**fit4IPA**

Probe-IPA BbcBank

**Leo Rudin**

**31. Januar 2017**

Durchführungsort:

**ICT Berufsbildungscenter Altstetten**

**Inhalt**

[1 Umfeld und Ablauf 4](#_Toc473644032)

[1.1 Aufgabenstellung 4](#_Toc473644033)

[1.1.1 Funktionale Anforderungen 4](#_Toc473644034)

[1.1.1.1 Präsentationsschicht 4](#_Toc473644035)

[1.1.1.2 Bussiness-Schicht 5](#_Toc473644036)

[1.1.1.3 Persistenz-Schicht 6](#_Toc473644037)

[1.1.2 Nicht funktionale Anforderungen 7](#_Toc473644038)

[1.1.3 Mittel und Methoden 7](#_Toc473644039)

[1.1.3.1 Tools und Technologien 7](#_Toc473644040)

[1.1.3.2 Methodik 8](#_Toc473644041)

[1.2 Projektaufbauorganisation 8](#_Toc473644042)

[1.3 Vorkenntnisse 8](#_Toc473644043)

[1.4 Vorarbeiten 8](#_Toc473644044)

[1.5 Firmenstandards 9](#_Toc473644045)

[1.6 Zeitplan 10](#_Toc473644046)

[1.7 Termine 11](#_Toc473644047)

[1.8 Methodik 11](#_Toc473644048)

[1.9 Arbeitsprotokoll 12](#_Toc473644049)

[1.9.1 16. Januar 2017 12](#_Toc473644050)

[1.9.2 17. Januar 2017 13](#_Toc473644051)

[1.9.3 18. Januar 2017 14](#_Toc473644052)

[1.9.4 23. Januar 2017 15](#_Toc473644053)

[1.9.5 24. Januar 2017 17](#_Toc473644054)

[1.9.6 25. Januar 2017 19](#_Toc473644055)

[1.9.7 30. Januar 2017 20](#_Toc473644056)

[1.9.8 31. Januar 2017 21](#_Toc473644057)

[2 Projekt 22](#_Toc473644058)

[2.1 Konzept 22](#_Toc473644059)

[2.1.1 Moqups 22](#_Toc473644060)

[2.1.1.1 Ein- und Auszahlung 22](#_Toc473644061)

[2.1.1.2 Konto Auszug anzeigen 23](#_Toc473644062)

[2.1.2 Use Case Diagramm 23](#_Toc473644063)

[2.1.3 Activity Diagramm 24](#_Toc473644064)

[2.1.3.1 Ein- und Auszahlung 24](#_Toc473644065)

[2.1.3.2 Kontoauszug anzeigen 25](#_Toc473644066)

[2.1.4 ERM Diagramm 25](#_Toc473644067)

[2.2 Implementierung 27](#_Toc473644068)

[2.2.1 Präsentationsschicht 27](#_Toc473644069)

[2.2.1.1 Default-Template 27](#_Toc473644070)

[2.2.1.2 Glyphicons von Bootstrap 28](#_Toc473644071)

[2.2.1.3 Screenshots 29](#_Toc473644072)

[2.2.2 Business-Schicht 30](#_Toc473644073)

[2.2.3 Persistenz-Schicht 31](#_Toc473644074)

[2.2.3.1 ManyToOne und OneToMany 32](#_Toc473644075)

[2.2.3.2 Named Querys 33](#_Toc473644076)

[2.2.3.3 Transient 33](#_Toc473644077)

[2.2.4 Validation 34](#_Toc473644078)

[2.2.5 Logging 36](#_Toc473644079)

[2.3 Testing 38](#_Toc473644080)

[2.3.1 JUnit Testing 38](#_Toc473644081)

[2.3.2 Testfälle 39](#_Toc473644082)

[2.3.3 Testprotokoll 42](#_Toc473644083)

[3 Anhang 44](#_Toc473644084)

[3.1 Quellen 44](#_Toc473644085)

[4 Schlusswort 45](#_Toc473644086)

# Umfeld und Ablauf

Für den Lehrabschluss muss eine «Individuelle Praxis Arbeit» (kurz: IPA) durchgeführt werden. Der Lernende soll diese selbständig durchführen und erhält dabei die Aufgabenstellung vom Fachvorgesetzten. Diese Aufgabenstellung wird vom Experten überprüft und validiert. Während der Durchführung stattet der Experte dem Lernenden und Fachvorgesetzten zwei Besuche ab, um sicherzustellen, dass alles korrekt abläuft.

## Aufgabenstellung

Die BbcBank modernisiert ihre Bank-Applikation aus den 90er-Jahren. Technologien, wie CGI (Common Gateway Interface) und CORBA werden abgelöst durch Java Enterprise. Objektorientierte Anwendungen ersetzen prozedurale Programm-Logik.

### Funktionale Anforderungen

#### Präsentationsschicht

Das User Interface für den Use Case Konto-Auszug anzeigen besteht aus zwei HTML-Seiten:

* Kunde authentifizieren (nicht Teil von fit4IPA)
* Konto-Auszug anzeigen

Die Seite für die Anzeige des Konto-Auszugs umfasst folgende Angaben:

* Adresse des im ersten Arbeitsschritt authentifizierten Kunden
* Tabellarische Darstellung des Konto-Auszugs mit einer Zeile pro Transaction-Record:
  + Kolonne 1: Datum (TT.MM.JJJJ)   
    Konto-Auszug aufsteigend nach Datum sortiert
  + Kolonne 2: Informationen
  + Kolonne 3: Belastungen (Transaction-Typ D)
  + Kolonne 4: Gutschriften (Transaction-Typ C)
  + Kolonne 5: Kontostand

Das User Interface für den Use Case Konto Einzahlung/Auszahlung besteht aus zwei HTML-Seiten:

* Kunde suchen (nicht Teil von fit4IPA)
* Einzahlung/Auszahlung durchführen

Die Seite für die Durchführung der Einzahlung/Auszahlung umfasst folgende Angaben:

* Adresse des im ersten Arbeitsschritt selektierten Kunden
* Liste zum Selektieren des Kunden-Kontos
* Checkbox zum Selektieren des Transaktions-Typs (Einzahlung/Auszahlung)
* Eingabefeld für den Transaktions-Betrag (immer CHF)
* Eingabefeld für den Transaktions-Text
* Button zum Auslösen der Transaktion
* Status-Feld für die Resultatanzeige:
  + Bestätigung erfolgreiche Transaktions-Durchführung mit Anzeige des neuen Saldos
  + ungültige Kontonummer
  + bei Auszahlung: Transaktion wegen Überschreiten des Kredit-Limits nicht durchgeführt mit Anzeige des aktuellen Saldos

Die Präsentations-Komponenten werden mit JSF entwickelt.

#### Bussiness-Schicht

Für den Use Case *Konto-Auszug anzeigen* sind folgende Business-Services sind zu entwickeln:

Kundendaten holen

|  |  |
| --- | --- |
| Request: | Kundennummer [999-999999] |
| Response: | Kundendaten (ohne Kontoangaben)  Exception im Fehlerfall |

Kontonummern des Kunden holen

|  |  |
| --- | --- |
| Request: | Kundennummer [999-999999] |
| Response: | Liste mit den Kontonummern des Kunden (IBAN)  Exception im Fehlerfall |

Kontodaten mit Transaction-Records holen

|  |  |
| --- | --- |
| Request: | Kontonummer (IBAN) |
| Response: | Kontodaten  Transaction-Records aufsteigend sortiert nach Transaction Zeitpunkt  Exception im Fehlerfall |

Für den optionalen Use Case *Konto Einzahlung/Auszahlung* sind folgende Business-Services zu entwickeln:

Einzahlung durchführen  
Schreibt den Transaction-Record mit Typ C (Gutschrift) , erhöht den Konto-Saldo

|  |  |
| --- | --- |
| Request: | Kontonummer (IBAN)  Betrag (in CHF)  Text |
| Response: | Neuer Saldo  Exception im Fehlerfall |

Auszahlung durchführen  
Stellt sicher, dass das Konto-Limit nicht überschritten wird. Schreibt den Transaction-Record

mit Typ D (Belastung), reduziert den Konto-Saldo

|  |  |
| --- | --- |
| Request: | Kontonummer (IBAN)  Betrag (in CHF)  Text |
| Response: | Neuer Saldo  Exception im Fehlerfall |

Die Business-Komponenten werden mit EJB Web Service (SOAP) entwickelt.

#### Persistenz-Schicht

Die Persistenz-Schicht wird im Bottom-Up Verfahren entwickelt. Die Datenbank (bbcbank) istbereits erstellt und mit Testdaten initialisiert.

Die DB umfasst folgende Tabellen:

CUSTOMER:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kolonne | Typ | Länge | Index/Key  PK=Primär  SK=Sekundär  FK=Fremd | NULL erlaubt | Beschreibung |
| ID | INT |  | PK | Nein | Kunden-ID fortlaufende automatisch vom DB Server generierte Nummer |
| CUSTOMER\_NUMBER | VARCHAR | 10 | SK | Nein | Kundennummer [999-999999] |
| TITLE | VARCHAR | 10 |  | Nein | Anrede |
| FIRST\_NAME | VARCHAR | 30 |  | Nein | Vorname |
| LAST\_NAME | VARCHAR | 30 |  | Nein | Nachname |
| STREET | VARCHAR | 30 |  | Nein | Strasse |
| CITY | VARCHAR | 30 |  | Nein | Ort |
| ZIPCODE | SMALLINT |  |  | Nein | Postleihzahl |

ACCOUNT:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kolonne | Typ | Länge | Index/Key  PK=Primär  SK=Sekundär  FK=Fremd | NULL erlaubt | Beschreibung |
| ID | INT |  | PK | Nein | Kunden-ID fortlaufende automatisch vom DB Server generierte Nummer |
| IBAN | VARCHAR | 30 | SK | Nein | Kontonummer |
| BALANCE | DECIMAL | 9,2 |  | Nein | Saldo |
| OVERDRAFT | DECIMAL | 7,2 |  | Nein | Kredit-Limit |
| CUSTOMER\_ID | INT |  | FK | Nein | Kunden-ID |

TRANSACTION\_RECORDS:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kolonne | Typ | Länge | Index/Key  PK=Primär  SK=Sekundär  FK=Fremd | NULL erlaubt | Beschreibung |
| ID | INT |  | PK | Nein | Kunden-ID fortlaufende automatisch vom DB Server generierte Nummer |
| TYPE | CHAR | 1 |  | Nein | Typ  C = Gutschrift  D = Belastung |
| TEXT | VARCHAR | 30 |  | Nein | Text |
| AMOUNT | DECIMAL | 9,2 |  | Nein | Betrag |
| TRANSACTION\_TIME | TIMESTAMP |  |  | Nein | Transaction-Zeitpunkt |
| ACCOUNT\_ID | INT |  | FK | Nein | Konto-ID |

Die Persistenz-Komponenten werden mit JPA entwickelt.

