
  @bankverbindung = Bankverbindung.new
  @bank       = Bank.new
  result      = nil

  # Gibt es eine bereits bestehende Bankverbindung für diesen Benutzer
  if !(result = Bankverbindung.find(:first, :conditions => { :Pnr => params[:Mnr] }, :order => "GueltigVon DESC")).nil?
    @bankverbindung = result
  end

  # Gibt es zu der Bankverbindung auch eine Bank
  if !(result = Bank.find_by_BLZ(@bankverbindung.BLZ)).nil?
    @bank = result
  end

  render "new"
end
```

Abb. 72: ozb_konto_controller.rb (Controller)

Um zu unterscheiden, ob der Benutzer eine neue Bankverbindung inkl. der Bank angelegt hat, oder eine bestehende verwendet, wird beim initialen Aufruf geschaut, ob die angegebenen Daten per HTTP-Parameterübergabe oder aus der Datenbank kommen. Abhängig vom Datenursprung kann dann die temporäre Variablenzuweisung erfolgen.

Hat der Benutzer noch keine Bankverbindung und auch bisher kein EE-Konto angelegt, so bleiben die temporären Variablen „blz“, „bic“, „bankname“, „bankktonr“ und „iban“ leer.

```
blz      = ""
bic     = ""
bankname = ""
bankktonr = ""
iban    = ""

# Falls manuell eine neue Bank eingegeben wurde
if !params[:ozb_konto].nil?
  blz      = params[:ozb_konto][:ee_konto_attributes][:Bankverbindung_attributes][:Bank_attributes][:BLZ]
  bic     = params[:ozb_konto][:ee_konto_attributes][:Bankverbindung_attributes][:Bank_attributes][:BIC]
  bankname = params[:ozb_konto][:ee_konto_attributes][:Bankverbindung_attributes][:Bank_attributes][:BankName]
  bankktonr = params[:ozb_konto][:ee_konto_attributes][:Bankverbindung_attributes][:BankKtoNr]
  iban    = params[:ozb_konto][:ee_konto_attributes][:Bankverbindung_attributes][:IBAN]
# Bankverbindung und Bank besteht bereits
elsif !@bankverbindung.nil? && !@bank.nil?
  blz      = @bank.BLZ
  bic     = @bank.BIC
  bankname = @bank.BankName
  bankktonr = @bankverbindung.BankKtoNr
  iban    = @bankverbindung.IBAN
end
```

Abb. 73: Bankverbindung

9.5 Punkteberechnung

9.5.1 Beschreibung

Die Berechnung der Punkte erfolgt für ein OZB-Konto und den dahinterliegenden Buchungen. Dabei werden die Punkte unter Berücksichtigung der KKL-Klassen und anhand des Belegdatums und der Kontonummer, die hinter einer Buchung, steht berechnet. Das Ergebnis kann dann dazu verwendet werden um für eine bestehende Buchung eine Punkteaktualisierung vorzunehmen.

Die Implementierung der Punkteberechnung liegt bereits als PHP-Code auf der noch produktiven Version der OZB-Applikation vor. Es wurde daher eine gleichwertige Portierung mit Ruby-Code umgesetzt. Nachfolgende Entwickler der OZB-Applikation können somit diese Punkteberechnung als Grundlage für weitere Implementierungen benutzen.

Um sicherzustellen, dass die Berechnung der Punkte und die Logik äquivalent zur PHP-Version funktioniert, wurden Tests mit gleichen Eingabeparametern durchgeführt. Die Resultate waren dabei stets identisch.

9.5.2 Implementierung

Für die Implementierung wurde die Klasse Punkteberechnung angelegt. Die Bezeichnung der Variablen und der Funktionen wurden wie auch bei der PHP-Version weitestgehend in Ruby übernommen. Um die Klasse möglichst universell und komfortabel zu nutzen wurden alle Methoden als statisch deklariert, so kann die Klasse und ihre Methoden ohne das Anlegen eines Objektes benutzt werden. Das Schlüsselwort „self.“ vor der Methodenbezeichnung legt eine statische Methode fest. Um den Grad der Modularisierung zu erhöhen wurde die Klasse außerdem in eine externe Bibliothek ausgelagert. So kann man nun bequem aus jeder Stelle im Programm die Funktion zur Berechnung der Punkte mit „*Punkteberechnung*::calc_score()“ aufrufen.

Die Implementierung fand in der folgenden Datei statt:

- app/lib/Punkte/Punkteberechnung.rb

„calc_score“ und „calc_score_I“ sind die primären Methoden zur Berechnung der Punkte. Die Methode „calc_score_I“ wird von der vorhergehenden Methode aufgerufen und berücksichtigt dabei den Kontenklassenverlauf in der Berechnung.

```

def self.calc_score(date_begin, date_end, money_begin, kontonummer)
  t = Array.new
  # morgiges Datum
  t << date_begin.tomorrow
  tt = Array.new
  # liefert Anzahl der Tage des Monat im aktuellen Jahr
  tt << Time::days_in_month(date_begin.month, date_begin.year)
  # Konvertierung des Start- und Enddatums in Sekunden
  b = date_begin.to_time.to_i
  e = date_end.to_time.to_i
  # Differenz der Tage bis zum Ende des Monats in Sekunden
  d = b + ((tt[0] - date_begin.day) * 86400)

  # Berechnet für die verbleibenden Tage in den Monaten, die Differenzen und die Anzahl der Tage im Monat
  while d <= e do
    t << Time.at(d).strftime("%Y-%m-%d").to_date
    b = d + 86400 # +1 Tag
    t << Time.at(b).strftime("%Y-%m-%d").to_date
    d = b + (Time::days_in_month(Time.at(b).to_datetime.month, Time.at(b).to_datetime.year) - 1) * 86400
    tt << Time::days_in_month(Time.at(d).to_datetime.month, Time.at(d).to_datetime.year)
  end
  t << date_end

  mfaktor = 0
  i = 0
  while i < t.size
    mfaktor += self.calc_score_1(t[i], t[i+1], money_begin, tt[i/2], kontonummer)
    i += 2
  end

  return mfaktor
end

def self.calc_score_1(date_begin, date_end, money_begin, months_days, kontonummer)
  db_begin = KklVerlauf.find(:all,
    :conditions => ["ktoNr = ? AND kklAbDatum <= ?", kontonummer, date_begin],
    :order => "kklAbDatum DESC"
  ).first

  db_duration = KklVerlauf.find(:all,
    :conditions => ["ktoNr = ? AND kklAbDatum <= ? AND kklAbDatum > ?", kontonummer, date_end, date_begin],
    :order => "kklAbDatum ASC"
  )

  t = Array.new
  t << date_begin
  t << db_begin.kkl

  db_duration.each do |temp1|
    t << temp1.kklAbDatum
    t << temp1.kkl
  end

  t << date_end.tomorrow
  t << "dummy"

  # hole die prozentsätze
  mfaktor = 0
  scores = 0
  i = 0
  while i < (t.size - 2)
    # 01.2011: beim neg. Saldo keine Berücksichtigung der KKL-Prozentsatzes, stattdessen immer KKL 100% annehmen
    if (money_begin >= 0)
      mfaktor = self.calc_percentage(t[i], t[i+2], t[i+1]) # Anzahl der Tage um KKL-Prozentsatz verringern
    else
      mfaktor = self.count_days_exact(t[i], t[i+2]) # Volle Anzahl der Tage als Faktor nehmen
    end

    scores += mfaktor * (money_begin / 30)
    i += 2
  end

  return scores
end

```

Die Hilfsmethode „*calc_percentage*“ ermittelt den gemittelten Faktor des Prozentsatzes einer Kontenklasse über eine Periode. Sie gibt die gemittelten Tage für eine Kontenklasse zurück.

