|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Oriet, Nils  17. Januar 2018     |  |  | | --- | --- | | Lehrbetrieb | Credit Suisse AG | | Berufsbildner: | Christian Heintz | | Fachvorgesetzter | Martin Haas | | Abteilung | Flow of Funds, SJHH 31 | | Durchführungsdatum: | 17.01.2018 – 07.02.2018 | | Experte: | Martin Haas | | Durchführungsort: | Berufsbildungscenter  Max-Högger-Strasse 2  8048 Zürich | |

|  |
| --- |
|  |
| Erstellen eines Services für die Verwaltung von Handelsöffnungszeiten im Wertschriftenhandel |
| Erstellen eines Services für die Verwaltung von Handelsöffnungszeiten |

**Informationen zum Dokument**

**Versionierung**

|  |  |  |
| --- | --- | --- |
| Autor | Datum | Version |
| Nils Oriet | 17.01.2017 | 0.1 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Freigabe**

Dieses Dokument wird vom Autor für Fachvorgesetzten, sowie Experten jeglicher Stufe freigegeben.

**Zustatzinformationen**

Dieses Dokument wurde im Rahmen einer Individuellen Praktischen Arbeit (IPA) erstellt und behandelt die organisatorischen Aspekte, sowie Planung und Umsetzung eines Projektes, welches den Anforderungen der Aufgabenstellung entspricht.

Inhaltsverzeichnis

[1 Vorwort 5](#_Toc446490989)

[2 Teil 1 – Umfeld und Ablauf der IPA 6](#_Toc446490990)

[2.1 Ausgangslage 6](#_Toc446490991)

[2.2 Detaillierte Aufgabenstellung 7](#_Toc446490992)

[2.3 Mittel und Methoden 8](#_Toc446490993)

[2.4 Projektorganisation 9](#_Toc446490994)

[2.5 Vorkenntnisse 10](#_Toc446490995)

[2.6 Versionisierung / Backup 10](#_Toc446490996)

[2.7 Zeitplan 11](#_Toc446490997)

[2.8 Journal 12](#_Toc446490998)

[3 Teil 2 – Projekt 22](#_Toc446490999)

[3.1 Management Summary 22](#_Toc446491000)

[3.1.1 Ausgangssituation 22](#_Toc446491001)

[3.1.2 Umsetzung 22](#_Toc446491002)

[3.1.3 Ergebnis 22](#_Toc446491003)

[3.2 Projektmanagement und Planung 23](#_Toc446491004)

[3.2.1 Analyse 23](#_Toc446491005)

[3.2.1.1 Projektstruktur 24](#_Toc446491006)

[3.2.2 Design 25](#_Toc446491007)

[3.2.2.1 Datenbank 25](#_Toc446491008)

[3.2.2.2 Ablauf 27](#_Toc446491009)

[3.2.3 Implementation 29](#_Toc446491010)

[3.2.3.1 Entity 29](#_Toc446491011)

[3.2.3.2 Data Access Object 34](#_Toc446491012)

[3.2.3.3 Static Data Accessor 36](#_Toc446491013)

[3.2.3.4 XML Schemas (XSD) 38](#_Toc446491014)

[3.2.3.5 Actions 43](#_Toc446491015)

[3.2.3.6 Validierung 50](#_Toc446491016)

[3.2.3.7 SoapUI Requests 51](#_Toc446491017)

[3.2.4 Testing 53](#_Toc446491018)

[3.2.4.1 Testprotokoll Persistence Layer 54](#_Toc446491019)

[3.2.4.2 Testprotokoll Business Layer 55](#_Toc446491020)

[3.2.4.3 Testprotokoll Integrationstests 57](#_Toc446491021)

[3.2.5 Integration / Wartung 59](#_Toc446491022)

[4 Schlusswort 60](#_Toc446491023)

[5 Glossar & Verzeichnisse 61](#_Toc446491024)

[5.1 Glossar 61](#_Toc446491025)

[5.2 Abbildungsverzeichnis 61](#_Toc446491026)

[5.3 Tabellenverzeichnis 61](#_Toc446491027)

[5.4 Quellenverzeichnis 61](#_Toc446491028)

[6 Anhang 62](#_Toc446491029)

[6.1 SoapUI Requests 62](#_Toc446491030)

[6.2 StreetMonitorOperationsServices.xsd 65](#_Toc446491031)

[6.3 StreetMonitorOperationsBusinessObjects.xsd 67](#_Toc446491032)

[6.4 StreetMonitorOperationsResult.xsd 69](#_Toc446491033)

[6.5 MonitorServiceEndpoint.java 70](#_Toc446491034)

[6.6 MarketTimeOpeningActions.java 71](#_Toc446491035)

[6.7 MarketTimeOpeningValidation.java 81](#_Toc446491036)

[6.8 StaticDataAccessor.java 85](#_Toc446491037)

[6.9 MarketTimeOpeningDao.java 86](#_Toc446491038)

[6.10 TimezoneDao.java 89](#_Toc446491039)

[6.11 MarketTimeOpening.java 90](#_Toc446491040)

[6.12 Timezone.java 95](#_Toc446491041)

[6.13 spring-otex-façade.xml 100](#_Toc446491042)

[6.14 spring-dao.xml 100](#_Toc446491043)

[6.15 persistence-jrave.xml 100](#_Toc446491044)

[6.16 MarketTimeOpeningDaoTest.java 101](#_Toc446491045)

[6.17 MarketTimeOpeningTest.java 104](#_Toc446491046)

# Vorwort

Zurzeit befinde ich mich im letzten Lehrjahr meiner Ausbildung zum Informatiker mit der Fachrichtung Applikationsentwicklung. In diesen vier Jahren konnte ich fundiertes Wissen im Bereich Softwareentwicklung erarbeiten. Zudem wurden mir durch mein Ausbildungsumfeld bei der Credit Suisse viele Kenntnisse über Bankspezifische Prozesse vermittelt.