### Nicht funktionale Anforderungen

Die wichtigen durchgeführten System-Aktivitäten sind mittels Java Logging API zu protokollieren.

### Mittel und Methoden

#### Tools und Technologien

Programmierung:

* Java SE 8u112
* Java EE 7 SDK Update 2  
  JSF (Präsentations-Schicht)  
  EJB (Business-Schicht)  
  JPA (Persistenz-Schicht)

IDE:

* Eclipse Neon mit GlassFish Tools und Subeclipse Plugin (Deployment)
* ArgoUML, Release 0.34

Team-Environment

* SVN

Design

* Visio 2017
* Balsamiq Mockups Version 3.1.2

Testing:

* JUnit 4

Applikationsserver

* Payara Server

Datenbank-Server

* MySQL 5.7.14
* MySQL Workbench 6.3.7
* Connector/J 5.1.39 (JDBC-Treiber)

#### Methodik

Wasserfall Methode oder agile Softwareentwicklung.

## Projektaufbauorganisation

|  |  |
| --- | --- |
| **Lernender** | Leo Rudin |
| **Fachvorgesetzter** | Christian Feuz |
| **Experte** | Christian Feuz |

## Vorkenntnisse

Zu Beginn meiner Lehrzeit habe ich zuerst ein Basislehrjahr bei der Santis Training AG besucht, bei dem ich mir die Grundkenntnisse der Programmierung mit PL/1 und Java angeeignet habe.

Während des zweiten Lehrjahrs war ich in einer PL/1-Abteilung. Dort konnte ich GUI’s mit REXX erstellen, sowie einige PL/1-Programme anpassen und erweitern, welche von meinen Vorgängern erstellt wurden.

Im dritten Lehrjahr besuchte ich während den ersten zwei Monaten einen Java EE Kurs, wobei wir hierbei mit JSF eine Webapplikation in zweier bis dreier Gruppen erstellen konnten. Danach kam ich in eine Java-Abteilung, welche Maintenance für das Online-Banking betreibt. Dort arbeite ich mit VBA sowie Java.

Im vierten Jahr bin ich zuständig für eine JSF Applikation und implementiere hier neue Anforderungen und fixe bestehende Bugs.

## Vorarbeiten

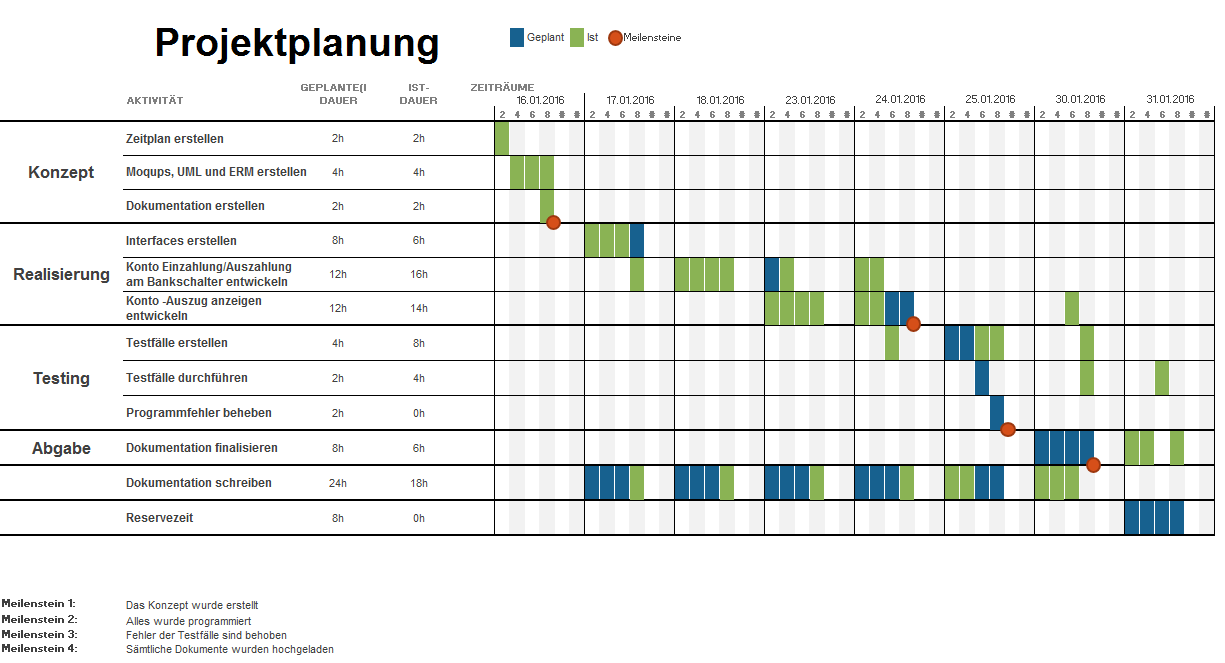
Für die IPA habe ich mein Arbeitsumfeld aufgesetzt. Dafür habe ich Java 8u112 installiert, sowie einen MySQL Server und weitere MySQL Komponenten, Eclipse und der Payara Server zur Ausführung des Codes.

Ich habe beim Payara Server bereits einen Connection Pool für die Datenbank erstellt. Schliesslich habe ich den das JSF Basisprojekt in den Eclipse Workspace importiert und das erstellte Projekt für die Datensicherung in ein SVN-Repository abgelegt.

## Firmenstandards

Da die IPA in einer Schulung durchgeführt wurde, existieren keine Firmenstandards.

## Zeitplan



## Termine

Die IPA wird an folgenden Daten durchgeführt:

* 16. Januar 2017 bis 31. Januar 2017

Expertenbesuche:

* 1. Besuch: 17. Januar 2017
* 2. Besuch: 25. Januar 2017

Präsentation:

* Mittwoch, 1. März 2017

## Methodik

Für meine IPA habe ich das Wasserfallmodell als meine Projektmethode gewählt. Dieses ist ein lineares Vorgehensmodell. Dieses wird in Phasen organisiert:

Anforderungen

Entwurf

Implementation

Testen

Support

Bei der Wasserfall-Methode soll man jeden Schritt zuerst komplett abschliessen, bevor zum nächsten Schritt übergegangen wird. Wenn zum nächsten Schritt übergegangen wird, dürfen erarbeitete Elemente aus den letzten Schritten nicht mehr bearbeitet werden.

## Arbeitsprotokoll

### 16. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Projekt/Codebase aufsetzten Ich habe das Beispielprojekt von Chris genommen und in Eclipse aufgesetzt und ausgeführt.* | L.R. | 0.0 | 0.5 |
| *Zeitplan erstellen Für die Zeitplanung habe ich einen Zeitplan erstellt.* | L.R. | 2.0 | 2.0 |
| *Moqups erstellen Ich habe für meine beiden Aufgabenstellungen die Moqups für die Views erstellt.* | L.R. | 2.0 | 2.0 |
| *Aktivitätsdiagramm, Use Case Diagramm und ERM erstellt Für beide Aufgabenstellungen habe ich für das Konzept zusätzlich die genannten Diagramme erstellt sowie das ERM für die Datenbankstruktur.* | L.R. | 2.0 | 2.5 |
| *Dokumentation erstellen Ich habe das Dokument für die Dokumentation erstellt und die Aufgabenstellung kopiert und das Konzept erstellt.* | L.R. | 2.0 | 2.0 |
| **Total** |  | **8.0** | **9.0** |
| **Probleme / Hilfestellungen** | | | |
| Der heutige Tag bestand aus der Erstellung von Konzepten. Bei Fragen zur Aufgabenstellung habe ich meinen Fachvorgesetzten gefragt, was genau gemeint ist. Ansonsten hatte ich keine nennenswerten Probleme bei der Erstellung der Konzepte. | | | |
| **Reflexion** | | | |
| Da ich an den ersten zwei Tagen der IPA im Militär war, ist heute der erste Tag meiner IPA. Letzten Montag haben wir alle Entwicklungsumgebungen installiert, um optimal in die IPA zu starten. Mein Plan für heute ist die Erstellung aller Konzepte. Dazu gehören die Moqups für die Präsentations-Schicht, das Use Case Diagramm, das Aktivitätsdiagramm und das ERM der Datenbank. Neben den Konzeptdokumenten muss ich auch noch das Dokument erstellen, welche die IPA-Dokumentation enthält, die Aufgabenstellung kopieren sowie die Konzepte in die Dokumentation kopieren und das Arbeitsjournal schreiben. | | | |
| **Nächste Schritte** | | | |
| * Beginn Phase Realisierung mit der Erstellung der Views für die Präsentationsschicht. | | | |

### 17. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *GUI’s erstellen Für meine Aufgabenstellungen habe ich beide GUI’s erstellt.* | L.R. | 8.0 | 5.0 |
| *Expertenbesuch Mein Experte kam das erste Mal vorbei.* | L.R. | 0.0 | 1.0 |
| *JPA Abstraktion erstellen Um im Java Teil mit der Datenbank zu interagieren habe ich die Datenbank mit JPA Klassen abgebildet.* | L.R. | 0.0 | 2.0 |
| **Total** |  | **8.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Bei der Erstellung der Views hatte ich wenig Mühe. Da ich mich mit Bootstrap und allgemein mit HTML und CSS schon ziemlich gut auskenne, gab es wenig Schwierigkeiten.  Da ich jedoch mit der Plain-Version von Bootstrap arbeite – das heisst, ich lade einfach nur das bootstrap.min.css und das bootstrap.min.js herunter und binde es ein – führte dies zu Komplikationen mit JSF. Für die Darstellung der Tabelle für die Transaktionen wollte ich das Datatables jQuery-Plugin verwenden. Dieses Plugin bietet auch Unterstützung von Bootstrap an. Bei der Einbindung und Erstellung einer Beispieltabelle ist mir dann aufgefallen, dass die Icons für das Sortieren fehlen. Der Grund war mir klar: Bootstrap sucht standardmässig automatisch nach einem «fonts» Ordner eine ebene über dem Platz, an dem sich das CSS von Bootstrap befindet. Jedoch rendert JSF die Locations für das CSS und JS anders als normal. Dadurch kann durch simples (../fonts) nicht die Location des «fonts» Ordner angesprochen werden.  Als Lösung gibt es für diese Problematik gibt es von JSF ein spezieller Tag, welcher den Pfad des angesprochenen Ordners zurückgibt. Daher muss ich die alle Pfade im bootstrap.min.css zum «fonts» Ordner mit dem JSF-Tag ersetzten. | | | |
| **Reflexion** | | | |
| Den heutigen Tag habe ich völlig dem Erstellen der GUI’s gewidmet. Die beiden GUI’s habe ich jedoch schon gegen den Nachmittag fertiggemacht und konnte bereits mit der Erstellung des JPA Teils beginnen. Um 14:00 Uhr hatte ich meinen ersten Expertenbesuch. Dafür sollte ich bereits mein Arbeitsjournal für den ersten Tag fertig haben, sowie eventuell das Arbeitsjournal des zweiten Tages begonnen haben. Der Experte stellte sich vor und liess mich ihm vorstellen. Er gab mir administrative Informationen über den weiteren Ablauf, den nächsten Besuch und die IPA-Präsentation. Zusätzlich interessierte er sich für den Stand meiner Arbeit, mein Arbeitsjournal und war offen für Fragen und Probleme jeglicher Art. Er verfolgte dabei eine Checkliste für Experten.  Gegen den Ende des Tages konnte ich noch die gesamte Datenbank im JPA Teil abbilden. | | | |
| **Nächste Schritte** | | | |
| * Fortfahren mit der Phase «Realisierung» und mit der Erstellung der Business-Logik sowie der Beans im JSF Teil. | | | |