```
def calc_percentage(date_begin, date_end, kkl)
  db_begin = Kontenklasse.find(:all,
    :conditions => ["kkl = ? AND kklAbDatum <= ?", kkl, date_begin],
    :order => "kklAbDatum DESC",
    :limit => 1
  ).first

  db_duration = Kontenklasse.find(
    :all,
    :conditions => ["kkl = ? AND kklAbDatum <= ? AND kklAbDatum > ?", kkl, date_end, date_begin],
    :order => "kklAbDatum ASC"
  )

  db_duration.each do |temp1|
    t << temp1.kklAbDatum
    t << temp1.prozent
  end

  t << date_end
  t << "dummy"

  # rechne gemittelte Tage
  mtage = 0
  i = 0
  while i < (t.size - 2)
    mtage += (count_days_exact(t[i], t[i+2])) * (t[i+1] / 100)
    i += 2
  end

  return mtage
end
```

Die „*count_days_exact*“-Hilfsmethode berechnet die Anzahl der Tage zwischen zwei Datumsangaben auf der Grundlage von 360 Tagen im Jahr und 30 Tagen im Monat. Sie simuliert die Excel-Funktion TAGE360(datum, datum, art)

```
def count_days_exact(first_time, second_time)
  # Es wurde beobachtet, dass das Ergebnis mit Nachkommastellen berechnet wird => Rundung nötig!
  return ((second_time.to_time.to_i - first_time.to_time.to_i) / 86400.0).round(0) # integer / float => float
end
```

Abb. 74: Hilfsmethode „*count_days_exact*“

9.5.3 Benutzung

Bevor die Klasse benutzt werden kann muss sie zuerst explizit importiert werden. Dies geschieht über die require-Direktive von Ruby am Anfang des Controllers.

```
require "Punkteberechnung"
```

Abb. 75: Punkteberechnung-Klasse wird importiert

10 Webimport (N. Usow)

10.1 Beschreibung

Mit dem Webimport ist es möglich eine bestehende CSV-Datei aus einem externen Finanzbuchhaltungssystem zu verarbeiten und nachher in die OZB-Datenbank zu importieren, und daraufhin die Punkte zu berechnen sowie bestehende Buchungen zu aktualisieren.

Das Hochladen der CSV-Datei auf das OZB-Dateisystem und das anschließende Parsen bzw. Zerlegen der CSV-Datei wurde schon erfolgreich von Herrn Hillert aus dem SS12 in Form der Klasse „*CSVImporter.rb*“ implementiert. Im Zuge des Wechsels auf die neue Ruby-Version sind allerdings einige Teile nicht mehr zur neuen Version kompatibel. Es wurde daher zuerst eine Anpassung des bestehenden Codes zum Hochladen und Einlesen der Datei vorgenommen. Das nachträgliche Verarbeiten der Inhalte der CSV-Datei nach einer bestimmten Importlogik und das Schreiben in die Datenbank werden hier neu implementiert.

10.2 Dateistruktur

Die Struktur der CSV-Datei ist relativ simpel gehalten. Es existieren acht Spalten: Belegdatum, Buchungsdatum, Belegnummernkreis, Belegnummer, Buchungstext, Buchungsbetrag Euro, Sollkonto und Habenkonto. Zu beachten sei, dass die Datei im LATIN1 bzw. ISO 8859-1 ausgeliefert wird, Ruby aber standardmäßig auf das UTF-8-Encoding setzt. Dieser Unterschied wurde aber bereits in der Implementierung von Herrn Hillert durch eine Konvertierung zu UTF-8 vollzogen, wo die Daten nun zur weiteren Verarbeitung konsistent im richtigen Encoding vorliegen.

```
1 |Belegdatum;Buchungsdatum;Belegnummernkreis;Belegnummer;Buchungstext;Buchungsbetrag Euro;Sollkonto;Habenkonto
2 |04.12.2012;04.12.2012;B-;775;0108 |Helga Einlage S090930;150,00;1200;70108
3 |04.12.2012;04.12.2012;U-;773;0108-S090930 |Helga Tilgung;150,00;70108;10108
4 |05.12.2012;05.12.2012;P-;55;70889-70088 Überweisung Punkte von |;13.000,00;80889;80088
```

Abb. 76: CSV-Struktur aus dem produktiven Finanzbuchhaltungssystem

10.3 Anpassungen für die neue Ruby-Version

Beim Wechsel auf die neue Rails-Version traten einige Inkompabilitäten auf. So verläuft nun das Hochladen/Öffnen und Einlesen der Datei nach einem minimal anderen Schema. Nachfolgend die Änderungen die am Code durchgeführt werden mussten:

Das Öffnen der Datei nach dem Hochladen und das anschließende Schreiben auf das Dateisystem muss jetzt zusätzlich mit dem binary-Flag ausgezeichnet werden, hier „*wb*“ write+binary. Durch diese Auszeichnung wird es ermöglicht die Inkompabilitäten zwischen den beiden Encodings auszugleichen.

```
File.open(uploaded_disk, 'wb') do |file|
  file.write(uploaded_io.read)
end
```

Abb. 77: Wegen Inkompabilitäten beim Encoding muss noch das "b"-Flag (binary) bei Öffnen der Datei gesetzt werden

Das Parsen der Datei geschieht nun mit der aktualisierten Ruby-Klasse CSV, diese erwartet in der aktuellen Ruby-Version eine andere Parametrisierung. So muss jetzt das Trennzeichen zwischen den Spalten in der CSV-Datei explizit mit dem Symbol „*:col_sep*“ angegeben werden. Die vorhergehende Parametrisierung hatte hier noch eine feste Parameterreihenfolge ohne Angabe des Symbols „*:col_sep*“ erwartet.

```
CSV.parse(utf8_encoded_file, :col_sep => ";") do |row|
```

Abb. 78: Das Symbol *:col_sep* erwartet nun das gewünschte Trennzeichen

10.4 Implementierung

Im bereits bestehenden produktiven System der OZB-Applikation, die in PHP programmiert worden ist, besteht bereits die Funktionalität des Importes. Das Augenmerk bei dieser Implementierung wurde daher auf eine möglichst äquivalente Portierung des bestehenden Codes in Ruby-Code gelegt.

Die Portierung des PHP-Codes war langwieriger als erwartet, da PHP in einigen Bereichen dem Programmierer doch sehr offene Möglichkeiten zur Implementierung bietet. Ruby und Rails haben dagegen strikte Strukturen, die eingehalten werden müssen. Es fand daher keine Eins-zu-eins-Portierung statt sondern eher eine Umsetzung der Logik in Ruby-Code.

Wie auch bei der Punkteberechnung wurde auch hier versucht die Bezeichnung der Variablen und Funktionen aus dem PHP-Code zu übernehmen.

Da der gesamte Code zur Verarbeitung der CSV-Datei, dem Hinzufügen der Buchungen und dem anschließenden Aktualisieren der Punkte mehr als 600 Zeilen Code enthält wird im folgenden nur auf die Schlüsselstellen der Implementierung eingegangen.

Die Implementierung fand in den folgenden Dateien statt:

- app/controllers/webimport_controller.rb
- app/views/webimport/index.html.erb

Durch die vorangegangene Arbeit von Herrn Hillert kann nun auf jede einzelne Spalte innerhalb einer Zeile der CSV-Datei zurückgegriffen werden und abhängig vom Inhalt der Zeilen eine eigene Verarbeitungslogik angewendet werden.

```
rows.each do |r|
begin
  ActiveRecord::Base.transaction do
    end
  end
end
```

Abb. 79: Jede Zeile der CSV-Datei wird durchlaufen (Code ist zusammengeklappt)

Jede Zeile ist ein Array, welches jeweils die Spalteninhalte auch als Array-Elemente in der Variable „r“ hält. Um den Code lesbar darzustellen und nicht ständig auf einen Array-Index zugreifen zu müssen wird innerhalb jeder Zeileniteration jeder Spalteninhalt einer leserlichen Variable zugewiesen.

```
habenbetrag      = 0.0
sollbetrag       = 0.0
typ              = "W"
pkte_acc         = 0
storno           = 0
belegnummernkreis = r[2]
belegnummer      = r[3].to_i
buchungstext     = r[4].to_s
sollkontonummer = r[6].strip.to_i
habenkontonummer = r[7].strip.to_i
buchungsdatum    = Date.strptime(r[0], "%d.%m.%Y").strftime("%Y-%m-%d")
buchungsjahr     = Date.strptime(r[0], "%d.%m.%Y").strftime("%Y").to_i
wertstellungsdatum = Date.strptime(r[1], "%d.%m.%Y").strftime("%Y-%m-%d")
betrag           = (r[5].gsub(/\./, '')).gsub(., '.').to_f
s                = r[6].length # Länge Söllkonto
h                = r[7].length # Länge Habenkonto

temp             = buchungstext.split(" ")
gesellschafter  = temp[0]
```

Abb. 80: Die Spalten innerhalb einer Zeile werden Variablen zugewiesen

Abhängig von der Zeichenlänge der Soll- und Habenkontonummer wird in verschiedene Bearbeitungsfälle verzweigt. Besteht die Soll- oder Habenkontonummer aus fünf Zeichen, so handelt es sich auf jeden Fall vorerst um eine normale Buchung. Trifft kein Fall zu, so ist es eine FIBU-Buchung. Diese wird vom System nicht importiert und kann auch daher in der aktuellen Zeileniteration ignoriert werden.