Während dem Basislehrjahr bei der Santis Training AG, sowie in der Berufsschule wurden mehrere Programmiersprachen erlernt, wobei die Schwerpunkte klar auf PL/1 und Java lagen. Ab dem 2. Lehrjahr begann ich mich auf Java zu spezialisieren und meine Kenntnisse dementsprechend zu vertiefen.  
Seit drei Jahren arbeite ich nun, hauptsächlich mit Java, in verschiedenen Bereichen der Bank und habe meine Kenntnisse stets erweitert. Zurzeit absolviere ich meinen dritten und letzten Praxiseinsatz im OTEx Development Team. OTEx ist der Name unserer Applikation, welche die Schnittstelle zwischen Credit Suisse und verschiedenen Börsen bzw. externen Broker darstellt. Die Applikation sendet erstellte Wertschriftenaufträge an Börsen oder eben Broker.

Diese IPA und die dazugehörige Dokumentation sollen aufzeigen, dass ich im Arbeitsalltag zurechtkomme und erteilte Aufträge korrekt und termingerecht erledigen kann.

Dieses Dokument besteht aus zwei Teilen. Der erste Teil besteht aus der Aufgabenstellung, Umfeld und Ablauf. Der zweite Teil behandelt das Projekt.

# Teil 1 – Umfeld und Ablauf der IPA

Implementieren einer Servicearchitektur für das Verwalten der Öffnungszeiten von Handelsbereichen einer internationalen Bank.

## Ausgangslage

OTEx ist eine Applikation welche für die Auftragsweiterleitung an Handelsplätze (Order Routing) diverser Wertschriftenarten verantwortlich ist. Dabei werden sowohl Kauf- sowie Verkaufsgeschäfte abgewickelt.

Die Applikation basiert zu einem überwiegenden Teil aus Java, welcher auf einer Linux Plattform ausgeführt wird. Für die Speicherung der Daten wird Sybase ASE verwendet.

Damit die Applikation in der Lage ist Aufträge an Handelsplätze weiterzuleiten, müssen Handelsöffnungszeiten beachtet werden. Falls ein Handelsplatz geschlossen ist wird der Auftrag entweder zurückgestellt oder zurückgewiesen.

Die Handelsöffnungszeiten werden in einer statischen Tabelle der Datenbank gespeichert. Diese werden zur Zeit per Script verwaltet. Da dies nicht mehr den heutigen Anforderungen an Nachvollziehbarkeit und Sicherheit entspricht, wurde entschieden dies künftig per GUI zu verwalten.

Ziel dieser IPA ist das Erweiterung der bestehenden Webserviceebene, damit Handelsöffnungszeiten per Administrationsoberfläche verwaltet werden können. Das Frontend (GUI) ist dabei nicht Teil dieser IPA.

Die bestehende Applikation verfügt bereits über die nötige Infrastruktur zur Anbietung von Webservices, die genutzt und erweitert werden soll. Zudem gibt es standardmässig eine Überprüfung der Zugriffssicherheit. (Autorisierung & Authentifizierung)

## Detaillierte Aufgabenstellung

Für die Verwaltung der Handelsöffnungszeiten soll ein Webservice erstellt werden. Der Webservice soll alle CRUD (create, read, update and delete) Operationen unterstützten. Dabei soll für jede Operation jeweils ein Service erstellt werden. Der Webservice ist in die bestehende SOAP Architektur der Applikation zu implementieren.

- Beim „CREATE“ soll ein einzelner Eintrag zuerst validiert und danach in die Datenbank eingefügt werden.

- Beim „READ“ sollen sämtliche Daten inklusive dem Primärschlüssel aus der Tabelle gelesen werden.

- Beim „UPDATE“ soll mit Hilfe des Primärschlüssels (TIME\_INTERNAL\_ID) ein einzelner Eintrag validiert und aktualisiert werden.

- Beim „DELETE“ soll mit Hilfe des Primärschlüssels ein einzelner Eintrag gelöscht werden.

Zusätzlich gibt es technische Attribute, welche nicht im GUI verändert werden sollen . Dies betrifft den Primarschlüssel und die „Last-Update“ Attribute. Diese sollen durch den Service intern abgefüllt werden.

Die Validierung soll sicherstellen das ein Eintrag gültig ist. Falls eine Validierung fehl schlägt soll ein sinnvolle Fehlermeldungen zurückgegeben werden. Die Handelsöffnungszeiten werden mittels Zeitstempel gespeichert. Davon wird nur die Zeitangabe benutzt, beim Datum muss als Wert „01.01.1900“ eingefügt werden. Die Zeitangaben bei den Handelsöffnungszeiten beziehen sich jeweils auf eine definierte Zeitzone. Dabei wird mit einem Fremdschlüssel (TIMEZONE\_INTERNAL\_ID) auf eine andere Tabelle referenziert. Es muss sichergestellt werden, das bei jeder Mutation ein gültige Zeitzone verwendet wird. Die beiden Attribute HOME\_OPENING und HOME\_CLOSING definieren die effektiven Handelsöffnungszeiten und sind deshalb zwingend anzugeben.

Die Attribute der Entität in der zu erstellenden XML Schema Definition (XSD) Schnittstelle müssen mit den Datentypen in der Datenbanken übereinstimmen. Ein Attribut hat in der Datenbank den Datentyp varchar, demzufolge soll es in Java als java.lang.String und in der Schnittstellendefinition (XSD) als xsd:string definiert sein. (namespace-xsd: http://www.w3.org/2001/XMLSchema)

Es ist nicht erforderlich, die Daten auf ihre Datenformat zu prüfen (z.B. ob ein nummerischer Wert keine Buchstaben enthält). Dies ist nämlich bereits durch die Schnittstellendefinition sichergestellt.

Sämtliche Mutationen müssen mit Hilfe der bestehenden „Logging-Infrastruktur“ protokolliert werden. Dabei soll die bestehende log4j Infrastruktur der Applikation verwendet werden.

- Beim Insert soll der neue Dateneintrag protokoliert werden

- Beim Update soll der alte und der neue Datenstand protokoliert werden

- Beim Delete soll der letzte Stand vor dem Delete protokoliert werden

Für jede Operation soll mindestens ein funktionaler Unit Test erstellt werden. Dieser soll den Datenbankzugriff mithilfe der bestehende Testinfrastruktur mocken. Für jede Operation soll mindestens einen Integrationstest erstellt werden, welcher auf dem bestehenden Testframework aufbaut und in der Testdatenbank seine Mutation vornimmt. Zusätzlich soll für jede Validierung ein eigener Test mit entsprechendem Fehlerresultat erstellt werden.