### 18. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Serviceklassen erstellen Für die Business-Logik habe ich die Service Klassen vorbereitet, welche die Business-Logik enthalten sollen.* | L.R. | 8.0 | 5.0 |
| *Named Querys erstellen Für den Kontoauszug muss ich mindestens drei Querys erstellen, mit denen die erforderlichen Daten selektiert werden.* | L.R. | 0.0 | 1.0 |
| *JPA Abstraktion erstellen Um im Java Teil mit der Datenbank zu interagieren habe ich die Datenbank mit JPA Klassen abgebildet.* | L.R. | 8.0 | 2.0 |
| **Total** |  | **8.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Als ich zu Beginn die Serviceklassen erstellt hatte, konnte ich nicht auf die Methoden in den EJB-Beans zugreifen, da die Variable «Null» war. Das erste Problem war, dass ich das eine Bean nicht mit der Annotation «Stateless» ausgestattet hatte. Ausserdem müssen Zugriffe auf die EJB-Bean in einer «PostConstruct» Methode erfolgen. Diese wird vor der Instanziierung des eigentlichen Web-Beans ausgeführt und sorgt dafür, dass die EJB-Beans ordnungsgemäss instanziiert werden.  Als ich die Vorgaben für die Business-Logik in den EJB-Beans durchgelesen habe, ist mir aufgefallen, dass dort genau spezifiziert ist, welche Datensätze genau ausgelesen werden sollen. Ein Grund dafür ist die Performance. Wenn man zum Beispiel ein Konto ausliest und zusätzlich noch eine «OneToMany»-Beziehung in der JPA-Konto Klasse definiert hat, werden zusätzlich zu den Kontodaten noch alle zugehörigen TransactionRecords ausgelesen. Dies wäre ineffizient, wenn lediglich die Kontodaten gefragt sind. Als Lösung habe ich bei den «OneToMany» und «ManyToOne» Beziehung das Attribut «fetch = FetchType.LAZY» hinzugefügt. Dies sorgt dafür, dass im vorherigen Beispiel die Liste mit den Transaktionen nur geladen werden, wenn diese benötigt werden.  Bei der Ausgabe der Transactions auf der View muss bei jeder Iteration entscheiden werden, ob der «Amount» in die «Belastungen» oder «Gutschriften» Tabelle kommen soll. Als erstes hatte ich die Idee, man könnte die dies mit dem «c:if»-JSF-Tag erreichen, indem jedesmal geschaut wird ob der «type» «D (Debit)» oder «C (Credit)» ist. Da das Vergleichen des Strings nicht funktioniert hat, habe ich stattdessen den «rendered»-JSF-Tag benutzt. Die Bedingung ist z.B. bei «Gutschriften»: Nur, wenn der «type» gleich «C» ist, soll der «outPutText» gerendert werden. | | | |
| **Reflexion** | | | |
| Gestern habe ich die JPA soweit fertig gemacht wie ich konnte. Das heisst ich habe alle Spalten der Datenbank eingetragen und die ManyToOne Verbindungen, sowie die Rückbeziehung OneToMany erstellt. Den Code habe ich dabei grösstenteils von alten JSF Projekten mit JAP Layern kopiert. Nebst der Definierung der Spaltennamen und weiteren Einstellungen habe ich bereits begonnen, die drei minimal erforderlichen NamedQuery’s für die Business-Logik zu erstellen. Nebst dem habe ich auch noch mit zusätzlichen Attributen dafür gesorgt, dass Daten nur geladen werden, wenn sie auch gebraucht sind (Lazy Load). | | | |
| **Nächste Schritte** | | | |
| * Fertigstellen der Aufgabenstellung «Kontoauszug anzeigen» * Business Logik für die zweite Aufgabe erstellen. * JSF-Beans für die zweite Aufgabe erstellen. | | | |

### 23. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Business-Logik für Kontoauszug Für den Kontoauszug habe ich die Logik für die Ausgabe des aktuellen Kontostandes erstellt.* | L.R. | 4.0 | 2.0 |
| *Business-Logik für Einzahlung/Auszahlung Für die Einzahlung/Auszahlung habe ich die Businesslogik in den EJB-Klassen erstellt.* | L.R. | 8.0 | 4.0 |
| *Web-Bean für Einzahlung/Auszahlung Für die Einzahlung/Auszahlung habe ich die Web-Bean erstellt und sowie einige Anpassungen an den Views vorgenommen.* | L.R. | 8.0 | 2.0 |
| **Total** |  | **18.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Bei den Transaktionen soll der User noch das gewünschte Konto auswählen können, für welches er die Transaktionen anzeigen will. Dafür kann der Nutzer dann auf einen Button «aktualisieren» klicken, um die View zu aktualisieren. Jedoch ist es bis dato nicht möglich die View zu aktualisieren, da die Daten nicht veränderbar sind. Das hat möglicherweise etwas mit dem Scope zu tun. Damit ich nicht zu viel Zeit mit diesem Problem verliere, habe ich mal mit dem anderen weitergemacht.  Die Daten waren zu Beginn nicht richtig sortiert, obwohl ich in der JPA Klasse mit der «@OrderBy»-Annotation definiert hatte, wie die Transaktionen sortiert werden sollen. Die Daten wurden richtig sortiert. Das Problem lag am jQuery-Datatables Plugin, welche eine «asc» Sortierung bei der ersten Spalte als Default hatte. Somit musste ich bei der Initialisierung der Library bei der Konfiguration definieren, dass die erste Spalte standardmässig mit «desc» definiert werden soll.  Bei JSF gibt es viele Tags, ebenfalls solche um «Radio-Buttons» zu generieren. Jedoch generiert JSF die «Radio-Buttons» in einem table-Layout, wobei Tabellen für die Positionierung verwendet werden. Dies ist ziemlich schlecht, wenn man versucht mit der «Plain»-Version von Bootstrap zu arbeiten, da das Layout nicht dazu passt. Somit musste ich die Buttons noch selbständig stylen, um diese in die gewünschte Position auszurichten. | | | |
| **Reflexion** | | | |
| Für die Ausgabe der Transaktionen habe ich heute in der Serviceklasse für die Accounts die Logik für die Transaktionsliste fertiggestellt. Für jeden Record soll auch noch der aktuelle Kontostand aktuelle Kontostand berechnet werden. Für diesen Zweck habe ich in der JPA Klasse noch ein weiteres Attribut «balance» hinzugefügt und mit der Annoation «@Transient» belegt, sodass JPA dieses Attribut bei der Interaktion mit der Datenbank ignoriert. In dieses Attribut wird dann für jeden Record der aktuelle Kontostand gespeichert. Dafür rechne ich einfach vom aktuellen Kontostand zurück bis zum Anfang der aufgezeichneten Records.  Für den zweiten Aufgaben-Teilbereich habe ich die Web-Beans erstellt, sowie die Methoden «debit()» und «credit()» in der Serviceklasse der Transaktionen. Diese beiden haben die Aufgabe je nach Auswahl des Transaktionstyps entweder eine Einzahlung oder Auszahlung vorzunehmen. In der «debit()»-Methode wird zusätzlich noch überprüft, dass der Nutzer sein Kontolimit nicht überschreitet welches im Attribut «OVERDRAFT» definiert ist. Wenn er unter dieses Limit fällt, wird eine Exception geworfen, welche im Web-Bean gecatched werden soll, damit dementsprechend Fehlermeldungen angezeigt werden. | | | |
| **Nächste Schritte** | | | |
| * Validation für die Inputfelder beim Einzahlung/Auszahlung. * Aktualisieren-Button zum Laufen bringen beim Kontoauszug. * Eventuell Datentyp der JPA Klassen zu BigDecimal ändern. | | | |

### 24. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Validation erstellen Für die Ein- und Auszahlung müssen die Inputfelder validiert werden.* | L.R. | 8.0 | 5.0 |
| *Logger erstellen Für die nichtfuntionalen Anforderungen soll ein Logger mit der Java Logging API erstellt werden.* | L.R. | 8.0 | 3.0 |
| *JUnit Testfälle erstellen Für das Unit-Testing wurden Testfälle mit JUnit erstellt und ausgeführt.* | L.R. | 0.0 | 1.0 |
| **Total** |  | **16.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Für die Validation benutze ich den «f:validate»-Tag, um die Inputfelder zu validieren. Von JSF gibt es bereits standardmässige Validatoren. Zu jedem Input Feld kann man noch einen «h:message»-Tag hinzufügen und definieren, für welches Feld die Message ausgegeben werden soll. Jedoch sieht die standardmässige Nachricht unschön aus, da die ID der Komponente noch ausgegeben wird. Also habe ich nach einer Möglichkeit gesucht, die Messages zu personalisieren. JSF hat für die Messages ein «preperties»-File im Jar-Archiv. Dieses File kann man mit dem «faces-context.xml»-File überschreiben, indem man sein eigenes File definiert.  Beim Custom Validator hatte ich ausserdem Probleme mit dem Value. Wenn ich versuchte den Value in einen double-Datentyp zu verwandelt gab es immer eine NullPointerException, da der Value «Null» war. Der Grund ist, dass wenn der Nutzer nichts eingibt, die Value logischerweise «Null» ist. Dies kann man entweder lösen, indem man im Validator noch eine Überprüfung auf «Null» einfügt oder anstatt den «f:validateRequired»-Tag verwendet, den «required=true»-Tag im Input-Feld verwendet. Der Unterschied ist, dass dann der custom Validator nicht ausgeführt wird, wenn noch nichts eingefüllt wurde.  Der «Aktualisieren-Button» bei den Transactions funktioniert jetzt zwar, jedoch werden die Daten nicht neu geladen. Für dieses Problem muss ich noch meinen Fachvorgesetzten fragen, ob es nicht doch eine Lösung gibt. | | | |
| **Reflexion** | | | |
| Beide Versionen Aufgaben funktionieren momentan (mit Ausnahme der neuen Daten bei der Aktualisierung). Was jedoch noch fehlt ist die Validation für die Ein- und Auszahlung. Da der User hier Daten in Inputfeldern eingeben kann ist dies notwendig. Da in JSF bereits bestehende Validationskomponenten enthaltet, habe ich mir vorgenommen, möglichst diese für die Validation zu verwenden.  Für die nicht funktionale Aufgabe soll noch ein Logging mit der Java Logging API realisiert werden. Dafür habe ich eine Util-Klasse mit statischen Methoden erstellt, welche immer aufgerufen wird, wenn ich etwas Loggen möchte. Eventuell kann ich die Logging Klasse noch nach dem Singleton-Pattern bauen, sodass der FileHandler nur einmal instanziert wird.  Währenddessen habe ich bereits mit dem Unit-Testing der Servicestellen mit JUnit angefangen. Für die Views muss ich noch definieren, was ich genau testen will. Dazu sollte Testfälle erstellt werden und ein Testprotokoll, wenn ich die Testfälle dann ausführen werde. | | | |
| **Nächste Schritte** | | | |
| * Alle Testfälle erstellen * Logger verbessern * Testfälle durchführen | | | |