```
if (h == 5 || s == 5) ...
else ...
# Eine FIBU-Buchung wird ignoriert.
end
```

Abb. 81: Abhängig von der Zeichenlänge der Soll- und Habenkontonummer werden verschiedene Importlogiken angestoßen (Code ist zusammengeklappt)

Sind daraufhin auf der nächsten Ebene beide Konten **nicht** mehr fünfstellig, so kann es sich um eine Storno-Buchung handeln oder auch um eine gewöhnliche Buchung. Sind aber beide Konten fünfstellig kann es sich um vier weitere verschiedene Arten einer Buchung handeln.

```
if (h == 5 || s == 5)
# Abbuchung-Leihpunkte-Buchung, Storno-Abbuchung-Leihpunkte-Buchung, Punkteüberweisung-Buchung oder Konto-zu-Konto-Buchung
if (h == 5 && s == 5) ...
# Gewöhnliche Buchung oder Storno-Buchung
else ...
end
else ...
# Eine FIBU-Buchung wird ignoriert.
end
```

Abb. 82: (Code ist zusammengeklappt)

Für den Fall, dass beide Kontonummern fünfstellig sind, wird anhand der gesamten Kontonummer und der ersten Ziffer der Kontonummer entschieden um welche Art der Buchung es sich hier handelt. Es wird dabei nach den folgenden vier Buchungen unterschieden:

- Abbuchung-Leihpunkte-Buchung
- Storno-Abbuchung-Leihpunkte-Buchung
- Punkteüberweisung-Buchung
- Konto-zu-Konto-Buchung

```
# Abbuchung-Leihpunkte-Buchung
if (habenkontonummer == 88888 && sollkontonummer != 88888) ...
end

# Storno-Abbuchung-Leihpunkte-Buchung
if (sollkontonummer == 88888 && habenkontonummer != 88888) ...
end

# Punkteüberweisung-Buchung
if (habenkontonummer[0] == "8" && sollkontonummer[0] == "8" && sollkontonummer != 88888 && habenkontonummer != 88888) ...
end

# Konto-zu-Konto-Buchung
if (habenkontonummer[0] != "8" && sollkontonummer[0] != "8") ...
end
```

Abb. 83: Abbuchung-Leihpunkte-Buchung, Storno-Abbuchung-Leihpunkte-Buchung, Punkteüberweisung-Buchung oder Konto-zu-Konto-Buchung (Code ist zusammengeklappt)

Die Logiken innerhalb dieser vier Buchungsarten unterscheiden sich geringfügig, benötigen aber rein von der Implementierung eine Menge an Platz. Eine Abbildung des Quellcodes würde hier daher jegliche Dimensionen sprengen. Es wird daher im Folgenden beispielhaft auf die Abbuchung-Leihpunkte-Buchung eingegangen. Für Informationen zu den weiteren Buchungsarten sei hier auf den Quelltext verwiesen.

Für die Abbuchung-Leihpunkte-Buchung werden die Daten im Buchungstextes nochmals weiter extrahiert, so dass man den Gesellschafter wie auch die Darlehensnummer bekommt. Der Typ bei dieser Art der Buchung wird hier auf „p“ gesetzt.

Im nächsten Schritt kann anhand der extrahierten Darlehensnummer aus dem Buchungstext eine gültige Einlage-Entnahme-Kontonummer aus einem ZE-Konto selektiert werden, welche später die Kontonummer innerhalb einer Buchung darstellt. Der bestehende Betrag aus der CSV-Datei wird negiert und als Sollbetrag verbucht.

Danach findet finalisierend eine Speicherung der extrahierten Daten als eine Buchung in die Datenbanktabelle „buchung“ statt. Schlussendlich wird mit dem Schlüsselwort „next“ zur nächsten Zeileniteration innerhalb der CSV-Datei gesprungen.

```
# Eine Abbuchung-Leihpunkte-Buchung. Buchung wird in DB eingetragen.
temp          = buchungstext.split(" ")
temp2         = temp[0].split("-")
gesellschafter = temp2[0]
darlehensnummer = temp2[1].to_i
typ           = "p"
kontonummer   = ZeKonto.find(
                  :all,
                  :conditions =>
                  {
                      :ZENr      => darlehensnummer,
                      :GueltigBis => "9999-12-31 23:59:59"
                  }
              ).first.EEKtoNr
sollbetrag    = betrag * (-1.0)

b = Buchung.new(
  :BuchJahr     => buchungsjahr,
  :KtoNr        => kontonummer,
  :BnKreis      => belegnummernkreis,
  :BelegNr      => belegnummer,
  :Typ          => typ,
  :Belegdatum   => buchungsdatum,
  :BuchDatum    => wertstellungsdatum,
  :Buchungstext => buchungstext,
  :Sollbetrag   => sollbetrag,
  :Habenbetrag  => habenbetrag,
  :SollKtoNr    => sollkontonummer,
  :HabenKtoNr   => habenkontonummer,
  :WSaldoAcc   => 0.0,
  :Punkte       => nil,
  :PSaldoAcc   => 0
)
b.save

collect_konten << kontonummer
imported_records += 1

next
```

Abb. 84: Implementierung einer Abbuchung-Leihpunkte-Buchung

Im Nachfolgenden wird die gewöhnliche Buchung beschrieben.

Bei einer gewöhnlichen Buchung ist die Zeichenlänge bei der Haben- oder Sollkontonummer nur fünf Zeichen lang. Handelt es sich um eine Buchung ins Haben wird der extrahierte Betrag aus der CSV-Datei dem Habenbetrag zugewiesen und schlussendlich mit den ausgelesenen Buchungsinformationen abgespeichert. Bei einer Sollbuchung ist das Vorgehen analog zur Habenbuchung, der Betrag wird hier nur dem Sollbetrag zugewiesen.

```
# Eine gewöhnliche Buchung. Buchung wird in DB eingetragen.
if (h == 5)
  kontonummer = habenkontonummer
  habenbetrag = betrag

  b = Buchung.new( ... )
)
b.save

collect_konten << kontonummer
imported_records += 1
next
elsif (s == 5)
  kontonummer = sollkontonummer
  sollbetrag = betrag

  b = Buchung.new( ... )
)
b.save

collect_konten << kontonummer
imported_records += 1
next
end
```

Abb. 85: Gewöhnliche Buchung (Soll- oder Habenbuchung) (Code ist zusammengeklappt)

Nach dem Import der Buchungen können nun mit der zuvor implementierten Punkteberechnung die Punkte der Buchungen und des OZB-Kontos aktualisiert werden.

11 Darlehensverlauf (M. Hoprich)

Bei dem Darlehensverlauf soll es einem Mitglied bei der o/ZB ermöglicht werden, eine Einsicht über die Buchungen eines gewählten EE- oder ZE-Kontos zu bekommen. Dies geschieht indem ein eingeloggtes Mitglied auf die Kontonummer der anzuzeigenden Buchungen klickt. Daraufhin werden die letzten zehn getätigten Buchungen, mit dem jeweiligen Buchungsdatum, Buchungstext, Sollbetrag, Habenbetrag und den Punkten angezeigt. Es wurde noch das Tagessaldo bis zu dem ersten angezeigten Buchungsdatum berechnet, sowie das Tagessaldo zu dem letzten Buchungsdatum. Weiterhin wurde noch eine Zeile mit den Punkten von dem Datum der letzten Buchung bis zu dem aktuellen Datum berechnet und angezeigt. Zu der Anzeige gehören auch noch der Vorname und der Nachname des Mitglieds sowie die Art des Kontos (ZE / EE) und die entsprechende Kontonummer. Die Möglichkeit ein Datumsintervall anzugeben sollte ebenfalls eingefügt werden, dies erlaubt es dem Mitglied nur die Buchungen in dem von ihm gewählten Zeitraum anzuzeigen. Eine weitere Funktion die dem Mitglied zur Verfügung stehen sollte war es einen Kontoauszug von den angezeigten Buchungen zu erstellen.

11.1 Umsetzung Darlehensverlauf

11.1.1 Anlegen des Controllers und der View

Zunächst mussten ein Controller und eine View angelegt werden, dies erfolgte mit folgender Kommandozeile die in den Editor eingegeben wurde:

```
rails generate controller Darlehensverlauf new
```

damit wurden die Dateien „darlehensverlauf_controller.rb“ und „new.html.erb“ erzeugt, in denen dann die Implementierung erfolgte.

11.1.2 Umsetzung des Links auf der Kontonummer

Um es zu ermöglichen auf die Kontonummer zu klicken, musste erst ein Link in den bereits vorhandenen Dateien „_index_table_ee.html.erb“ für die EE-Konten

```
<a href="/Darlehensverlauf/<%= ee_konto.KtoNr %>/EE"><%= ee_konto.KtoNr %></a>
```

Abb. 86: Beschreibt den Link auf einem EE-Konto mit der KontoNr. als Parameter

und „_index_table_ze.html.erb“ für die ZE-Konten

```
<a href="/Darlehensverlauf/<%= ze_konto.KtoNr %>/ZE"><%= ze_konto.KtoNr %></a>
```

Abb. 87: Beschreibt den Link auf einem ZE-Konto mit der KontoNr. als Parameter

eingefügt werden. Anhand des Links kann man unterscheiden ob es sich um ein EE- oder um ein ZE-Konto handelt, da der Parameter hierzu übergeben wird. Ebenfalls wird hier noch die dazugehörige Kontonummer übergeben. Dadurch wurde es ermöglicht die Buchungen des bestimmten Kontos anzuzeigen.

EE-Konto	Letzte Kontobewegung	Währungssaldo	Punktesaldo	Einrichtungsdatum	Dispo-Limit	Konto-Nr.	BLZ
50013	31.12.2011	0,00	0	28.04.2010	40,00	1011507447	12030000

Abb. 88: Ein Konto mit eingefügten Link

11.1.3 Implementierung des Controllers

In dem Controller werden sämtliche Parameter ausgewertet und nach der Auswertung so aufbereitet, dass man die Buchungsansicht anzeigen kann. Es gab im Controller den Fall, dass auf den Link des Kontos geklickt wurde bei dem daraufhin nur die letzten zehn Buchungen aus der Datenbank geholt wurden. Weiterhin gab es noch den Fall, dass ein Intervall von dem Mitglied angegeben werden konnte, indem es ein Anfangsdatum und ein Enddatum angab, um die Buchungen innerhalb dieses Intervalls zu bekommen.

11.1.4 Implementierung des Falls, dass auf den Link eines Kontos geklickt wurde

Im Controller mussten die zehn Buchungen zu der entsprechenden Kontonummer aus der Datenbank geholt werden, absteigend nach Belegdatum und Habenbetrag sortiert. Dies wurde mit folgender Codezeile durchgeführt:

```
@Buchung = Buchung.where("KtoNr = ?", params[:KtoNr]).order("Belegdatum DESC, HabenBetrag DESC").limit(10).reverse
```

Abb. 89: Die in Rails implementierte SQL-Abfrage um die zehn letzten Buchungen zu bekommen

Zur Berechnung des alten Tagessaldos musste noch zusätzlich die vorangegangene Buchung bestimmt werden. Dabei musste darauf geachtet werden, dass es sich um eine Währungsbuchung und nicht um eine Punktebuchung handelt, davon wurden die Werte der Felder „WaldoAcc“, „Punkte“ und das „Belegdatum“ benötigt. Zusätzlich musste noch die aktuelle Kontenklasse zu dem Konto bestimmt werden. Um die Kontenklasse zu bestimmen musste man in der Datenbank schauen, welche Kontenklasse gerade die Gültige ist. Dies konnte man durch eine absteigende Sortierung aller Kontenklassen zu diesem Konto erreichen. Von dieser sortierten Liste konnte man dann den ersten Eintrag nehmen.

```
@kklVerlaufKlasse = KklVerlauf.where("KtoNr = ?", params[:KtoNr]).order("KKLAbDatum DESC").limit(1).first.KKL  
@kklZhal = 1  
case @kklVerlaufKlasse  
when "A"  
  @kklZhal = 1  
when "B"  
  @kklZhal = 0.75  
when "C"  
  @kklZhal = 0.50  
when "D"  
  @kklZhal = 0.25  
when "E"  
  @kklZhal = 0  
end
```

Abb. 90: Die Implementierung zur Bestimmung des KKLS

Es musste nur noch die Differenz in Tagen von dem Belegdatum und des Datums der ersten Buchung gebildet werden, um danach nach der Formel

$$\frac{DifferenzInTagen \times WsaldoAcc}{30} \times Kontenklasse + Punkte$$

die Punkte für den vorangegangenen Tagessaldo zu bestimmen. Der Soll- oder Haben-Betrag des Tagessaldos entsprach dem Wert in dem Feld „WsaldoAcc“. Je nach Vorzeichen dieses Wertes wurde er bei der Spalte Haben, bei positivem Vorzeichen oder Soll, bei negativem Vorzeichen eingetragen. Das Datum des vorangegangenen Tagessaldos entsprach dem Datum der ersten Buchung aus den zehn Buchungen, die zuvor aus der Datenbank geholt wurden.

Buchungsdat	Buchungstext	Soll	Haben	Punkte
01.11.2011	Tagessaldo		2006,48	-47298
01.11.2011	Tilgung	111,11		0
01.11.2011	RDU	2,92		0
01.11.2011	ZE [REDACTED] ?berweisung		52,68	1354
02.11.2011	[REDACTED] R?ckbuchung Leihpunkte v. 29.04.10			80133
02.11.2011	Tilgung Rest	2111,13		0
02.11.2011	?bertrag Leihpunkte			24385
02.11.2011	Einlage [REDACTED]		166,00	48
02.11.2011	Tats. verbrauchte Leihpunkte			-57217
31.12.2011	Kienle Tassilo Umbuchung	32,12		0
31.12.2011	RDU Erstattung 2011		32,12	0
07.03.2013	Punkte von 31.12.2011 bis 07.03.2013			346
07.03.2013	Tagessaldo	0,00		1751

Abb. 91: Der Kontoauszug für die ersten zehn Buchungen eines Kontos

Um die vorletzte Zeile „Punkte von 31.12.2011 bis 20.02.2013“ zu bestimmen mussten erst die Tage zwischen dem Datum der letzten Buchung und des aktuellen Datums bestimmt werden, den Saldo der letzten Buchung und die Kontenklasse zum entsprechenden Konto. Mit diesen Werten konnte man dann anhand der Formel

$$\frac{AnzahlTage \times Saldo}{30} \times Kontenklasse$$

die Punkte der Zeile errechnen. Diese Zahl musste immer abgerundet werden, da stets nur ganze Zahlen vergeben werden konnten. Die Werte des Tagessaldos zum aktuellen Datum wurden dann anhand der Werte der zehn Buchungen und des vorangegangenen Tagessaldos sowie der Zeile „Punkte von TT.MM.JJJJ bis TT.MM.JJJJ“ berechnet. Dabei wurden alle Habenbeträge aufaddiert und davon die Summe der Sollbeträge abgezogen. War dieser Wert negativ, wurde er in der Spalte Sollbetrag des Tagessaldos eingetragen. War der Wert positiv, wurde er in der Spalte des Habenbetrages eingetragen. Die Spalte der Punkte des Tagessaldos wurde durch die Summe aller Punkte der angezeigten Buchungen bestimmt.

11.1.5 Implementierung für den Fall das ein Datumsintervall angegeben wurde

In diesem Fall mussten zuerst die angegebenen Daten auf das richtige Format geprüft werden.