Es existiert noch kein GUI, weshalb das Resultat mit Hilfe des Webservice Testwerkzeugs (SoapUI) getestet werden soll. Für alle Szenarien (Read, Insert, Delete, Update) soll einen Testfall (XML Request) erstellt werden, welcher mittels SoapUI an die laufende Applikation gesendet werden kann.

„READ“: Alle Einträge der Datenbanktabelle inklusive allen Attributen können angezeigt werden.

„INSERT“: Ein einzelner Eintrag kann in die Datenbanktabelle eingefügt werden.

„DELETE“: Ein einzelner Eintrag kann in der Datenbanktabelle gelöscht werden.

„UPDATE“ Ein einzelner Eintrag kann in der Datenbanktabelle aktualisiert werden.

Die Applikation kann lokal gegen eine Entwicklungsdatenbank gestartet werden. Diese Entwicklungsdatenbank ist ein Kopie der produktiven Datenbank. (Aus einer Datenbanksicherung) Damit wird sichergestellt. Damit kann der Webservice mit echten Daten getestet werden. Diese Infrastruktur ist vorhanden und kann direkt verwendet werden.

Für die Dokumentation muss kein Standard eingehalten werden.

## Mittel und Methoden

Ziel ist die Integration in die bestehende Applikation, weshalb die gewohnte Softwareumgebung verwendet werden soll.

Der Webservice Architektur basiert auf Spring-WS, welches auf ein Jetty Webserver deployed ist. Die Webservice Schnittstellen Definition (WSDL) wird mittels XSD während der Laufzeit erzeugt.

Für den Zugriff auf die Datenbank wird JPA (Hibernate) verwendet. Die Applikation lässt sich lokal starten, dabei wird eine lokale Datenbank verwendet. (H2)

Die Tests sollen mit Hilfe von JUnit und Mockito erstellt werden.

## Projektorganisation

|  |  |
| --- | --- |
| Kandidat | Lehrbetrieb |
| Nils Oriet | Credit Suisse AG |
| Tel: +41 44 332 98 87 | Thurgauerstrasse 30/32 |
| Email G: [roman.loosli@credit-suisse.com](mailto:roman.loosli@credit-suisse.com) | 8050 Zürich |
| Email P: roman.loosli@mailtop.ch |  |

|  |
| --- |
| Durchführungsort |
| Credit Suisse AG |
| Uetlibergstrasse 233 |
| 8045 Zürich |

|  |  |
| --- | --- |
| Fachvorgesetzter | Berufsbildner |
| Stefan Lötscher | Christian Heintz |
| Email: stefan.loetscher@credit-suisse.com | Email: christian.heintz@credit-suisse.com |

|  |  |
| --- | --- |
| Experte | Zweit-Experte |
| Igor Bozic | Elisa Valenti |
| Email: bozic.igor@bluewin.ch | Email: elisasofiavalenti@gmail.com |

|  |
| --- |
| Validexperte |
| Renzo Bonfranchi |
| Email: lorenzo@bonfranchi.com |

|  |  |
| --- | --- |
| Thematik | Klassierung |
| Implementieren einer Service-architektur für das Verwalten der Öffnungszeiten von Handelsbereichen einer internationalen Bank. | * Applikationsentwicklung OO * UNIX / Linux * Java / Javascript |

|  |  |
| --- | --- |
| Durchführungsblock |  |
| IPA-Durchführung | 02.03.2016 – 23.03.2016 |
| Erster Expertenbesuch | 02.03.2016 |
| Zweiter Expertenbesuch | 18.03.2016 |
| Präsentation / Fachgespräch | zum Abgabezeitpunkt noch nicht definiert |

## Vorkenntnisse

Der Kandidat hat in den letzten 6 Monaten in unterschiedlichen Bereichen der Applikation gearbeitet, davon 4 Monate im Umfeld der Webservices. Der Kandidat ist zudem mit der Testumgebung vertraut.

Der Kandidat hat bereits Erfahrungen mit der Webservice Architektur der Applikation gesammelt und dabei drei Abfrageservice in einem anderen fachlichen Kontext erstellt.

Zusätzlich hat der Lernende in diesem Jahr ein Service für eine ähnlichen Fall in einem anderen fachlichen Hintergrund erarbeitet. Dieser deckt sich in etwa mit der in der IPA verlangten Ziele ab.

Insgesamt hat der Kandidat bei IPA-Start ungefähr 4 Monate im Webservicebereich der Applikation gearbeitet.

**Bekannte Technologien:**

* Java
* SoapUI
* XML / XSD
* JUnit
* SVN
* Mockito
* Hibernate (JPA)
* ANT (Build Tool)

**Vorarbeiten:**

* Dokumentation Vorlage
* Zeitplan Vorlage
* Projekt wurde bereits in Eclipse aufgesetzt

## Versionisierung / Backup

Die Dokumentation wird mindestens täglich auf einem Subversion Repository gespeichert, damit zu einem späteren Zeitpunkt auf jede Version zugegriffen werden kann.

Mit Hilfe des Tortoise SVN Log kann jede Version einzeln aufgerufen werden und verschiedene Versionen verglichen werden. Das Vergleichen von Versionen ist besonders beim Auftreten von Fehlern sehr hilfreich.