### 25. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Dokumentation nachführen Die Dokumentation muss noch nachgeführt werden.* | L.R. | 8.0 | 5.0 |
| *Logger verbessern Da mein Logger momentan jedes Mal den FileHandler instanziert kann ich diesen zu einem Singleton Pattern umwandeln.* | L.R. | 8.0 | 3.0 |
| *JUnit Testfälle erstellen Für die Service-Klassen muss ich noch die JUnit Tests erstellen.* | L.R. | 0.0 | 1.0 |
| *Testfälle erstellen Für die Tests welche ich manuell auf dem GUI durchführe, muss ich noch Testfälle erstellen.* | L.R. | 0.0 | 1.0 |
| *Expertengespräch Heute steht das zweite Expertengespräch an.* | L.R. | 0.0 | 1.0 |
| **Total** |  | **16.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Abgesehen von dem Problem mit der Aktualisierung des Kontostands je nach Auswahl des Kontos gibt es bis jetzt keine Probleme. Für das Problem habe ich die Action-Methode zu «void» geändert. Somit wird nun nicht mehr der «@PostConstruct» ausgeführt, sondern nur die Action-Methode. Vorher wurde bei jeder Aktualisierung auch der PostConstruct ausgeführt.  Momentan habe ich das Projekt meinem Fachvorgesetzten abgegeben. Dieser schaut es sich an und wird dann mögliche Lösungsansätze mit mir besprechen. | | | |
| **Reflexion** | | | |
| Für den Logger habe ich den Pfad bisher statisch in meine Logging Klasse geschrieben. Diese soll nun den User-Pfad dynamisch holen und die Logging Dateien in den erstellten Ordner schrieben. Ausserdem muss ich das Singleton Pattern ausführen, sodass der FileHandler nur einmal ausgeführt wird.  Heute hatte ich das zweite Expertengespräch. Hierbei wollte der Experte den Stand meines Arbeitsprotokolls sehen, sowie den Stand meines Zeitplans. Wie beim ersten Besuch befragt er mich, ob es bis anhin Probleme oder Schwierigkeiten gibt. Ausserdem gab er mir einige administrative Informationen über die Abgabetermine, den Umfang und Erwartungen des IPA-Berichtes und machte mich auf das Dokument FArbeit.pdf aufmerksam, in welchem alle relevanten Informationen drin stehen. | | | |
| **Nächste Schritte** | | | |
| * Weitere Testfälle erstellen * Alle JUnit Tests ausführen * Aktualisierung zum Laufen bringen * Dokumenation weiterführen * Methodik der Dokumentation erstellen * Testfälle durchführen und Testprotokoll erstellen | | | |

### 30. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Dokumentation schreiben Der Hauptteil der Dokumentation sollte noch nachgeführt werden.* | L.R. | 0.0 | 5.0 |
| *Fehler beim Kontoauszug Der Fehler, dass es nicht immer die aktuellen Daten beim Kontoauszug holt, sollte noch gefixt werden.* | L.R. | 0.0 | 2.0 |
| *Testfälle durchführen Die erstellten Testfälle sollen durchgeführt und protokolliert werden.* | L.R. | 0.0 | 1.0 |
| **Total** |  | **8.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Am Anfang des Tages bestand immer noch der Fehler, dass die Daten beim Kontoauszug nicht aktualisiert wurden. Der Grund dafür war, dass ich die Daten via «LazyLoad» hole, sodass nur diejenigen Daten geholt werden, welche auch wirklich benötigt werden. Da zuerst ein Account-Objekt hole und dann via der getter-Methode die Transactions für diesen Account hole, wurden die Daten immer gecached. Eine Möglichkeit zur Behebung dieses Problems wäre die Abschaltung des Cache für die betroffenen Entities. Dies wäre jedoch eher destruktiv, da man dann nicht von der Caching Funktionalität profitieren könnte. Eine zweite Möglichkeit wäre, dass man beim Hinzufügen eines neuen TransactionRecords einfach manuell ein neues Objekt zur Liste im Account-Objekt hinzufügt oder die «refresh»-Methode für das Account-Objekt aufruft, sodass die Daten aktualisiert werden.  Schliesslich kann man auch einfach ein Named Query im TransactionRecord-Entity erstellen, sodass die Transaction direkt selektiert werden. Dies würde jedoch eine Verletzung der Aufgabenstellung zur Folge haben. Laut dem Fachvorgesetzen sollte dies aber kein Problem sein. | | | |
| **Reflexion** | | | |
| Am Morgen habe ich erst einmal angefangen, die Dokumentation zu erweitern. Da ich im Hauptteil noch praktisch keine Inhalte habe, muss ich beim Punkt Implementierung noch weitere Erläuterungen zu den Punkten «Business-Schicht», «Persistenz-Schicht», «Validation» und «Logger» schreiben.  Zusätzlich gab es immer noch den Fehler, dass die Daten beim Kontoauszug nach einer Einzahlung nicht aktualisiert wurden. Nachdem ich dies mit meinem Fachvorgesetzten besprochen hatte und er mir einige Lösungen dafür vorschlug, konnte ich den Fehler lösen. Danach habe ich weiter an der Dokumentation geschrieben.  Schliesslich habe ich noch einen weiteren Testfall erstellt, der sicherstellt, dass der Betrag nur nummerische Werte enthalten darf. Gegen Ende des Tages startete ich mit dem Testprotokoll und der Durchführung der Testfälle. Dazu musste ich die BbcBank Datenbank zurücksetzten, damit die die richtigen Daten zur Ausführung verwendet werden. | | | |
| **Nächste Schritte** | | | |
| * Testfälle durchführen * IPA abschliessen * Code aufräumen | | | |

### 31. Januar 2017

|  |  |  |  |
| --- | --- | --- | --- |
| Tätigkeiten | Person | Soll (Std) | Ist (Std) |
| *Testfälle durchführen Die erstellten Testfälle sollen durchgeführt und protokolliert werden.* | L.R. | 0.0 | 2.0 |
| *JUnit Tests aktualisieren Da die Business-Services geändert wurden, müssen die JUnit Tests ebenfalls angepasst werden.* | L.R. | 0.0 | 1.0 |
| *Dokumentation schreiben Der Hauptteil der Dokumentation sollte noch nachgeführt werden.* | L.R. | 0.0 | 6.0 |
| **Total** |  | **0.0** | **8.0** |
| **Probleme / Hilfestellungen** | | | |
| Ein Test wurde beim Testen als «not OK» markiert, da die falsche Fehlermeldung ausgegeben wurde. Der Fehler lag bei der Überprüfung des Wertes in der «validate»-Funktion im PaymentBean. Dort wird nicht zwischen Credit und Debit unterschieden. Wenn nun aktuell gerade 950'000 im Konto liegen und man versucht eine Auszahlung von mehr als 50'000 zu machen wird das nicht gehen, da der Betrag dann mehr als 999'999 betragen würde. Somit muss bei der if-Bedingung zusätzlich noch geprüft werden, ob es sich um eine Einzahlung handelt. | | | |
| **Reflexion** | | | |
| Am Morgen habe ich alle Testfälle durchgetestet. Ausser einem Test wurden alle mit OK markiert. | | | |

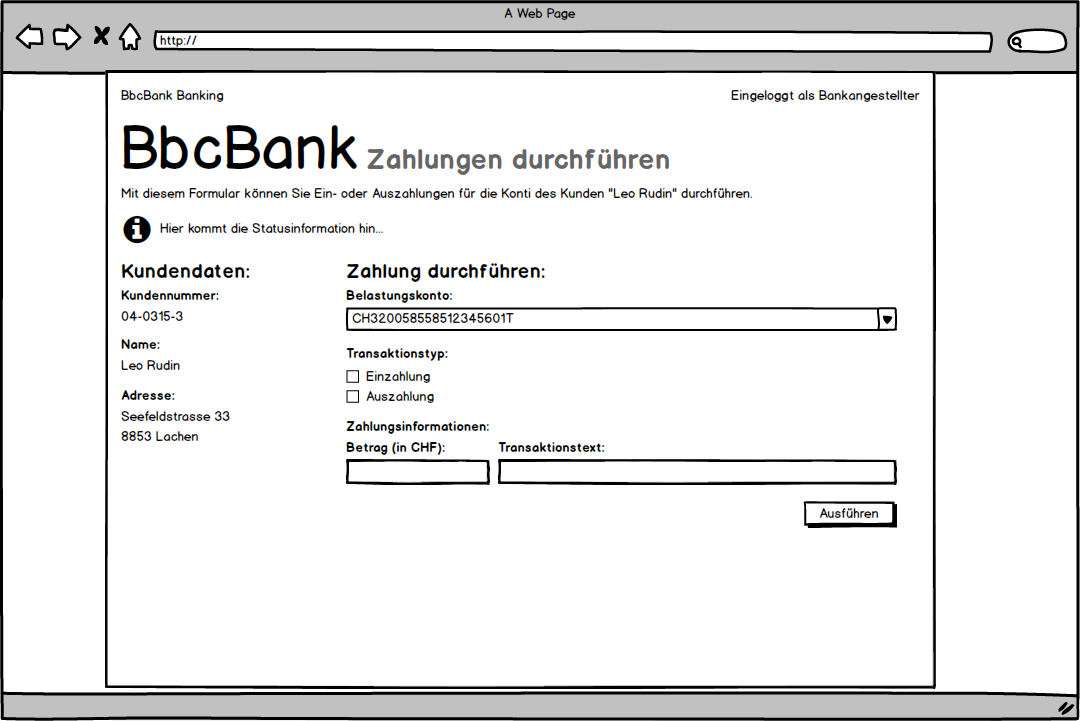
# Projekt

## Konzept

### Moqups

Für die GUI’s der BankApp habe ich vorgängig zwei Moqups für die beiden User Interfaces erstellt.