```
#prüfen ob das anfangsdatum in richtigem format angegeben wurde
if (params[:vonDatum] =~ /[0-9]{2}.[0-9]{2}.[0-9]{4}/)
  @vonDatum = params[:vonDatum]
end

#prüfen ob das enddatum in richtigem format angegeben wurde
if (params[:bisDatum] =~ /[0-9]{2}.[0-9]{2}.[0-9]{4}/)
  @bisDatum = params[:bisDatum]
end
```

Abb. 92: Die Implementierung der Formatprüfung der beiden Daten

Wurde ein Datum geprüft, das nicht im richtigen Format war so wurde dem Mitglied eine Fehlermeldung ausgegeben mit der Aufforderung ein valides Datum einzugeben.

von: bis: Anzeigen

Es konnte keine Suche gestartet werden:

- Geben sie bitte ein valides Datum ein.

Abb. 93: Die Fehlermeldung bei falscher Eingabe eines Datums

Sind die beiden Daten im richtigen Format, so kann man die Buchungen in dem angegebenen Zeitraum aus der Datenbank holen.

```
@Buchung = Buchung.where("KtoNr = ? AND BuchDatum >= ? AND BuchDatum <= ?",
  params[:KtoNr],
  params[:vonDatum].to_date,
  params[:bisDatum].to_date).order("Belegdatum DESC, HabenBetrag DESC").reverse
```

Abb. 94: Die SQL-Abfrage zu den Buchungen in dem Datumsintervall

Sollte es keine Buchungen in dem angegebenen Zeitraum geben oder noch überhaupt keine Buchungen getätigten worden sein, so wird eine entsprechende Warnung ausgegeben

von: 01.02.2013 bis: 01.02.2013 Anzeigen

In dem angegebenen Zeitraum gibt es noch keine Buchungen.

Abb. 95: Die Warnung wenn es noch keine getätigten Buchungen in dem angegebenen Zeitraum gab

Gibt es in dem angegebenen Intervall Buchungen, so werden ebenfalls wie oben beschrieben die Tagessalden und die Spalte „Punkte von TT.MM.JJJJ bis TT.MM.JJJJ“ bestimmt und angezeigt.

von: 01.04.2010 bis: 01.07.2010 **Anzeigen**

Einlage-/Entnahmekonto Ktonr. [REDACTED]

Buchungsdat	Buchungstext	Soll	Haben	Punkte
01.04.2010	Tagessaldo			-60133
29.04.2010	[REDACTED] Abbuchung Leihpunkte			-80133
29.04.2010	[REDACTED] Entnahme [REDACTED]	4000,00		0
29.04.2010	[REDACTED] RDU ZE [REDACTED]		4000,00	0
29.04.2010	[REDACTED] Überweisung Punkte			20000
01.07.2010	[REDACTED] Tilgung	111,11		0
01.07.2010	[REDACTED] RDU	3,50		0
01.07.2010	Punkte von 29.04.2010 bis 01.07.2010			6300
01.07.2010	Tagessaldo	114,61		-113966

Abb. 96: Die Buchungen in dem angegebenen Zeitraum

Um den Namen des Mitglieds angezeigt zu bekommen musste man zunächst die Bank ID zu dem jeweiligen Konto bestimmen. Handelte es sich um ein ZE-Konto, so konnte man nur die Kontonummer zu dem dazugehörigen EE-Konto bestimmen und damit dann die dazugehörige Bank ID. Mit dieser Bank ID konnte man nun die PNr. des Mitglieds herausfinden, daraufhin die Person anhand der PNr., die dann den Namen und den Vornamen enthielt.

```
if params[:EEoZEkonto] == "EE"
  @bankId = EeKonto.where("KtoNr = ?", params[:KtoNr]).first.bankId
else
  @EeKtoNrZumZeKto = ZeKonto.where("KtoNr = ?", params[:KtoNr]).first.eeKtoNr
  @bankId = EeKonto.where("KtoNr = ?", @EeKtoNrZumZeKto).first.bankId
end

@pnrDesInhabers = Bankverbindung.where("ID = ?", @bankId).first.pnr
@personZurPnr = Person.where("Pnr = ?", @pnrDesInhabers).first
@nameZurPerson = @personZurPnr.name
@vornameZurPerson = @personZurPnr.vorname
```

Abb. 97: Die Implementierung um den Vor- und den Nach-Namen des Mitglieds zu bekommen

Nachdem der Name ausgelesen wurde, konnte man ihn in die Überschrift der Buchungen einfügen.

11.1.6 Kontoauszug erstellen

Um es dem Mitglied zu ermöglichen seine Buchungen auszudrucken musste ein Button hinzugefügt werden der auf eine weitere Seite verlinkt, die das Aussehen eines Kontoauszugs der o/ZB hat. Diese Seite konnte man ausdrucken um den Kontoauszug in Papierform zu haben.

o/ZB Stuttgart Einlage-/Entnahmekonto Ktonr. [REDACTED]

Datum des Kontoauszugs: 07.03.2013

Datum	Buchungstext	Soll	Haben	Punkte
01.11.2011	Tagessaldo		2006,48	-47298
01.11.2011	[REDACTED] Tilgung	111,11		0
01.11.2011	[REDACTED] RDU	2,92		0
01.11.2011	[REDACTED] ZE [REDACTED] Überweisung		52,68	1354
02.11.2011	[REDACTED] R?ckbuchung Leihpunkte [REDACTED]			80133
02.11.2011	[REDACTED] Tilgung Rest	2111,13		0
02.11.2011	[REDACTED] ?bertrag Leihpunkte			24385
02.11.2011	[REDACTED] Einlage [REDACTED]		166,00	48
02.11.2011	[REDACTED] Tats. verbrauchte Leihpunkte			-57217
31.12.2011	[REDACTED] Umbuchung	32,12		0
31.12.2011	RDU Erstattung 2011		32,12	0
07.03.2013	Punkte von 31.12.2011 bis 2013-03-07			346

Abb. 98: Die Ansicht eines Kontoauszuges

Bei dem Kontoauszug musste darauf geachtet werden, dass die selben Buchungen angegeben wurden die man auch in der Seite vor dem Klick auf den Drucken-Button angezeigt hatte. Dies wurde damit gelöst, indem man das Von-Datum und das Bis-Datum als Parameter übergab und mit diesen wieder die Buchungen rekonstruierten konnte.

11.1.7 Implementierung der View

In der View musste die Möglichkeit ein Anfangs- und ein Enddatum anzugeben implementiert werden. Um die Eingabe eines Datums im richtigen Format zu erleichtern wurde ein Datepicker eingesetzt. Der Datepicker sorgt zudem dafür dass das Datum im europäischen Format angezeigt wird.



Abb. 99: Der Datepicker zur vereinfachten Eingabe eines Datums

Um einen Datepicker in diesem Format zu erhalten musste ein jquery-script eingefügt werden, dass dann mit den beiden Feldern „vonDatum“ und „bisDatum“ verbunden wurde.