## Zeitplan

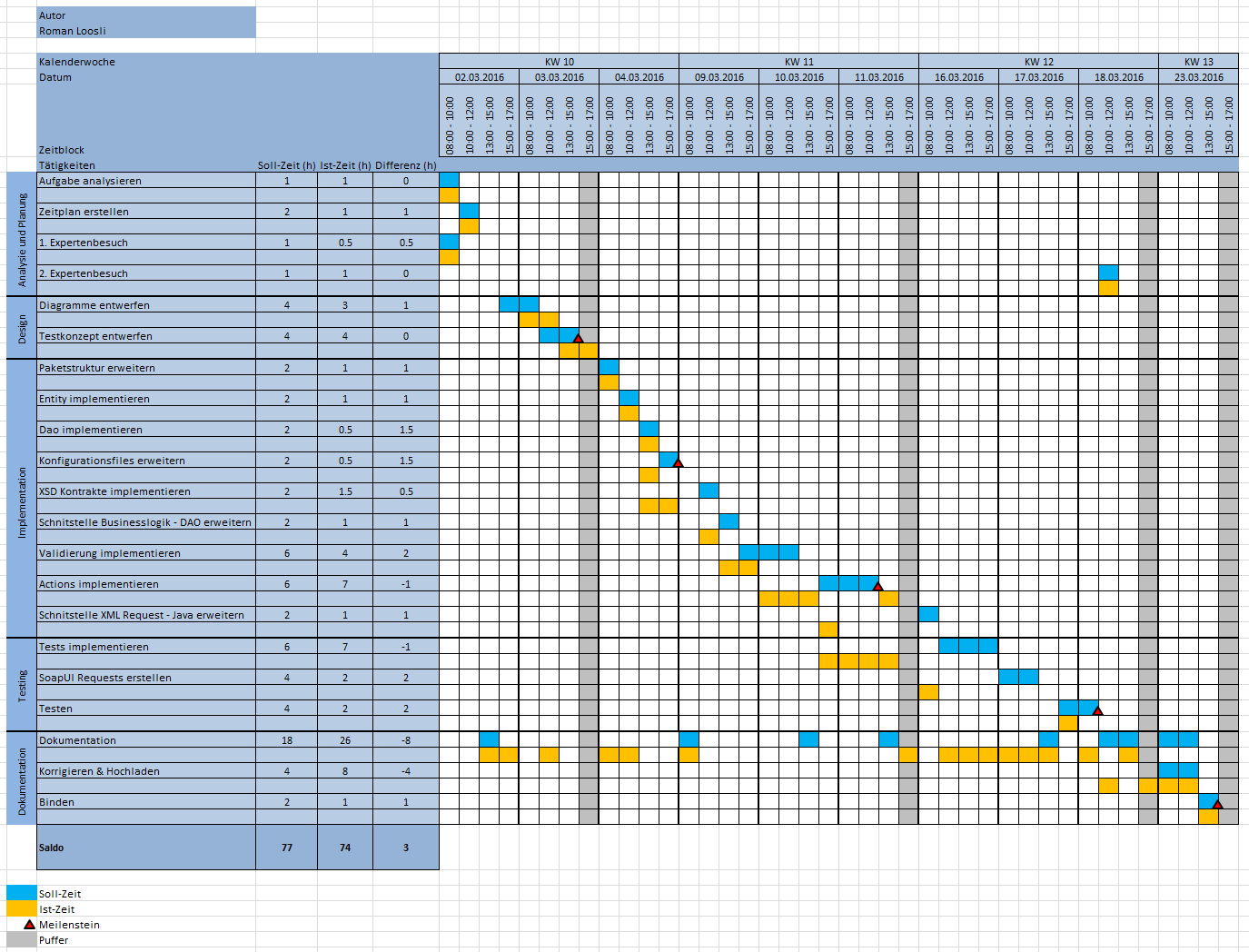


Abbildung 1: Zeitplan

## Journal

Im Journal werden die ausgeführten Tätigkeiten erläutert. Zudem werden allfällige Probleme, sowie deren Lösung erwähnt um daraus auch wieder zu lernen. Am Ende des Tages wir jeweils reflektiert, welche Tätigkeiten gut gelungen sind und welche Aufgaben eher holprig ausgeführt wurden.  
Hilfestellungen und Expertenbesuche werden hier ebenfalls erläutert.

**Tag 1**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Mittwoch, 02.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Ganz zu Beginn meiner IPA habe ich nochmals die Aufgabe im Detail analysiert und mir Gedanken darüber gemacht, wie ich diese in Arbeitspakete unterteilen kann. Anschliessend habe ich diese Arbeitspakete beim Erstellen des Zeitplans festgehalten. Zudem habe ich Teil 1 meiner Dokumentation, bis auf den Zeitplan und das Journal, bereits fertiggestellt. Ausserdem habe ich bereits damit begonnen Diagramme zu entwerfen und einige Punkte zum Thema Analyse und Design zu dokumentieren. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Ich habe mich damit schwer getan, die geeigneten Diagramme für dieses Projekt auszuwählen. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Das sonst so beliebte Klassendiagramm habe ich bei diesem Projekt aussen vor gelassen, weil sich meiner Meinung nach nicht hilfreich ist, wenn man den Rest dieser riesigen Applikation nicht kennt. Darum habe ich mich dafür entschieden als Hauptdiagramm zu meinem Projekt ein Flow Chart zu entwerfen, welches den Ablauf meiner zu schreibenden Komponente dokumentiert. Zudem habe ich ein bescheidenes ERD mit zwei Tabellen erstellt. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| In der Dokumentation habe ich an einigen Ecken schon etwas begonnen zu schreiben, jedoch muss ich diese Themen während der Umsetzung noch etwas vertiefen. Bei der Auswahl und Darstellung der Diagramme habe ich etwas Zeit verloren. Obwohl ich denke, dass ich heute auch ein wenig mehr hätte erreichen können, bin ich zufrieden mit meiner Leistung. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |
| http://www.topseiten.ch/modules/content/images/suchmaschinen_experte.jpg | **Expertenbesuch** |
| Der für mich zuständige Experte, Herr Bozic, hat den ersten Besuchstag gleich auf den ersten Tag angesetzt. Als erstes haben wir uns gegenseitig vorgestellt. Anschliessend haben wir nochmals die wichtigsten Punkte dieser Arbeit besprochen, wobei mir Herr Bozic ein paar Tipps zum IPA-Bericht vorgeschlagen:   * Keine Ich-Formulierungen ausserhalb von Journal und Schlusswort verwenden * Text auf Abbildungen beziehen * Am Anfang eines Kapitels stets eine kurze Einleitung schreiben.   Zuletzt haben wir noch den zweiten Besuchstermin vereinbart. Dieser wird am 18.03.2016 stattfinden. |