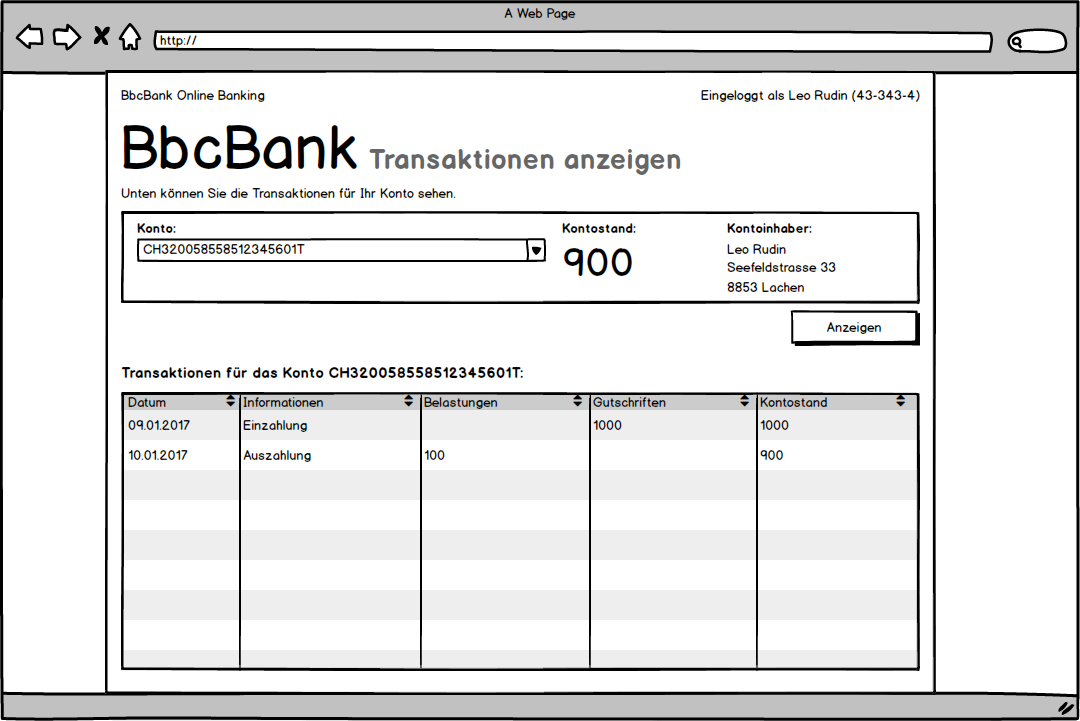
#### Ein- und Auszahlung



Nachdem der Bankangestellte nach dem Kunden gesucht hat und erfolgreich einen Kunden finden konnte, wird er auf diese View weitergeleitet. Diese enthält einige Stammdaten des Kunden, beispielsweise den Namen und die Adresse. Rechts daneben gibt es das Formular zur Durchführung der Zahlung. Da ein Kunde unter Umständen mehre Konten haben kann, gibt es für den Bankangestellten die Möglichkeit im Drop Down ein Bankkonto auszuwählen. Danach kann man wählen, ob eine Einzahlung oder Auszahlung vorgenommen werden möchte und den Betrag in CHF und den Transaktionstext definieren. Mit dem Button «Ausführen» kann die Zahlung vorgenommen werden.

Das runde Ausrufezeichen dient als Platzhalter für den Statustext.

#### Konto Auszug anzeigen



Für die Anzeige des Kontoauszuges muss sich der Nutzer zuerst einloggen und erhält dann eine Liste von Transaktionen des vorselektierten Kontos. Zusätzlich sieht man auch den aktuellen Kontostand des ausgewählten Kontos sowie den Kontoinhaber. Wenn man das Konto ändern möchte, muss man im Drop Down ein anderes auswählen und drückt dann auf den Button «Anzeigen».

### Use Case Diagramm

D:\IPA\Use Case Diagramm\use_case_diagramm.emf

Im Use-Case Diagramm gibt es zwei Akteure: Bankangestellter und Kunde. Nur der Bankangestellte kann Einzahlungen und Auszahlungen für Bankkonten durchführen. Diese Operationen werden vom Schalter aus getätigt. Wenn z.B. eine Kunde an den Schalter kommt und eine Auszahlung vornehmen möchte, kann der Bankangestellte auf das GUI zugreifen.

Das dritte und letzte Element ist der «Kontoauszug anzeigen». Dabei kann der Kunde die Bewegungen seiner Konten nach erfolgreicher Authentifizierung sehen. Ich persönlich denke, dass diese Operation auch vom Bankangestellten durchgeführt werden kann, da dieser auch ein Interesse an den Kontenbewegungen der Kunden hat.

### Activity Diagramm

#### Ein- und Auszahlung

D:\IPA\Aktivitätsdiagramm\Einzahlung_Auszahlung.emf

Bei der Ein- und Auszahlung kann der Bankangestellte zuerst einen Nutzer suchen. Sofern ein Nutzer gefunden werden konnte, wird man zur Ein- und Auszahlungsseite weitergeleitet. Dort kann man dann eine Zahlung vornehmen.

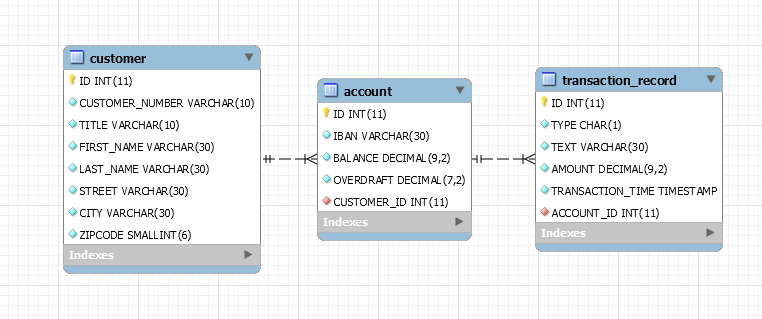
Bei der Validierung der Zahlung wird entschieden, ob diese durchgeführt wird oder nicht. Je nachdem wird dann im Status entweder eine Fehlermeldung oder der neue Saldo des Kontos angezeigt.

#### Kontoauszug anzeigen

D:\IPA\Aktivitätsdiagramm\Kontoauszug_Anzeigen.emf

Beim Kontoauszug anzeigen muss sich der Kunde zuerst authentifizieren. Sofern die Authentifizierung erfolgreich war, wird der Kunde zur Anzeige des Kontoauszuges weitergeleitet. Dort kann er dann eine seiner Konten auswählen, und die Transaktion für das ausgewählte Konto werden angezeigt. Somit führt eine Änderung der Filtereinstellungen logischerweise wieder zum Kontoauszug.

### ERM Diagramm



In der Bank-Applikation gibt es insgesamt drei Tabellen. Die erste Tabelle ist die Tabelle «customer». Diese enthält die Kundendaten jedes Kunden. Neben den generellen Kundendaten gibt es noch eine Kundennummer, welche dem Kunden eindeutig zugeordnet werden kann.

In der «account» Tabelle werden alle Konten geführt. Jedes Konto erhält einen Fremdschlüssel zur Kunden-Tabelle, da jedes Konto einem Kunden gehört.

Schlussendlich gibt es noch die «transaction\_record» Tabelle. Diese enthält die Bewegungen der Konten. Jede Bewegung benötigt daher auch einen Fremdschlüssel auf die Konten-Tabelle, da jede Bewegung zu einem Konto gehört.

## Implementierung

### Präsentationsschicht

In der Präsentationsschicht liegen alle Komponenten, welche der Nutzer schlussendlich sieht. Das können Formulare für die Eingabe von Daten sein, oder Ausgabeseiten, welche Daten in einer ansprechenden Form anzeigen. Um ein optimales Nutzererlebnis zu garantieren, sollte das GUI möglichst leicht verständlich und selbsterklärend sein.

Diese Elemente werden in den sogenannten «Views» zusammengefasst. Das sind Dateien, welche die Endung «xhtml» haben. Sie liegen dabei im Web-Teil der Applikation, also in gleichen Teil wie der JSF-Code. JSF ist auch dafür verantwortlich, mit den Views zu kommunizieren und bestimmt dabei die Lebenszeit der Daten. View und Programmlogik liegen also im Web-Teil nah beieinander.

Die Views wurden mit folgenden Komponenten realisiert:

* Bootstrap
* Roboto Font von Google
* Datatables jQuery Plugin
* Glyphicons von Bootsrap

Für Bootstrap gäbe es auch die speziell für JSF entwickelte Library «Bootsfaces», welche die Bootstrap HTML-Struktur durch die Definierung mittels JSF-Tags generiert. Da ich jedoch privat bereits viel mit Bootstrap gearbeitet habe, kenne ich mit dem generierten HTML-Code besser aus. Aus diesem Grund habe ich die Standard-Version von Bootstrap gewählt, da ich so schneller und effizienter arbeiten kann.

Als Ersatz der Standard Windows-Schriftart «Arial» habe ich die Schriftart «Roboto» von Google gewählt: <https://fonts.google.com/specimen/Roboto>.

Das «DataTable»-jQuery Plugin dient zur Generierung von Tabellen. Verglichen mit normalen HTML-Tabellen besitzen diese Tabellen zusätzliche Funktionen: Einzelne Spalten können mit JavaScript sortiert werden, es gibt eine «Pagination» und mithilfe einer Textsuche kann nach einzelnen Tabellen gesucht werden.

#### Default-Template

Damit ich repetitive Dinge wie zum Beispiel die Angabe des Stylesheets oder der JavaScript-Files nicht auf jeder Seite neu schreiben muss, bietet JSF die Erstellung eines Default-Templates an. In diesem werden die vorher genannten Dinge eingebunden. Für den Content wird dann im Template-File ein Bereich definiert, wo der Content hinkommen wird:

<ui:insert name="content">

</ui:insert>

Wenn man das Template nun verwenden möchte, muss man folgendes tun:

<ui:composition template="resources/templates/default.xhtml">

<ui:define name="content">

</ui:define>

</ui:composition>

Man bindet eine «ui:composition» ein und definiert, wo der das Template liegt. In diesem Tag können dann die mittels «ui:define» die im Template definierten Stellen überschrieben werden. So kann man sicherstellen, dass auf jeder Seite die gleichen eingebundenen Elemente verwendet werden. Wenn sich eine Ressource ändert, muss man diese in nur einem File ändern.

Neben dem Content habe ich im Template-File noch einen Bereich für den title und dem javascript Teil definiert. Insbesondere den «title»-Tag möchte man eventuell immer individuell anpassen.[[1]](#footnote-1)

#### Glyphicons von Bootstrap

Das Datatables jQuery-Plugin verwendet – sofern Bootstrap eingebunden ist – die Glyphicons von Bootstrap. Diese wurden aber nicht angezeigt. Als Lösung für dieses Problem muss der «@font-faces»-Tag von Bootstrap im default-Template mit den richtigen Pfaden überschrieben werden.