```
<script>
$(function() {
    $('#vonDatum').datepicker({
        dateFormat: 'dd.mm.yy',
        changeMonth: true,
        changeYear: true
    });
    $('#bisDatum').datepicker({
        dateFormat: 'dd.mm.yy',
        changeMonth: true,
        changeYear: true
    });
})
</script>
```

Abb. 100: Die Implementierung des Jquery-scripts zur Erzeugung des Datepickers

Für einen Kontoauszug sollte ein Button am unteren Bildschirmrand eingefügt werden, bei einer Betätigung des Buttons wurde, wie oben beschrieben, die Weiterleitung zur Ansicht des Kontoauszugs ausgeführt.



Abb. 101: Die Ansicht des Buttons zur Erstellung des Kontoauszuges

12 Darlehensvertrag berechnen (M. Hoprich)

Bei der Berechnung des Darlehensvertrags sollte es einem Mitglied ermöglicht werden ein Formular über eine Zusatzentnahme elektronisch auszufüllen. Am Anfang der Berechnung musste ein Mitglied seine Mnr. angeben um automatisch die jeweiligen Daten aus der Datenbank zu bekommen. Nach dem Ausfüllen des Formulars wurden nach dem Klicken auf den „Berechne“-Button sämtliche Konditionen angezeigt. Bei dieser Art der elektronischen Berechnung der Konditionen kann das Mitglied seine Eingaben verändern, sollten die resultierenden Konditionen nicht seinen Wünschen entsprechen, indem die zuvor eingegebenen Werte wieder neu eingegeben werden und somit neue Konditionen berechnet werden.

12.1 Umsetzung Darlehensvertrag

12.1.1 Anlegen des Controllers und der View

Es mussten am Anfang ein neuer Controller und eine neue View eingefügt werden.

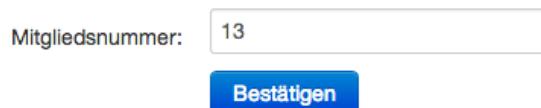
```
rails generate controller Darlehensvertrag new
```

Mit dieser Codezeile wurden die beiden Dateien „darlehensvertrag_controller.rb“ und „new.html.erb“ erzeugt, in denen dann die Implementierung erfolgte. Zudem musste noch eine weitere View zum Anzeigen der Konditionen eingefügt werden „anzeigen.html.erb“, da in der View „new.html.erb“ nur die Eingabe der Mitgliedernummer erfolgte.

12.1.2 Erstellen der Eingabemöglichkeit

Der erste Bildschirm, der zu der Berechnung der Konditionen angezeigt wurde, ist der in der ein Mitglied seine Mnr. angeben muss.

Neuen Darlehensvertrag berechnen



Mitgliedsnummer:

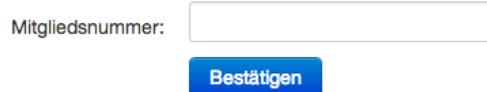
Bestätigen

Abb. 102: Das Eingabeformular für die Mnr

Ist die Mnr. nicht in der Datenbank vorhanden, so wird eine Fehlermeldung angezeigt, die eine erneute Eingabe der Mnr. verlangt.

Es konnte keine Suche gestartet werden:

- Die angegebene Mitgliedsnummer konnte nicht gefunden werden.



Mitgliedsnummer:

Bestätigen

Abb. 103: Die Fehlermeldung bei nicht in der Datenbank vorhandener Mnr.

Um es zu ermöglichen alle Werte der Konditionen zu berechnen musste, für jede benötigte Eingabe ein Textfeld eingefügt werden. Die für die Berechnung notwendigen Felder waren das „Heutige Datum“, das „Auszugs Datum“, die „Höhe der Zusatzentnahme“, die „Eigenen Sparpunkte“, die „Schenkungspunkte“, das „Sparguthaben“, das „Gewünschte Auszahldatum“, die „Tilgungszeit in Jahren“ und das „Datum der ersten Tilgungsrate“. Nachdem das Mitglied alle diese Werte ausgefüllt hatte, konnte es mit einem Klick auf den „Berechnen“ Button die Berechnung durchführen.

Neuen Darlehensvertrag berechnen

Mitgliedsnummer:	<input type="text" value="13"/>		
Vorname:	<input type="text" value="██████████"/>	Sparpunkte bis Auszahlungsdatum:	<input type="text"/>
Nachname:	<input type="text" value="██████████"/>	Leihpunkte auszahlung bis 1. Rate:	<input type="text"/>
Heutiges Datum:	<input type="text" value="07.03.2013"/>	Leihpunkte Tilgungszeitraum:	<input type="text"/>
Auszugs Datum:	<input type="text"/>	Leihpunkte gesammt:	<input type="text"/>
Sparguthaben:	<input type="text"/>	Nachsparpunkte:	<input type="text"/>
Eigene Sparpunkte:	<input type="text"/>	Nachsparbetrag:	<input type="text"/>
Schenkungspunkte:	<input type="text"/>	Monatliche Darlehenstilgung:	<input type="text"/>
Höhe der Zusatzentnahme:	<input type="text"/>	Monatlicher Betrag Nachsparen:	<input type="text"/>
Gewünschtes Auszahldatum:	<input type="text"/>	Risikodekungsbeitrag:	<input type="text"/>
Tilgungszeit in Jahren:	<input type="text"/>	Kostendeckungsumlage:	<input type="text"/>
Nachsparzeit in Jahren:	<input type="text"/>	Monatliche Gesamtbelastrung:	<input type="text"/>
Datum der ersten Tilgungsrate:	<input type="text"/>		
<input type="button" value="Berechnen"/>			

Abb. 104: Ansicht des Formulars zur Berechnung eines ZE-Vertrages

12.1.3 Berechnung der Werte

Bevor die Berechnung durchgeführt wurde, musste zuerst die Eingabe des Mitglieds auf Korrektheit überprüft werden. Wenn einer der eingegebenen Werte nicht korrekt war, so wurde eine entsprechende Fehlermeldung ausgegeben mit der Bitte den entsprechenden Wert zu korrigieren.

Neuen Darlehensvertrag berechnen

Es konnte keine Suche gestartet werden:

- Geben sie bitte eine Korrekte Zusatzentnahme ein
- Geben sie bitte die Eigenen Sparpunkte korrekt ein
- Geben sie bitte die Schenkungspunkte korrekt ein
- Geben sie bitte ein Valides Gewuenschtes Auszahlungsdatum ein
- Geben sie bitte die Tilgungszeit in Jahren korrekt ein
- Geben sie bitte das Datum der 1. Tilgungsrate korrekt ein
- Geben sie bitte das Sparguthaben korrekt ein
- Geben sie bitte das heutige Datum korrekt ein
- Geben sie bitte das Auszugs Datum korrekt ein

Mitgliedsnummer:	<input type="text" value="13"/>	Sparpunkte bis Auszahlungsdatum:	<input type="text"/>
Vorname:	<input type="text"/>	Leihpunkte auszahlung bis 1. Rate:	<input type="text"/>
Nachname:	<input type="text"/>	Leihpunkte Tilgungszeitraum:	<input type="text"/>
Heutiges Datum:	<input type="text"/>	Leihpunkte gesammelt:	<input type="text"/>
Auszugs Datum:	<input type="text"/>		

Abb. 105: Alle Fehleingaben auf einmal

Waren die Werte jedoch korrekt, konnte die Berechnung der Werte anhand von Formeln gemacht werden, die aus einem Excel-Dokument das von Herrn Kienle zur Verfügung gestellt wurde entnommen wurden.

Die Formeln sahen wie folgt aus:

Das Datum der letzten Tilgungsrate entspricht dem Datum an dem die Zusatzentnahme komplett zurückbezahlt wurde:

$$\text{DatumErsteTilgungsrate} + \text{TilgungszeitInJahren} \times 365 - 30$$

Das Datum der frühestmöglichen Entnahme entspricht dem Datum, an dem man seinen Nachsparbetrag komplett eingezahlt hat.

$$\text{DatumErsteTilgungsrate} + \text{NachsparzeitInJahren} \times 365$$

Sparpunkte bis heute:

$$SparpunkteInsgesamt + \frac{(HeutigesDatum - AuszugsDatum) * SaldoSparguthaben}{30} * Kontenklasse$$

Die Sparpunkte bis zum Auszahldatum sind die bereits angesparten Punkte des Mitglieds.

$$SparpunkteBisHeute + \frac{(GewünschtesAuszahldatum - HeutigesDatum) * SaldoSparguthaben}{30} * Kontenklasse$$

Leihpunkte bei der Auszahlung zur 1. Rate:

$$\frac{(DatumErsteTilgungsrate - Auszahlungsdatum) \times Darlehen}{30}$$

Leihpunkte im Tilgungszeitraum:

$$\frac{TilgungszeitInJahren \times 12 \times Darlehen}{2}$$

Die gesamten Leihpunkte entsprechen der Anzahl der Punkte die man während der Zusatzentnahme zurückzahlen muss.

$$LeihpunkteTilgungszeitraum + LeihpunkteAuszahlungBis1.Rate$$

Die Nachsparpunkte entsprechen der Anzahl an Punkten die man nach erfolgreichem Abschließen der Zusatzentnahme wieder ansparen muss:

$$LeihpunkteGesamt - SparpunkteBisAuszahldatum$$

Nachsparbetrag:

bei der Berechnung des Nachsparbetrages gab es zwei Fälle zu unterscheiden bei denen sich die Berechnung minimal unterschied.

Für den Fall, dass das erforderliche Darlehen größer als $\frac{2 \times Nachsparpunkte}{12 \times NachsparzeitInJahren}$ ist:

$$\frac{\text{ErforderlichesDarlehen}}{\frac{\text{Kontenklasse}}{100}}$$

Für den Fall, dass das erforderliche Darlehen größer ist:

$$\frac{\left(\frac{2 \times Nachsparpunkte}{12 \times NachsparzeitInJahren}\right)}{\left(\frac{\text{Kontenklasse}}{100}\right)}$$

Die monatliche Darlehenstilgung ist der Betrag den man jeden Monat einzahlen muss bis die Zusatzentnahme getilgt ist:

$$\frac{\text{ErforderlichesDarlehen}}{12 \times \text{TilgungszeitInJahren}}$$

Monatlicher Betrag Nachsparen:

$$\frac{\text{Nachsparbetrag}}{12 \times \text{NachsparzeitInJahren}}$$

Risikodeckungsbeitrag mtl.:

$$\frac{\text{ErforderlichesDarlehen} \times \text{RDU}}{12}$$

Kostendeckungsumlage mtl.:

$$\frac{\text{ErforderlichesDarlehen} \times \text{KDU}}{12}$$

Bei den Werten RDU und KDU musste der entsprechende Wert aus der Tabelle Umlage entnommen werden, die neu angelegt wurde. Damit dies funktionieren konnte musste zuvor noch ein neues Model mit dem Dateinamen „Umlage.rb“ in das Projekt aufgenommen werden. Danach konnte mit diesen beiden Zeilen Code