**Tag 2**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Donnerstag, 03.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Heute habe ich das Flowchart, welches den Ablauf der Webservices verdeutlichen soll fertiggestellt und einige Teile meiner Dokumentation erweitert. Zudem habe ich das Testkonzept erfolgreich fertiggestellt. Mit dem Abschluss der Design Phase habe ich meinen ersten Meilenstein erreicht. Allerdings habe ich dafür den ersten Puffer gebraucht. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Für die Erstellung des Testkonzepts musste ich die betroffene Datenbanktabelle analysieren. Dabei ist mir aufgefallen, dass für die Öffnungszeiten jeweils zwei Spalten vorhanden waren. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Ich habe mein Problem mit den doppelten Spalten mit meinem Fachvorgesetzten besprochen. Er hat mir erklärt, dass nur die Spalten *HOME\_OPENING* und *HOME\_CLOSING* relevant sind. Die anderen beiden Spalten sind nur Hilfsspalten, ich werde diese jedoch trotzdem in meinem Service validieren und im Entity Objekt einfügen. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Ich bin froh habe ich die Design Phase heute nach Zeitplan abgeschlossen. Ich freue mich morgen endlich mit dem Codieren zu beginnen.  Insgesamt denke ich, dass ich heute wieder einen guten Schritt vorwärts gekommen bin, auch wenn das Dokumentieren etwas schleppend voran ging. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Fachvorgesetzter |

**Tag 3**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Freitag, 04.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Am heutigen Tag stand die Implementation im Vordergrund. Da die Paketstruktur bereits weitgehend vorhanden ist, hat diese Aufgabe sehr wenig Zeit in Anspruch genommen. Allgemein kam es heute zu keinen grösseren Problemen, wodurch ich nun im Zeitplan schon weiter gekommen bin als erwartet. Implementiert wurden heute das Java Entity, sowie das zugehörige Data Access Object. Ausserdem wurden mehrere XML-Konfigurationsfiles angepasst (*persistence-jrave.xml, spring-dao.xml, spring-otex-facade.xml*). Zudem wurden die XSD Kontrakte implementiert und die dazugehörigen Java Klassen generiert. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Da ich meinen Code auf einem SVN Repository ablege, kann es vorkommen, dass andere Teammitglieder am selben Code arbeiten. Heute war dies der Fall und es kam zu einigen Konflikten, als ich meinen Code commiten wollte. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Ich habe das File auf dem Repository, sowie mein aktuelles File gemerged. Dabei habe ich darauf geachtet, dass aus beiden Versionen der ganze Code im endgültigen File vorhanden ist. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Bei der Planung der Implementation habe ich jeweils zusätzliche Zeit eingerechnet, weil unerwartete Probleme auftauchen können, welche viel Zeit beanspruchen. Glücklicherweise war das heute nicht der Fall und ich konnte mir einen Vorsprung gegenüber dem Zeitplan erarbeiten. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Sybase Infocenter, um auf der Sybase Datenbank Abfragen zu Spaltenwerten, wie Länge und nullable zu senden. |

**Tag 4**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Mittwoch, 09.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Heute habe ich gemäss Zeitplan die Schnittstelle zwischen der Businesslogik und den Data Access Objects mit den benötigten Methoden erweitert, sowie die Validierung implementiert. Die Validierung konnte bereits heute vollständig codiert werden, was gemäss Zeitplan erst morgen zu erwarten gewesen wäre. Zu den geplanten Aufgaben kam noch die Implementation einer weiteren Entity Klasse, nämlich *Timezone*, sowie deren DAO. Diese Entity Klasse musste für die Validierung eingebaut werden, da diese den Fremdschlüssel auf die *Timezone* Tabelle überprüft. Im *TimezoneDao* wurden jedoch nur die Funktionen *findAll* und *findById* implementiert.  Ausserdem habe ich ein wenig an meiner Dokumentation gearbeitet, wobei ich bemerkt habe, dass diese mehr Zeit in Anspruch nimmt als erwartet. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Ich habe bemerkt, dass ich beim Erstellen des Zeitplans bei der Implementierung sehr vorsichtig geschätzt habe. Jedoch überziehe ich regelmässig die geplante Zeit für das Dokumentieren. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Bis zu diesem Zeitpunkt hat sich die verlorene Zeit beim Dokumentieren durch den Vorsprung beim Codieren kompensiert. Ich versuche dies beizubehalten. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Nach dem vierten Tag habe ich bemerkt, dass ich die verfügbare Zeit in meinem Zeitplan anders hätte verteilen könne und wohl auch sollen. Nichts desto trotz komme ich rasch vorwärts und befinde mich sogar ein wenig vor dem Zeitplan. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |

**Tag 5**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Donnerstag, 10.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Die Hauptaufgabe heute bestand darin, die Businesslogik umzusetzen. Dafür habe ich die Klasse *MarketTimeOpeningActions* erstellt und darin je eine Methode für die vier Operationen bereitgestellt. Zusätzlich dazu musste ich einige Hilfsmethoden codieren. Beispielsweise mussten sehr oft *TimeStamp* zu *XMLGregorianCalendar* konvertiert werden, da die Entity Klasse das Attribut als *TimeStamp* abgespeichert hat. Bei der Konvertierung der XSD Kontrakte zu Java Klassen jedoch wird der Datentyp *xsd:time* im XSD zu einem *XMLGregorianCalendar* Objekt in Java umgewandelt.  Heute habe ich zudem begonnen JUnit Tests zu implementieren. Ca. 75% der Tests, welche den Persistence Layer betreffen ergeben bereits ein positives Resultat. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Bei der Implementierung der DAO Tests habe ich bemerkt, dass negative Tests mit falschen Fremdschlüsseln reibungslos funktionieren. Dies sollte natürlich nicht der Fall sein, aber nach Absprache mit meinem Fachvorgesetzten kamen wir zum Schluss, dass unsere H2 Datenbank dieses Problem verursacht. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Mein Fachvorgesetzter hat mir mitgeteilt, dass bei den Tests für den Persistence Layer positive Tests ausreichen. Diese überprüfen die *Spring* Funktionalität, was der Hauptaspekt dieser Tests ausmacht. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Die komplette Businesslogik konnte heute implementiert werden, was als sehr positiv zu werten ist. Damit habe ich auch meinen dritten Meilenstein erreicht. Natürlich lässt sich erst nach den Tests sagen, ob noch weiterer Aufwand benötigt wird, um die Funktionalität zu optimieren.  Zudem scheint es mir sehr wichtig, dass ich bereits jetzt damit beginnen kann, Testfälle zu implementieren, damit ich auf allfällige Probleme frühzeitig und mit wenig Zeitdruck reagieren kann.  Insgesamt komme ich mit der Implementation viel schneller voran als erwartet, was mir jedoch mehr Zeit gibt, an meiner Dokumentation zu arbeiten. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Java 6 API (XMLGregorianCalendar)  Fachvorgesetzter |