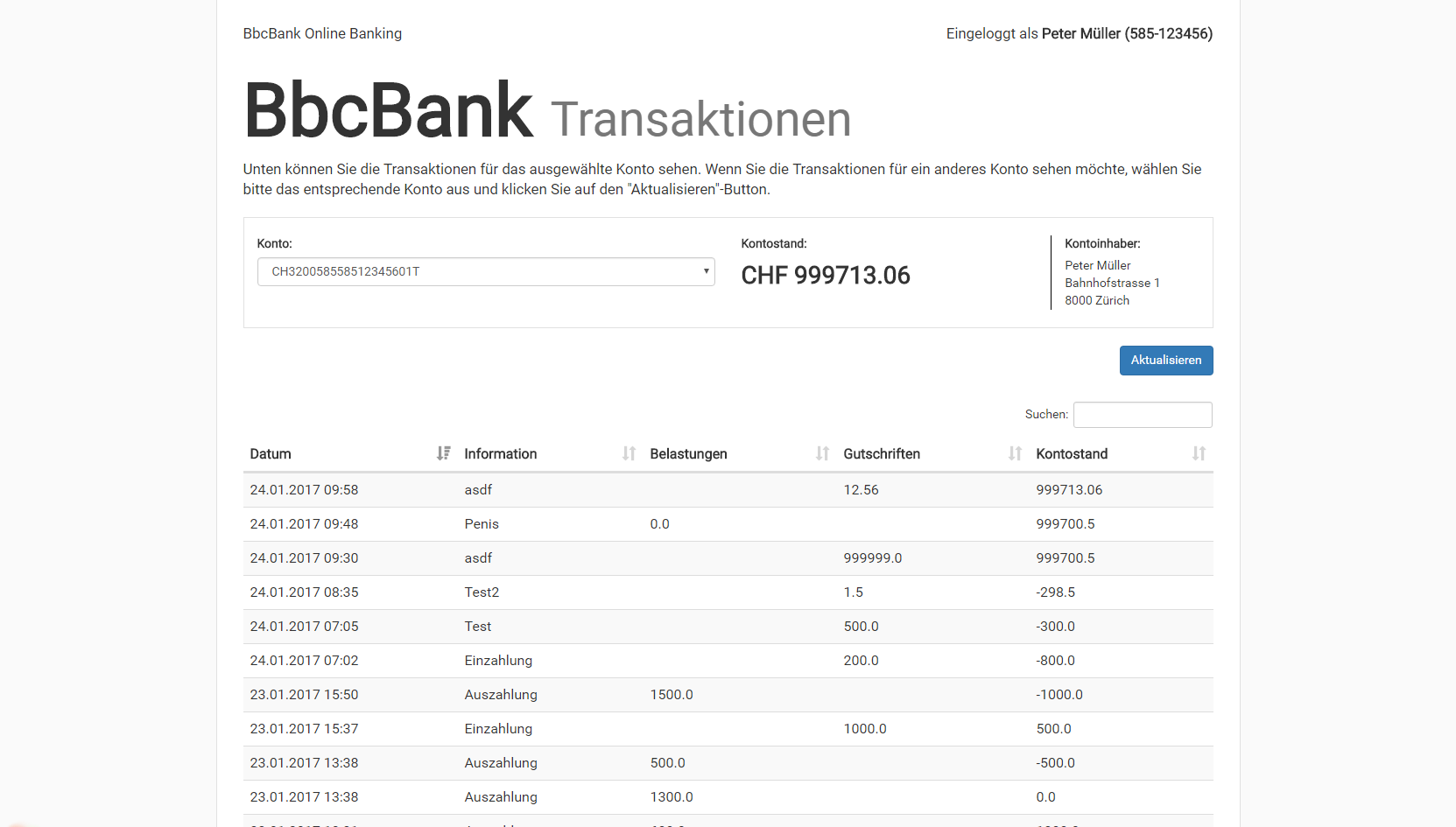
Da CSS, JavaScript etc. bei Bootstrap im «resources» Ordner liegen, müssen diese immer mit den Tags eingebunden werden, sodass die Pfade generiert werden. Für diesen Zweck gibt es folgenden Tag, um den Pfad zu Ressourcen zu generieren:[[2]](#footnote-2)

#{resource['fonts:glyphicons-halflings-regular.eot']}

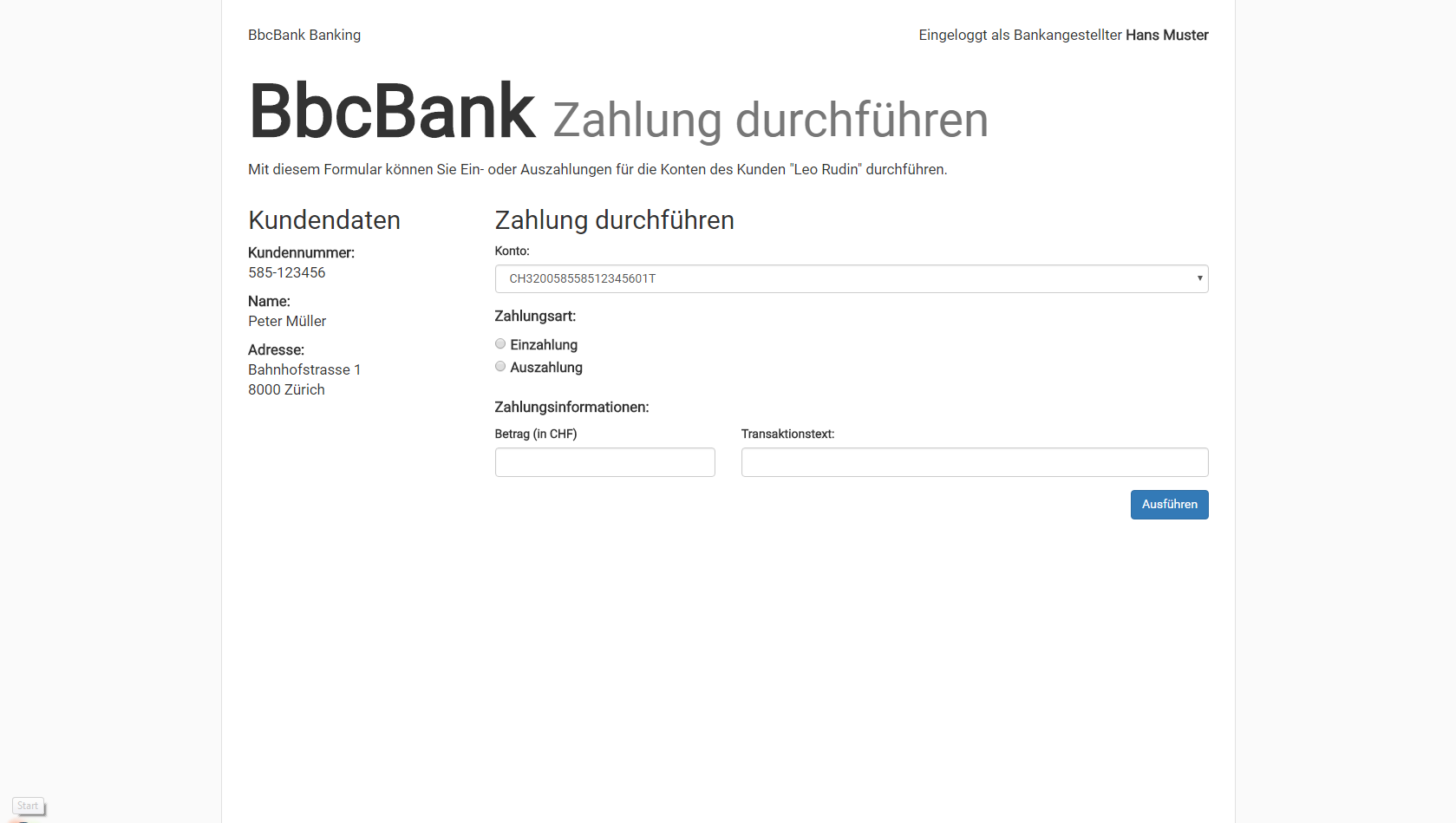
Somit muss anstelle des statischen Pfades dieser Tag zur Generierung verwendet werden.

#### Screenshots

##### Kontoauszug anzeigen:



##### Ein- und Auszahlung:



### Business-Schicht

In der Business-Schicht liegen alle EJB-Klassen mit den Service-Methoden und Funktionen. Diese werden vom JSF-Teil aus aufgerufen. Instanziiert werden die Service-Klassen durch eine Annotation «EJB». Die Funktionen können in einer «PostConstruct»-Methode aufgerufen werden:

@EJB

private CustomerService customerService;

Angesprochen wird hier das erstellte Interface für die EJB-Klasse. Wenn eine neue Klasse erstellt werden möchte, muss ein Interface mit den gewünschten Methoden erstellt werden, welches dann von der eigentlichen EJB-Bean (im Fall oben: «CustomerServiceBean.java») implementiert wird.

Ein primäres Ziel mit der Erstellung von EJB Klassen ist die Vermeidung von Redundanzen. Die erstellten Klassen können beispielsweise auch von anderen Orten aufgerufen werden. Die Business-Schicht kann sozusagen als API für weitere Apps verwendet werden und dabei wird sichergestellt, dass der Code nicht noch einmal geschrieben wird.

Für die EJBs in der BbcBank Applikation wurde noch per Annotation ein State definiert:

@Stateless(name = "CustomerService")

public class CustomerServiceBean implements CustomerService {

Eine «Statefull» deklarierte Session-Bean funktioniert dabei etwa gleich wie ein «SessionScope» deklariertes JSF-Bean. Die Daten werden während mehreren Aufrufen beibehalten während ein «Stateless» deklariertes Bean jedes Mal neu instanziiert wird. Diese sind dabei nicht an einen Aufrufer gebunden.[[3]](#footnote-3)

Für die BbcBank-Applikation gibt es drei Services:

* TransactionService
* AccountService
* CustomerService

### Persistenz-Schicht

Die oberste Schicht ist die Persistenz-Schicht. Diese beinhaltet den JPA Teil sowie die Datenbank.

Normalerweise kann man aus dem Java Code nicht direkt auf die Datenbank zugreifen. Es müssen immer zuerst manuell SQL-Querys abgesetzt werden. Mithilfe von JPA kann man über Java-Objekte (JPA-Entities) mit der Datenbank interagieren, was eine Erleichterung für den Entwickler ist.[[4]](#footnote-4)

Die Tabellen werden dabei in Java Objekte abgebildet. Diese Objekte müssen als solches markiert werden:

@Entity

@Table(name = "account")

public class Account extends AbstractBaseEntity {

Mit der Annotation «Entity» wird die Klasse als JPA-Entity markiert. Mit der Annotation «Table» kann der Name der Tabelle angegeben werden, welche in dieses Objekt gemappt wird.

Im Beispiel oben wird ein «AbstractBaseEntity» vererbt. Da normalerweise jede Tabelle einen Primärschlüssel enthält, kann man gleiche Attribute wie sonst auch üblich in eine Oberklasse deklarieren. Diese Oberklasse sieht dann folgendermassen aus:

@MappedSuperclass

public abstract class AbstractBaseEntity {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ID")

private Long id;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

}

Mit der Annotation «MappedSuperclass» wird die Klasse als ebenso eine deklariert, welche von anderen Entitys vererbt wird.[[5]](#footnote-5) Damit sie nicht instanziiert werden kann wird sie «abstract» deklariert.

Wie vorhin schon erwähnt gibt es hier nur das Attribut Id, da dieses Attribut in jeder Tabelle enthalten ist. Mit der Annotation «Id» wird das Attribut als ein Primärschlüssel definiert. Zusätzlich wird definiert, dass das Attribut in der Datenbank als AutoIncrement deklariert wurde, also soll auch bei JPA die Id immer hochgezählt werden.