```
@rdu = Umlage.where("Jahr = ?", @tilgZeitInJahren.to_i).first.RDU / 100
@kdu = Umlage.where("Jahr = ?", @tilgZeitInJahren.to_i).first.KDU / 100
```

Abb. 106: Codezeilen zum ermitteln der RDU bzw. der KDU

die jeweilige RDU bzw. die KDU der Tabelle entnommen werden.

Die Monatliche Gesamtbelaistung ist der Betrag der anfällt bis man die Zusatzentnahme und das Nachsparen beendet hat:

$$\begin{aligned} \text{MonatlicheDarlehenstilgung} &+ \text{MonatlicherBetragNachsparen} \\ &+ \text{Risikodeckungsbeitrag} \end{aligned}$$

Nachdem die Konditionen berechnet wurden, konnte man sie im Formular auf der rechten Seite sehen. Zudem war es möglich die Berechnung jederzeit neu zu starten, nachdem man neue Werte eingegeben hatte. Dies ermöglicht es einem Mitglied sich einen Vertrag mit Konditionen nach seinen Wünschen zu erstellen.

Neuen Darlehensvertrag berechnen

Mitgliedsnummer:	13	
Vorname:	[REDACTED]	Sparpunkte bis Auszahlungsdatum: 2.850
Nachname:	[REDACTED]	Leihpunkte auszahlung bis 1. Rate: 46.000
Heutiges Datum:	07.03.2013	Leihpunkte Tilgungszeitraum: 120.000,0
Auszugs Datum:	01.03.2013	Leihpunkte gesammt: 166.000,0
Sparguthaben:	1000	Nachsparpunkte: 163.150,0
Eigene Sparpunkte:	100	Nachsparbetrag: 7251,11
Schenkungspunkte:	2000	Monatliche Darlehenstilgung: 104,17
Höhe der Zusatzentnahme:	5000	Monatlicher Betrag Nachsparen: 120,85
Gewünschtes Auszahldatum:	31.03.2013	Risikodekungsbeitrag: 4,58
Tilgungszeit in Jahren:	4	Kostendeckungsumlage: 0,00
Nachsparzeit in Jahren:	5	Monatliche Gesamtbelastung: 229,60
Datum der ersten Tilgungsrate:	01.01.2014	
Berechnen		

Abb. 107: Ein mit Werten gefülltes Formular und die entsprechenden Ergebnisse

13 Fazit (M. Hoprich, S. Stroh und N. Usow)

Die Projektarbeit hat uns eine Menge Spaß gemacht, auch wenn die Einarbeitung in Ruby-On-Rails viel an Zeit gekostet hat und es am Anfang ein wenig schwer gefallen ist, einen Einblick in die bestehenden Geschäftsprozesse der o/ZB zu bekommen. Die Schwierigkeiten bzgl. der Einarbeitung in Ruby-On-Rails röhren hauptsächlich daher, weil Rails ein Framework ist, bei dem es Abhängigkeiten und Automatismen gibt die man nicht auf Anhieb versteht, die aber im Hintergrund erfolgreich ihren Dienst verrichten. Hat man aber erst mal die Eigenheiten des Rails-Frameworks verstanden, so stellt dieses einen deutlichen Mehrwert im raschen Entwicklungsprozess dar. Was die Basis der Applikation aufbauend auf Rails betrifft, haben unsere Vorgänger in ihren Aufgabenteilen schon eine gute Vorarbeit geleistet. Daher möchten wir für ausführlichere Informationen zum Rails-Framework auf die Dokumentation unserer Vorgänger vom SS12 verweisen.

Die Umsetzung unserer Vorgänger haben wir aber in einigen Teilen nicht nur erweitert sondern auch zum Teil neuimplementieren müssen. Ein Problem mit dem wir ständig zu kämpfen hatten waren ständige Änderungen an der Struktur der Datenbank, da diese uns nicht rechtzeitig oder gar nicht mitgeteilt worden sind und wir somit meist gegen die ältere Datenbankstrukturen implementiert haben. Dadurch mussten einige Funktionen wieder neu implementiert oder korrigiert werden, um somit auf der aktuellen Datenbankstruktur erfolgreich arbeiten zu können. Es ist dadurch sehr vieles an Arbeit entstanden, die einfach nicht in eine Dokumentation aufgenommen werden kann, aber dennoch getan werden musste um den bestehenden Fortbestand der Applikation am Laufen zu erhalten. Diese Arbeit war teilweise sehr zeitintensiv. Aus diesem Grund ist für uns ein Gefühl entstanden, dass wir neben unseren Hauptaufgaben noch zusätzlich viel an Bugfixing betrieben haben.

Eine sehr gute Unterstützung bei unseren Aufgaben waren die Feedbacks von Herrn Kienle in Form von Testberichten. Diese wurden immer fortgeschrieben und schilderten das zu erwartende Verhalten der Applikation beim Auslösen einer Aktion. Sehr unterstützend waren die darin enthaltenen Bilder, da diese sehr schnell den Sachverhalt verdeutlichen konnten und somit auch deutlich zu einer rascheren Implementierung geführt haben. Gut war an dem Testbericht außerdem, dass er an alle Projektteilnehmer ging und sich somit jeder seinen zugewiesenen Aufgabenteil raussuchen konnte. Ein anderer sehr positiver Aspekt war, dass uns bei aufkommenden Fragen Herrn Kienle stets mit Rat und Tat zur Seite stand.

Die Durchführung dieser Projektarbeit hat uns auch einen Einblick in das Projektleben im möglichen Berufsalltag gegeben, von dem wir auch in Zukunft profitieren werden. Wir möchten uns für gute Zusammenarbeit bei Herrn Kienle, Herrn Kleinert und Herrn Schaefer bedanken, die uns bei Fragen und Problemen rund um das Projekt unterstützt haben.

14 Abbildungsverzeichnis

Abb. 1: Ansicht "Passwort ändern"	7
Abb. 2: Pfad „Passwort ändern“	8
Abb. 3: Passwort Bestätigung.....	8
Abb. 4: Überschreiben After_Update	8
Abb. 5: Validierung des Passwortes	9
Abb. 6: Tabelle Veranstaltung	10
Abb. 7: Ansicht neue Veranstaltung	11
Abb. 8: Auflistung Teilnahmen	12
Abb. 9: Teilnehmer-Statistik	13
Abb. 10: Teilnahmen eines Mitglieds	14
Abb. 11: DB-Tabelle Veranstaltung.....	14
Abb. 12: Link zur Veranstaltung.....	15
Abb. 13: Stylesheet der Tabelle	15
Abb. 14: JavaScript Function „submit_form_delete“	15
Abb. 15: JavaScript Function „bearbeiten“	16
Abb. 16: ID-Zuweisung Tabelleneintrag.....	16
Abb. 17: Routes Veranstaltung	17
Abb. 18: Tabelle Sonderberechtigungen	18
Abb. 19: Neue Sonderberechtigung.....	19
Abb. 20: Liste Geschäftsprozesse	20
Abb. 21: Javascript Function „submit_form“	22
Abb. 22: Berechtigung Darstellung	22
Abb. 23: Routes Sonderberechtigungen	22
Abb. 24:Aktuelle Bürgschaften	23
Abb. 25:Historisierte Bürgschaften	24
Abb. 26: Neue Bürgschaft.....	24
Abb. 27:Bürgschaft editieren	25
Abb. 28: Routes Bürgschaften	27
Abb. 29: Dynamische Collection Javascript	27
Abb. 30: Textauswahl.....	28
Abb. 31: Zusammenhängende Pakete	28