**Tag 6**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Freitag, 11.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Das implementieren von JUnit Tests, stand heute erneut im Mittelpunkt. Parallel dazu wurden kleine Änderungen an der Businesslogik vorgenommen, denn beim Schreiben der Tests fielen mir plötzlich einige Details auf, welche in der Verarbeitung der Actions angepasst werden müssen. Die Tests für den Persistence Layer, welche gestern nicht fertiggestellt werden konnten, wurden heute vollständig implementiert. Zusätzlich wurden alle bisher festgelegten Tests für den Business Layer heute implementiert und bereits einmal zur Sicherheit ausgeführt. Zudem konnte ich heute die XML Requests in SoapUI implementieren. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Heute hatte ich keine nennenswerten Probleme. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| - |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Da alle Tests erfolgreich gelaufen sind, denke ich, dass meine Implementation ziemlich korrekt den funktionalen Anforderungen entspricht. Allerdings können nie zu viele Tests durchgeführt werden, weshalb ich mir vorstellen kann bei genügend überschüssiger Zeit weitere Tests zu implementieren. Für die nächsten 2 Tage steht nun aber die Dokumentation meiner geleisteten Arbeit im Vordergrund. Bisher konnte ich beim Dokumentieren noch nicht meine gewünschte Leistung bezüglich Effizienz abrufen. Ich hoffe das wird sich in den nächsten Tagen ändern. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |

**Tag 7**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Mittwoch, 16.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Der heutige Tag wurde ausschliesslich zur Erweiterung der Dokumentation genutzt. Die Implementierung der XSD Kontrakte, sowie die Businesslogik habe ich im Projektteil beschrieben. Zudem wurden kleinere Änderungen in der Implementation wahrgenommen, als sie mir gleich ins Auge stachen. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Ich musste heute mit mir selber ringen, wie ich den Implementationsteil meiner Dokumentation aufteilen soll. Natürlich wäre es am schönsten gewesen diesen Teil nach dem Ablauf der Applikation zu strukturieren, allerdings wäre die Struktur Bottom-Up besser für das Verständnis, weil jeweils auf die untere Schicht referenziert wird. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Schlussendlich habe ich die Dokumentation nach Implementationsschritten, also Bottom‑Up strukturiert, um die Abhängigkeiten klarer darzustellen. Wenn eine Abhängigkeit erklärt, wurde die betroffene Komponente bereits vorher im Text erklärt. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Dass ich beim Dokumentieren teilweise holprig vorankomme, habe ich während dieser IPA bereits gelernt. Heute allerdings kann ich grosse Fortschritte vorweisen, da ich mich selber jeweils immer wieder motiviert habe speditiv vorwärts zu arbeiten. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |

**Tag 8**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Donnerstag, 17.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Heute habe ich die komplette Dokumentation zu meiner Implementation fertiggestellt. Zudem habe ich die Kurzfassung geschrieben. Daneben habe ich meine Applikation erfolgreich getestet. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Beim Testen mit JUnit tauchten grössere Probleme mit den Import Statements des Frameworks auf. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| Ich musste probieren welche Import Statements ich alles brauche und welche Methoden ich verwenden soll. Schlussendlich konnte ich alles zum Laufen bringen, indem ich die Methode *assertNotSame* anstelle von *assertNotEquals* verwendet habe. |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Die Dokumentation meiner Implementation fertigzustellen war für mich ein riesen Erfolg und es ist mir eine Last von den Schultern gefallen. Mit dem Testen habe ich zusätzlich einen, im Zeitplan festgelegten, Meilenstein erreicht. In den restlichen zwei Tagen muss ich nur noch das Schlusswort schreiben und die Dokumentation an einigen wenigen Stellen überarbeiten. Am Code muss nichts mehr verändert werden. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| JUnit Dokumentation (Siehe Quellenverzeichnis) |

**Tag 9**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Freitag, 18.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Ich habe heute die Dokumentation vervollständigt. Schlusswort und Management Summary sind geschrieben und der Anhang ist auch im Dokument vorhanden. Zusätzlich habe ich das Feedback von Herrn Bozic dazu genutzt einzelne Abschnitte meiner Dokumentation zu überarbeiten. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Heute hatte ich keine nennenswerten Probleme |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| - |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Wenn ich das grösstenteils fertige Produkt meiner Dokumentation vor mir sehe ist das schon beeindruckend. Zudem bin ich froh, dass ich voraussichtlich am nächsten Mittwoch, dem letzten Tag meiner IPA keine grossen Arbeiten mehr vor mir habe. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |
| http://www.topseiten.ch/modules/content/images/suchmaschinen_experte.jpg | **Expertenbesuch** |
| Herr Bozic hat mich heute zum zweiten Mal besucht. Er hat mir dabei Feedback zu einigen Abschnitten meiner Dokumentation gegeben und mich nochmals über das weitere Vorgehen informiert. Unter anderem hat er mir erklärt, dass das Web Summary bis vor der Präsentation hochgeladen werden muss. Zudem hat er mich nochmals daran erinnert, dass ich ganz sicher bis Mittwoch 18:00Uhr meinen Bericht hochladen soll. |