Schliesslich gibt es noch die «Column»-Annotation. Diese definiert, dass dieses Attribut einer Tabellen-Spalte entspricht. Zusätzlich kann noch der Name der Spalte auf der Datenbank angegeben werden.

Weitere Spalten werden etwa gleich deklariert:

@Column(name = "IBAN", nullable = false, unique = true)

private String iban;

Was zusätzlich noch angegeben werden kann ist «nullable», welches definiert, ob das Attribut in der Datenbank als «Not Null» deklariert wurde oder nicht sowie «unique» welches definiert, ob das Attribut in der Datenbank als «Unique» deklariert wurde oder nicht.

#### ManyToOne und OneToMany

Wenn man Beziehungen zwischen Tabellen definieren möchte, kann dafür die «ManyToOne» oder «OneToMany»-Annotationen verwendet werden.

Im BbcBank-Projekt gibt es zum Beispiel eine Tabelle «accounts», welche einen Fremdschlüssel auf einen Record der Tabelle «customers» besitzt. In diesem Fall verwendet man die «ManyToOne»-Annotation:

@ManyToOne(optional = false, cascade = CascadeType.REFRESH, fetch = FetchType.LAZY)

@JoinColumn(name = "CUSTOMER\_ID")

private Customer customer;

Mit der Angabe von «optional» wird gesagt, das sicher immer ein Record für den gegebenen Fremdschlüssel auf der Datenbank existiert, der customer also sicher nicht null ist.[[6]](#footnote-6) Mit cascade kann definiert werden, ob bei einem Refresh des Account Objektes auch gleich ein Refresh auf das assoziierte Kunden Objekt durchgeführt werden soll. [[7]](#footnote-7)

Da wir laut Aufgabenspezifikation die benötigten Ressourcen von der Datenbank laden sollen, wurde bei der Relation noch der fetch auf «FetchType.LAZY» gesetzt. Dies sorgt dafür, dass das auf das assoziierte Customer Objekt nur geholt wird, wenn auch wirklich darauf zugegriffen wird. Dies sorgt z.B. dafür, dass bei einem Select auf einem einzelnen Account nicht gleich auch alle TransactionRecords geholt werden, sofern diese Rückbeziehung definiert ist.[[8]](#footnote-8)

Schliesslich wird mit «JoinColumn» noch definiert, was der Name des Fremdschlüssel in der «accounts»-Tabelle ist, mit welchem dann der Join durchgeführt.

Beziehungen können auch in die andere Richtung definiert werden. Dies wird mit der «OneToMany»-Annotation erreicht:

@OneToMany(mappedBy = "account", cascade = CascadeType.ALL, orphanRemoval = true, fetch = FetchType.LAZY)

private List<TransactionRecord> transactionRecords = new ArrayList<>();

Die «mappedBy» Annotation sagt dabei aus, das der Fremdschlüssel in der anderen Tabelle liegt (hier «transactionRecords»-Tabelle) und dieser auf den Primärschlüssel im Account Entity zeigt.[[9]](#footnote-9) Mit «CascadeType.ALL» werden alle Operationen definiert (persist, remove, refresh etc.) welche auch im anderen Objekt durchgeführt werden sollen, sofern einer diese Operationen auf das Account-Objekt angewendet wird. [[10]](#footnote-10) Wenn «orphanRemoval» auf true gesetzt ist, werden bei einer Löschung des Account Objektes auch alle zugehörigen TransactionRecords gelöscht, da diese dann kein zugewiesenes Objekt mehr haben.

#### Named Querys

Mithilfe von Named Querys werden die eigentlichen Abfragen in den Serviceklassen durchgeführt. Diese werden über dem Entity, zu dem sie gehören, erstellt:

@NamedQueries({

@NamedQuery(name = "TransactionRecord.getTransactionsByIban", query = "select t from TransactionRecord t where t.account.iban = :iban order by t.transactionTime desc")

})

Mit der «NamedQueries»-Annotation wird definiert, dass gleich mehrere NamedQuerys definiert werden. Mit «NamedQuery» wird ein einzelnes Query definiert. Diese müssen immer einen eindeutigen Namen haben, sodass sie aus der Service-Klasse eindeutig identifizierbar sind. Danach können diese mit dem entityManager im EJB-Bean aufgerufen werden:

List<TransactionRecord> transactions = entityManager.createNamedQuery("TransactionRecord.getTransactionsByIban", TransactionRecord.class).setParameter("iban", iban).getResultList();

In diesem Beispiel wird eine Liste aus TransactionRecords erstellt.

#### Transient

Mithilfe der «Transient»-Annotation werden deklarierte Attribute von JPA ignoriert:

@Transient

private Double balance;

In der BbcBank-Applikation wird zum Beispiel der Balance jedes TransactionRecords ignoriert, da ich erst nach dem selektieren der Attribute hier dynamisch immer der aktuelle Balance berechnet wird.

### Validation

Die Validation übernehmen zum einen die Standard-Validatoren von JSF, zum anderen wurden eigene Validation-Klassen erstellt. Diese werden bereits vor der Ausführung der Action-Methode ausgeführt und sorgen daher direkt für die Validation.

Um ein Validator für ein Input-Feld zu erstellen, kann man die gewünschten Regeln mit dem «f:validator»-Tag definieren:

<h:inputText id="transaction-info" value="#{paymentBean.transactionRecord.text}" requiredMessage="Dieses Feld ist erforderlich.">

<f:validateRequired />

<f:validateLength maximum="30" />

</h:inputText>

Zusätzlich kann mithilfe von z.B. «requiredMessage» die Nachricht im Falle eines Fehlers angepasst werden. Dies ist jedoch nicht empfehlenswert, da dies sehr viele redundante Codestellen zur Folge hat.

Als Alternative kann auch das Standard «Message.properties»-File überschrieben werden. Dazu sollte man sich zuerst einmal das Standard-File ansehen, um herauszufinden, welcher properties-Key überschrieben werden soll. Wenn der gewünschte Key gefunden wurde, kann man den Key und den Value in ein eigenes .properties-File kopieren. Das neue .properties-File muss im Ordner «bbcbank-web\src\main\resources\ch\bbcbank» gespeichert werden.

Nun muss das File noch im faces-config.xml eingebunden werden. Sofern dieses .xml noch nicht besteht, kann man es unter «bbcbank-web\src\main\webapp\WEB-INF» erstellt werden. Darin kopiert man folgenden Inhalt:

<?xml version="1.0" encoding="UTF-8"?>

<faces-config

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-facesconfig\_2\_0.xsd"

version="2.0">

<application>

<message-bundle>

ch.bbc.messages

</message-bundle>

</application>

</faces-config>

Wenn nun ein Default-Fehlermeldung provoziert wird, sollte die eigene Meldung angezeigt werden.

Wenn die Standard-Validation nicht ausrecht können auch eigene erstellt werden. Definiert werden diese mit dem «f:validator»-Tag mit zusätzlicher Angabe der «validatorId»

<h:inputText id="amount" styleClass="form-control" value="#{paymentBean.transactionRecord.amount}" required="true" requiredMessage="Dieses Feld ist erforderlich.">

<f:validator validatorId="ch.bbc.fit4ipa.bbcbank.validators.AmountValidator" />

</h:inputText>

Die ValidatorId wird bei der Validator-Klasse mit der «FacesValidator»-Annotation definiert. Eine vollständige eigene Validationsklasse sieht folgendermassen aus:

@FacesValidator("ch.bbc.fit4ipa.bbcbank.validators.AmountValidator")

public class AmountValidator extends BaseValidator implements Validator {

@Override

public void validate(FacesContext context, UIComponent component, Object value) throws CustomValidatorException {

}

}

Wie bereits vorher erwähnt besteht eine Validator-Klasse aus der «FacesValidator»-Annotation. Implementiert wird die «javax.faces.validator.Validator»-Klasse, und die mit dieser Klasse definierte «validate»-Methode, welche vier Parameter besitzt. Der wichtigste ist dabei der «value»-Parameter. Hier wird der eigentliche Wert übergeben, welcher validiert werden kann. Wenn ein Fehler auftritt, sollte eine Exception vom Typ «javax.faces.validator.ValidatorException» geworfen werden. Diese erwartet ein JSF-FacesMessage.

Im Beispiel oben habe ich die standart-Fehlermeldung überschrieben, da ich in meiner Exception noch den Fehler logge.[[11]](#footnote-11)

### Logging

Bei der Aufgabenstellung ist eine zusätzliche nicht-funktionale Anforderung gegeben, dass ein Logger mit der Java Logging API erstellt werden muss.

Dafür habe ich eine LogUtil Klasse nach dem Singleton Pattern erstellt:

public class LogUtil {

private static LogUtil instance;

private Logger logger;

private LogUtil() {

logger = Logger.getLogger("logger");

FileHandler fh;

String path = System.getProperty("user.home");

System.out.println(path);

File directory = new File(path + File.separator + "bbcbankLog");

if(!directory.exists()) {

directory.mkdirs();

}

try {

// Configure File Handler

fh = new FileHandler(path + File.separator + "bbcbankLog" + File.separator + "bbcbankLog.log");

logger.addHandler(fh);

SimpleFormatter formatter = new SimpleFormatter();

fh.setFormatter(formatter);

} catch (SecurityException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

public static LogUtil getInstance() {

if (instance == null) {

instance = new LogUtil();

}

return instance;

}

public void log(String message, Level level) {

logger.log(level, message);

}

}

Wann immer man den Logger benötigt, kann man die Instanz der Klasse mithilfe der «getInstance»-Methode holen. Sofern die Klasse noch nicht instanziiert wurde, wird diese instanziiert und dabei wird der FileHandler erstellt. Die Log-Files werden dabei im Nutzerverzeichnis im Ordner «bbcbankLog» abgelegt.

Nachdem die Instanz der Klasse verfügbar ist, kann man mit der «log»-Methode eine neuen Record hinzufügen.

Momentan werden die Nachrichten in den Validations-Exceptions geloggt und es wird ein neuer Log-Record hinzugefügt, wenn ein neuer TransactionRecord hinzugefügt wird.

## Testing

Beim Testing sollen sowohl Unit-Tests, sowie Tests über das GUI durchgeführt werden. Die Unit-Tests wurden mit dem Testing-Framework jUnit geschrieben. Hierbei werden alle Serviceklassen getestet. Dies setzt natürlich voraus, dass bereits Daten in der Datenbank bestehen.

Voraussetzung, um die Tests durchzuführen, sind die beim Generieren der Datenbank erstellten Daten (vom SQL-Dump). Bei den jUnit Tests wird auch geprüft, wieviel Konten beispielsweise für eine mitgegebene Kundennummer zurückkommen.