Abb. 32: Text „keine Bürschaften“	28
Abb. 33: Original-Datensatz in der Tabelle Adresse welcher verändert wird	31
Abb. 34: Erster Datensatz wurde historisiert. Das Straße-Attribut wurde geändert.	32
Abb. 35: Zweiter Datensatz wurde historisiert. Das Vermerk-Attribut wurde geändert.....	32
Abb. 36: Callback „before_update“	33
Abb. 37: Callback: „after_update“	33
Abb. 38: Methode „Adresse.get“	34
Abb. 39: Stabile serverseitige Sortierung nach Name.....	34
Abb. 40: Zur Sortierung werden GET-Parameter an die URL angehängt, siehe rote Umrandung	35
Abb. 41: Durchreichung der Parameter in die Pagination, siehe rote Umrandung.....	35
Abb. 42: sort_column und sort_direction in verwaltung_controller.rb (Controller)	36
Abb. 43: listOZBPersonen in verwaltung_controller.rb (Controller)	36
Abb. 44: sortable in application_helper.rb (Helper)	37
Abb. 45: Ausgabe von sortable in listOZBPersonen.html.erb (View)	37
Abb. 46: Benutzer die irgendwo im Vor- oder Nachnamen die Zeichenfolge „schm“ enthalten	38
Abb. 47: searchOZBPerson in verwaltung_controller.rb (Controller)	38
Abb. 48: Suchfeld in searchOZBPerson.html.erb (View)	38
Abb. 49: Überführung der Ruby-Objekte nach JSON in searchOZBPerson.html.erb (View).....	39
Abb. 50: HTML-Inputfeld wird zur Suche transformiert in searchOZBPerson.html.erb (View)	39
Abb. 51: Suchergebnisse werden als eine Liste präsentiert in searchOZBPerson.html.erb (View)	39
Abb. 52: Weiterer Einsatz der Schnellsuche in der Veranstaltungsverwaltung.....	39
Abb. 53: Benutzer mit den Rechten Öffentlichkeits- und Projekterverwaltung.....	40
Abb. 54: Auszug aus den Geschäftsprozessen inkl. der Gruppenberechtigungen	40
Abb. 55: Dem Benutzer wird das Recht Finanzenverwaltung vergeben. Rechts werden die möglichen Rechte als HTML-Liste für diese Gruppe angezeigt.....	41
Abb. 56: Falls die selbe Berechtigung bereits im System vorhanden ist.....	41
Abb. 57: Existiert das Recht schon in verwaltung_controller.rb (Controller)	42

Abb. 58: Generiert eine HTML-Liste zu der Berechtigungsgruppe in editBerechtigungen.html.erb (View)	42
Abb. 59: Gibt die generierten HTML-Listen aus in editBerechtigungen.html.erb (View)	42
Abb. 60: application.css (CSS-Stylesheet). Markiert den Berechtigungs-Kontainer initial als nicht sichtbar	42
Abb. 61: Ein- und Ausblenden der HTML-Liste mit Berechtigungen in editBerechtigungen.html.erb (View)	43
Abb. 62: Ehemalige Form der Dateneingabe. Die Eingabefelder waren unabhängig über mehrere Tabs verteilt	43
Abb. 63: Neue übersichtliche Form der Dateneingabe	44
Abb. 64: Zusammengeführte Personal- und Kontaktdaten umgeben von einem Zeilenkontainer	44
Abb. 65: Überprüfung auf ein gültiges Objekt, und schlussendliche Konvertierung nach DD.MM.YYYY	45
Abb. 66: Die Auswahl der Jahre und Monate ist bequem per Auswahldialog möglich	45
Abb. 67: Erweitert den Kalender um die Umschaltung zwischen den Jahren und den Monaten	45
Abb. 68: Horizontales Scrollen innerhalb der tabellarischen ZE-Konto-Darstellung	46
Abb. 69: Tabelle wird in ein Container-Element inkludiert	46
Abb. 70: Ausrichtung inkl. der Formatierung der Zahlenwerte mit „number_with_precision“	48
Abb. 71: Dieser Benutzer hat bereits ein EE-Konto mit einer Bankverbindung hinterlegt, daher werden nun die Felder vorausgefüllt	49
Abb. 72: ozb_konto_controller.rb (Controller)	49
Abb. 73: Bankverbindung	50
Abb. 77: Hilfsmethode "count_days_exact"	52
Abb. 78: Punkteberechnung-Klasse wird importiert	52
Abb. 79: CSV-Struktur aus dem produktiven Finanzbuchhaltungssystem	53
Abb. 80: Wegen Inkompatibilitäten beim Encoding muss noch das "b"-Flag (binary) bei Öffnen der Datei gesetzt werden	53
Abb. 81: Das Symbol :col_sep erwartet nun das gewünschte Trennzeichen	53
Abb. 82: Jede Zeile der CSV-Datei wird durchlaufen (Code ist zusammengeklappt)	54
Abb. 83: Die Spalten innerhalb einer Zeile werden Variablen zugewiesen	54
Abb. 84: Abhängig von der Zeichenlänge der Soll- und Habenkontonummer werden verschiedene Importlogiken angestoßen (Code ist zusammengeklappt)	55
Abb. 85: (Code ist zusammengeklappt)	55
Abb. 86: Abbuchung-Leihpunkte-Buchung, Storno-Abbuchung-Leihpunkte-Buchung, Punkteüberweisung-Buchung oder Konto-zu-Konto-Buchung (Code ist zusammengeklappt)	55
Abb. 87: Implementierung einer Abbuchung-Leihpunkte-Buchung	56
Abb. 88: Gewöhnliche Buchung (Soll- oder Habenbuchung) (Code ist zusammengeklappt)	57
Abb. 89: Beschreibt den Link auf einem EE-Konto mit der KontoNr. als Parameter	58
Abb. 90: Beschreibt den Link auf einem ZE-Konto mit der KontoNr. als Parameter	58
Abb. 91: Ein Konto mit eingefügten Link	58
Abb. 92: Die in Rails implementierte SQL-Abfrage um die zehn letzten Buchungen zu bekommen	59
Abb. 93: Die Implementierung zur Bestimmung des KKLs	59
Abb. 94: Der Kontoauszug für die ersten zehn Buchungen eines Kontos	60
Abb. 95: Die Implementierung der Formatprüfung der beiden Daten	61
Abb. 96: Die Fehlermeldung bei falscher Eingabe eines Datums	61
Abb. 97: Die SQL-Abfrage zu den Buchungen in dem Datumsintervall	61
Abb. 98: Die Warnung wenn es noch keine getätigten Buchungen in dem angegebenen Zeitraum gab	61
Abb. 99: Die Buchungen in dem angegebenen Zeitraum	62
Abb. 100: Die Implementierung um den Vor- und den Nach-Namen des Mitglieds zu bekommen	62
Abb. 101: Die Ansicht eines Kontoauszuges	63
Abb. 102: Der Datepicker zur vereinfachten Eingabe eines Datums	63
Abb. 103: Die Implementierung des Jquery-scripts zur Erzeugung des Datepickers	64
Abb. 104: Die Ansicht des Buttons zur Erstellung des Kontoauszuges	64
Abb. 105: Das Eingabeformular für die Mnr	65
Abb. 106: Die Fehlermeldung bei nicht in der Datenbank vorhandener Mnr	65
Abb. 107: Ansicht des Formulars zur Berechnung eines ZE-Vertrages	66
Abb. 108: Alle Fehleingaben auf einmal	67
Abb. 109: Codezeilen zum ermitteln der RDU bzw. der KDU	69
Abb. 110: Ein mit Werten gefülltes Formular und die entsprechenden Ergebnisse	70