**Tag 10**

|  |  |
| --- | --- |
| 25796 | **Datum** |
| Mittwoch, 23.03.2016 |
| [Vollbild anzeigen](http://www.stefan-poeppel.de/Musikschule/fragezeichen.gif) | **Tätigkeiten** |
| Der letzte Tag wurde von mir so geplant, dass nicht mehr allzu viele Arbeiten anstehen, so dass er auch als Notfallpuffer benötigt werden könnte. Dies war allerdings nicht nötig. Alle Arbeiten wurden bereits vorher abgeschlossen und heute konnte ich mich darauf konzentrieren, Details in meiner Dokumentation zu überarbeiten. Zudem musste ich das Dokument binden lassen und natürlich bis 18:00 Uhr auf pkOrg hochladen. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\probleme.png | **Probleme** |
| Heute sind keine Probleme aufgetaucht |
| \\chca6021.eur.beluni.net\a329449$\Desktop\lösungsansätze.jpg | **Lösungsansätze** |
| - |
| [Vollbild anzeigen](http://www.minden-luebbecke.de/media/custom/501_77_1_k.GIF) | **Reflexion** |
| Ich bin sehr froh, konnte ich diese 10 Tage erfolgreich hinter mich bringen. Ich denke ich habe eine ansprechende Arbeit abgeliefert und hoffe, dass ich dafür auch belohnt werde. Nach Abgabe der IPA folgt natürlich noch die Präsentation, sowie die Demo und das Fachgespräch. |
| \\chca6021.eur.beluni.net\a329449$\Desktop\Hilfestellung.png | **Hilfestellung** |
| Heute habe ich keine Hilfe benötigt. |

# Teil 2 – Projekt

## Management Summary

### Ausgangssituation

Die OTEx Applikation bildet die Schnittstelle zwischen der Credit Suisse und verschiedenen Börsenplätzen bzw. externen Brokern. Um Aufträge zu diesen weiterzuleiten müssen selbstverständlich die Öffnungszeiten dieser externen Stellen beachtet und akzeptiert werden. Wenn ein neuer Börsenplatz zur Datenbank der Applikation hinzugefügt wird oder ein bestehender Partner seine Öffnungszeiten ändert, müssen diese in unserer Datenbank ebenfalls angepasst werden. Momentan wird jede Änderung der Öffnungszeiten von Börsen oder Broker für unsere Applikation manuell per SQL Script vorgenommen. Dies soll sich in Zukunft ändern und ein GUI zur Anpassung der Tabelle mit den gespeicherten Öffnungszeiten soll entstehen. Dieses GUI ist nicht Teil dieser IPA, jedoch sollen die Webservices, welche zukünftig von diesem GUI aufgerufen werden zur Verfügung gestellt werden. Es werden je eine Insert, Update, Select und Delete Operation implementiert.

### Umsetzung

Es werden vier SOAP Webservices implementiert, welche die vier verschiedenen CRUD Operationen ausführen können. Die Webservices und die dahinter liegende Businesslogik werden in die bestehende Architektur eingegliedert. Alle Benutzereingaben müssen validiert werden, bevor die Operationen auf der Datenbank ausgeführt werden. Zudem sollen mit der bestehenden Logging Infrastruktur die Datenbank Transaktionen aufgezeichnet werden.

### Ergebnis

Die bestehende Architektur wurde erweitert, um die Funktionalen Aspekte der Aufgabenstellung zu erfüllen. Es wurden, wie geplant, vier Webservices erstellt und in die Architektur eingebaut. Bestehende Design Patterns der Architektur wurden übernommen. Alle Anforderungen konnten komplett und im Zeitrahmen eingebaut werden. Jegliche Funktionen können fehlerfrei ausgeführt werden.

## Projektmanagement und Planung

Das folgende Kapitel erklärt wie das Projekt aufgebaut ist und wie die einzelnen Phasen durchgeführt werden. Als Grundlage wird das Wasserfallmodell verwendet, welches in Abbildung 2 zu erkennen ist. Im Team, welches diese Applikation betreut, wird normalerweise die SCRUM Methode benutzt. Allerdings wurde entschieden, dass das Wasserfallmodell für so ein Kleinprojekt besser geeignet ist. Es wurde in 5 Phasen aufgeteilt, wobei die letzte Phase „Integration / Wartung“ nicht Teil dieser IPA ist.

Abbildung 2: Wasserfallmodell

### Analyse

In der ersten Phase des Projektes werden die Anforderungen dieser IPA analysiert. In der Aufgabenstellung sind diese bereits klar definiert:

* SOAP Webservices für alle CRUD Operationen
* Webservices mit SoapUI testen
* XSD Kontrakte erstellen mit Datentypen, welche mit der DB Entität übereinstimmen
* Validierung sämtlicher neu implementierter Webservices
* Technische Attribute im Service selber abfüllen
* Protokollierung mittels log4j
* JUnit Tests

#### Projektstruktur

Um die geforderten Webservices zu implementieren müssen mehrere Teilprojekte des gesamten Projektes bearbeitet werden. In der folgenden Abbildung sind sämtliche Sub-Projekte der OTEx Applikation zu sehen. Die blau markierten Ordner in Abbildung 3 bilden jene Sub-Projekte, welche für die Umsetzung der Aufgabenstellung relevant sind.

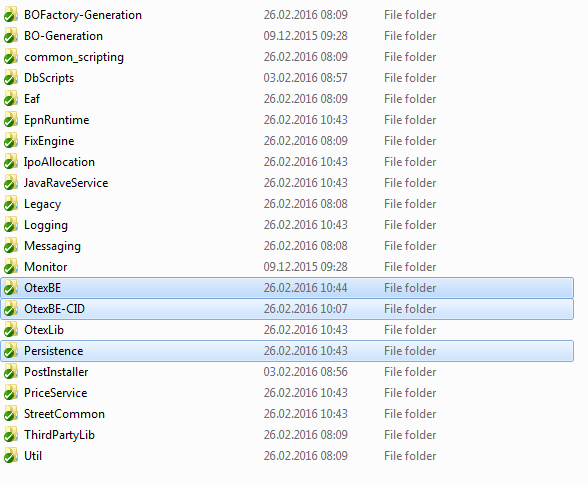


Abbildung 3: Benötigte Sub-Projekte aus dem Gesamtprojekt

Das Projekt *OtexBE* beinhaltet die Businesslogik. Hier werden die Requests validiert und zur Ausführung freigegeben. Im Projekt *OtexBE-CID* befinden sich sämtliche JUnit Tests, mit welchen die Richtigkeit der Businesslogik überprüft wird. Im *Persistence* Projekt befinden sich sämtliche Entity und DAO Klassen, wie auch die dazugehörigen JUnit Tests.