### JUnit Testing

Mit JUnit werden die Methoden in den Service-Klassen getestet. Die dazugehörigen Tests wurden im Ordner «src/test/java/ch/bbc/fit4ipa/bbcbank» im EJB-Projekt abgelegt. Jede neue Testklasse muss die Klasse «AbstractBaseTest» erben. In dieser Klasse werden die Pfade zu den Service-Klassen deklariert, sodass diese dann bei Bedarf instanziiert werden kann. Ausserdem wird hier die SQL-Verbindung erstellt.

Wenn eine neue ServiceKlasse getestet werden soll, muss in der Parent-Klasse ein neuer Pfad zum EJB angegeben werden. Zusätzlich sollte auch eine Methode erstellt werden, welche die Instanz der neuen ServiceKlasse übergibt, sodass mit dieser interagiert werden kann.

### Testfälle

Diese Testfälle setzten voraus, dass die Daten des Standard SQL-Dumps eingefügt wurden. Die Tests sollten in der korrekten Reihenfolge durchgeführt werden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr** | **Beschreibung** | **View** | **Input** | **Erwartetes Resultat** |
| 1 | Erfolgreich eine Auszahlung durchführen | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Auszahlung  Betrag: 1000  Text: Auszahlung | Die Auszahlung war erfolgreich mit der Meldung, dass der neue Kontostand CHF -100 beträgt. |
| 2 | Erfolgreich eine Einzahlung durchführen | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 1100  Text: Einzahlung | Die Einzahlung war erfolgreich mit der Meldung, dass der neue Kontostand CHF 1000.0 beträgt. |
| 3 | Ein- und Auszahlung werden auf dem beim Kontoauszug angezeigt. | transactions.xhtml | Konto: CH320058558512345601T | Die Ein- und Auszahlung werden mit dem aktuellen Kontostand auf dem Kontoauszug angezeigt. |
| 4 | Keine Zahlungsart angezeigt. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: -  Betrag: 10  Text: Test | Information unter den Radio-Buttons dass die Angabe der Zahlungsart zwingend ist. |
| 5 | Kein Transaktionstext angegeben. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 10  Text: - | Information unter dem Feld «Transaktionstext», dass dieses Feld erforderlich ist. |
| 6 | Kein Betrag angegeben. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: -  Text: Test | Information, dass das Betragsfeld erforderlich ist. |
| 7 | Einzahlung von einem Betrag von mehr als 1'000'000 CHF. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 5'000’000  Text: Einzahlung | Information, dass der Betrag zwischen 0 und 999'999 liegen muss. |
| 8 | Einzahlung von einem negativen Betrag. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: -1000  Text: Einzahlung | Information, dass der Betrag zwischen 0 und 999'999 liegen muss. |
| 9 | Einzahlung von einem Betrag mit mehr als zwei Kommastellen. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 100.255  Text: Einzahlung | Information, dass der Betrag nicht mehr als zwei Kommastellen enthalten darf. |
| 10 | Die Einzahlung resultiert in einem Kontobetrag von mehr als 1'000'000 CHF. | payments.xhtml | Vor diesem Test folgende Einzahlung tätigen:  Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 949’000  Text: Einzahlung  Dann folgende Einzahlung tätigen:  Konto: CH320058558512345601T  Zahlungsart: Einzahlung  Betrag: 100’000  Text: Einzahlung | Die erste Einzahlung sollte erfolgreich sein und der neue Kontostand von 950'000 CHF soll angezeigt werden.  Der zweite Einzahlung soll nicht durchgeführt werden und es erscheint eine Fehlermeldung, dass der Kontostand nicht höher als 999’999 CHF sein darf. |
| 11 | Auszahlung mit einem Kontoüberzug. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Auszahlung  Betrag: 990’000  Text: Auszahlung | Fehlermeldung, dass das Kontolimit erreicht ist und der Nutzer nicht tiefer als CHF 1'000 gehen darf (als Beispiel für dieses Konto). |
| 12 | Buchstaben beim Betrag angeben. | payments.xhtml | Konto: CH320058558512345601T  Zahlungsart: Auszahlung  Betrag: ABC  Text: Auszahlung | Es soll ein Fehlermeldung kommen, dass der Betrag einem double bzw. Integer entsprechen muss. |

### Testprotokoll

Im Testprotokoll sind die Resultate der vorhin formulierten Tests enthalten. Dazu wurde der Standard SQL-Dump ausgeführt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nr** | **Tatsächliches Resultat** | **Status** | **Bemerkung** |
| 1 | Die Auszahlung war erfolgreich mit der Meldung, dass der neue Kontostand CHF -100 beträgt. | OK | - |
| 2 | Die Einzahlung war erfolgreich mit der Meldung, dass der neue Kontostand CHF 1000.0 beträgt. | OK | - |
| 3 | Die Ein- und Auszahlung werden mit dem aktuellen Kontostand auf dem Kontoauszug angezeigt. | OK | - |
| 4 | Fehlermeldung unter den Radio-Buttons mit der Nachricht «Bitte geben Sie an, um welche Zahlungsart es sich handelt.» | OK | - |
| 5 | Fehlermeldung unter dem Feld «Transaktionstext» mit der Nachricht: «Dieses Feld ist erforderlich». | OK | - |
| 6 | Fehlermeldung unter dem Feld «Betrag (in CHF)» mit der Nachricht «Dieses Feld ist erforderlich». | OK | - |
| 7 | Fehlermeldung unter dem Feld «Betrag (in CHF)» mit der Nachricht «Der Betrag muss zwischen 0 und 999'999 liegen.». | OK | - |
| 8 | Fehlermeldung unter dem Feld «Betrag (in CHF)» mit der Nachricht «Der Betrag muss zwischen 0 und 999'999 liegen.». | OK | - |
| 9 | Fehlermeldung unter dem Feld «Betrag (in CHF)» mit der Nachricht «Sie können beim Betrag nicht mehr als zwei Stellen nach dem Komma haben.». | OK | - |
| 10 | Zahlung erfolgreich durchgeführt mit dem neuen Kontostand: 950'000.  Zahlung wurde nicht durchgeführt mit der Meldung, dass das Limit erreicht wurde, es können nicht mehr als 999'999 einbezahlt werden. | OK | - |
| 11 | Zahlung wurde nicht durchgeführt mit der Meldung, dass das Limit erreicht wurde, es können nicht mehr als 999'999 einbezahlt werden. | Not OK | Muss noch gefixt werden. Grund ist, dass im Code bei dieser Überprüfung nicht zwischen Debit und Credit unterschieden wird. |
| 12 | Fehlermeldung unter dem Feld «Betrag (in CHF)» mit der Meldung «Bei ‘ABC’ muss es sich um eine Zahl handeln» | OK | - |

# Anhang

## Quellen

Informationen zur Einbindung von Font Ressourcen:

<http://stackoverflow.com/questions/18000216/glyphicons-dont-get-displayed-properly>

Informationen zum Lazy Loading:

<https://jaxenter.de/was-du-spater-kannst-besorgen-lazy-loading-in-jpa-2-1-2375>

<http://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api>

Informationen zu Beziehungen in JPA:

<https://en.wikibooks.org/wiki/Java_Persistence/ManyToOne>

<http://docs.oracle.com/javaee/6/api/javax/persistence/ManyToOne.html>

<http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html>

Nur bestimmte Spalten im JPA Query selektieren:

<http://stackoverflow.com/questions/24710626/jpa-query-selecting-only-specific-columns-without-using-criteria-query>

Informationen, wie ein Attribut in einer JPA-Entity von JPA ignoriert wird:

<http://stackoverflow.com/questions/2154622/why-does-jpa-have-a-transient-annotation>

Eine Liste von gejointen Elementen sortieren:

<http://stackoverflow.com/questions/3002135/how-to-dynamically-order-many-to-many-relationship-with-jpa-or-hql>

Informationen zu Custom Error Messages und eigene Validation Klassen erstellen:

<https://www.mkyong.com/jsf2/custom-validator-in-jsf-2-0/>

<http://www.mkyong.com/jsf2/customize-validation-error-message-in-jsf-2-0/>

Informationen zu den JSF Scopes:

<http://stackoverflow.com/questions/6025998/difference-between-view-and-request-scope-in-managed-beans/>

# Schlusswort

Zum Schluss kann ich sagen, dass die Probe-IPA sehr gut verlaufen ist. Obwohl ich zwei Tage weniger Zeit hatte, da ich aufgrund des Militärs abwesend war, hatte ich keine Probleme mit der Zeit.

Die Aufgabenstellung war interessant, jedoch vom technischen Aspekt her nicht extrem herausfordernd. Eine grössere Herausforderung war die Arbeit mit dem JavaEE Webframework und den Komponenten und wie sie zusammenhängen.

Zu Beginn war die Planung noch ziemlich parallel zu meiner Arbeitsweise und Geschwindigkeit. Doch gegen Ende nahmen die Unterschiede zum Zeitplan zu. Die Reservezeit war ein wenig zu grosszügig kalkuliert. Diese Zeit hätte ich besser in die einzelnen Arbeitsschritte planen können. Zudem habe ich das Arbeitsprotokoll eher am nächsten Tag geschrieben. Ich hätte direkt während der Arbeit das Protokoll nachführen sollen. Somit hätte ich dies eventuell ein wenig besser formulieren können und wäre am Abend nicht in den Stress gekommen, dieses noch zu schreiben.

Im Grossen und Ganzen hat die Probe-IPA viel Spass gemacht. Ich war konstant beschäftigt, aber nicht überfordert. Ausserdem gab es keine grossen Restriktionen und ich konnte meiner Kreativität freien Lauf lassen. Ich kann nun viel Wissen und Erfahrung in meine «echte» IPA mitbringen, was mir den Ablauf sicher vereinfacht.

1. Quelle: <https://www.mkyong.com/jsf2/jsf-2-templating-with-facelets-example/> [↑](#footnote-ref-1)
2. Quelle: <http://stackoverflow.com/a/18000835> [↑](#footnote-ref-2)
3. Quelle: <http://stackoverflow.com/a/2351231> [↑](#footnote-ref-3)
4. Quelle: <https://de.wikipedia.org/wiki/Java_Persistence_API> [↑](#footnote-ref-4)
5. Quelle: <http://docs.oracle.com/javaee/5/api/javax/persistence/MappedSuperclass.html> [↑](#footnote-ref-5)
6. Quelle: <http://stackoverflow.com/a/17987718> [↑](#footnote-ref-6)
7. Quelle: <http://stackoverflow.com/a/1403734> [↑](#footnote-ref-7)
8. Quelle: <https://jaxenter.de/was-du-spater-kannst-besorgen-lazy-loading-in-jpa-2-1-2375> [↑](#footnote-ref-8)
9. Quelle: <http://stackoverflow.com/a/32440578> [↑](#footnote-ref-9)
10. Quelle: <https://docs.oracle.com/cd/E19798-01/821-1841/giqxy/> [↑](#footnote-ref-10)
11. Quellen:  
    <https://www.mkyong.com/jsf2/custom-validator-in-jsf-2-0/>

    <http://www.mkyong.com/jsf2/customize-validation-error-message-in-jsf-2-0/> [↑](#footnote-ref-11)