### Design

Hier wird beschrieben, wie das Projekt implementiert werden soll. Es werden Abläufe und bereits vorgegebene Einschränkungen erwähnt. Zudem wird klargestellt, wie mit diesen Einschränkungen gearbeitet wird.

#### Datenbank

Die Tabellen, mit welchen bei diesem Projekt gearbeitet wird, sind bereits implementiert und werden auch nicht verändert. Es handelt sich hierbei um die beiden Tabellen *ref\_MARKET\_TIME* und *ref\_TIMEZONE*.

n:1

Abbildung 4: ERD involvierte Tabellen

Wie in Abbildung 4 zu sehen ist, referenziert die Tabelle *ref\_MARKET\_TIME* auf die Tabelle *ref\_TIMEZONE*. Der Fremdschlüssel befindet sich dabei in der Spalte *TIMEZONE\_INTERNAL\_ID.* Die Zeitzone kann natürlich für mehrere Märkte (Börsen, Broker) verwendet werden. Jeder Markt allerdings richtet seine Öffnungszeiten nur nach einer Zeitzone, weshalb eine n:1 Beziehung erforderlich ist. Die zu erstellenden Webservices bearbeiten ausschliesslich die Tabelle *ref\_MARKET\_TIME*. Es muss jedoch ein Select Query auf die referenzierte Tabelle erstellt werden, um das Vorhandensein des Fremdschlüssels zu überprüfen.

In Tabelle 1 werden die Spalten der Tabelle *ref\_MARKET\_TIME* mit ihren Eigenschaften aufgezeigt.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Datentyp | Kann null sein? | Länge |
| TIME\_INTERNAL\_ID | numeric | Nein | 6 |
| UPDATE\_BIN | varbinary | Ja | 8 |
| TIMEZONE\_INTERNAL\_ID | numeric | Ja | 6 |
| HOME\_OPENING | datetime | Ja | 8 |
| HOME\_CLOSING | datetime | Ja | 8 |
| OPENING\_TIME | datetime | Ja | 8 |
| CLOSING\_TIME | datetime | Ja | 8 |
| LAST\_UPDATED\_BY\_USER | varchar | Ja | 30 |
| LAST\_UPDATE | datetime | Ja | 8 |
| EXECPOOL\_SETTLE\_TIME | datetime | Ja | 8 |

Tabelle 1: ref\_MARKET\_TIME Spaltenbeschreibung

Die Spalte *UPDATE\_BIN* (dunkel hinterlegt) wird nicht in die Java Entity Klasse übernommen, da es sich hier um ein binäres Feld handelt. Die übrigen Attribute werden alle so in der Entity Klasse übernommen.

#### Ablauf

Die Verarbeitung eines validen Requests erfolgt linear und wird in Abbildung 5 grafisch dargestellt. Damit der User die Resultate auf einer Oberfläche ansehen kann, wird der ganze Prozess über SoapUI gestartet. Über SoapUI wird ein XML Request versendet. Während der Laufzeit wandelt JAXB den XML Request mittels XSD Files zu einer Java Klasse um. Dieser Java Request trifft anschliessend im Service Endpoint ein. Der Service Endpoint ist bereits in Spring als Webservice Endpoint definiert.

Der Service Endpoint delegiert die Verarbeitung des Requests an die Businesslogik. In dieser Applikation werden die Klassen, welche Businesslogik beinhalten, Actions genannt. Die *MarketTimeActions* müssen vollständig neu implementiert werden. Der dazugehörende Validator, welcher den Request auf irreguläre Daten prüft, wurde bisher ebenfalls noch nicht implementiert. Bei irregulären Daten im Request soll der Validator eine Liste mit Fehlermeldungen an die Actions zurücksenden. In diesem Fall würde der Prozess die Fehlermeldungen in ein Response Objekt verpacken und dem Service Endpoint zurücksenden. Dieser wiederum würde den XML Request zurück an SoapUI schicken, damit der Benutzer die Fehler auf der Oberfläche sehen kann.

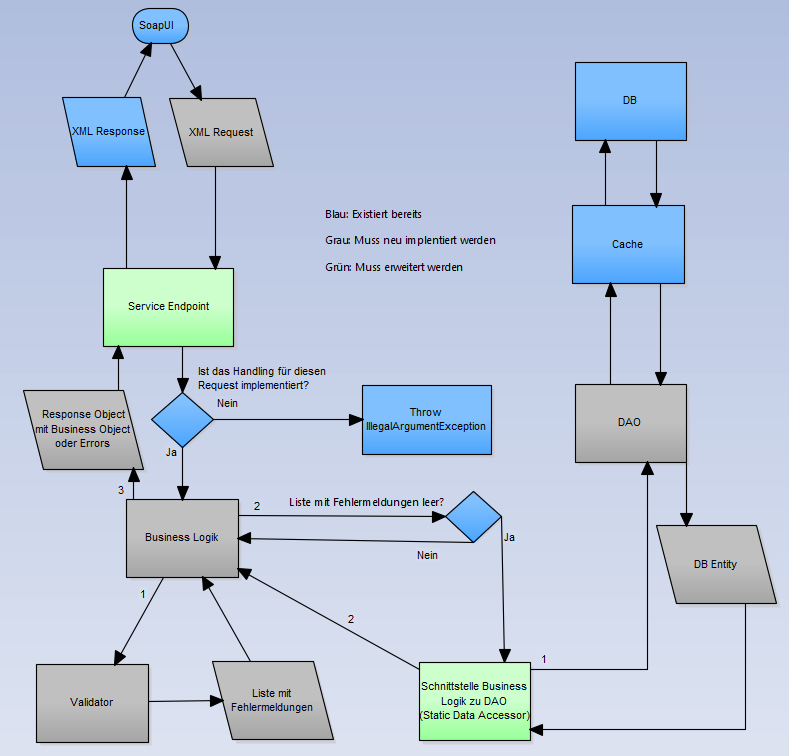


Abbildung 5: Prozessablauf

Falls der Request korrekt abgefüllt wurde, wird, falls notwendig (Update, Insert), in der Action Klasse ein Entity Objekt erstellt welches über die Schnittstelle zwischen Business Layer und Persistence Layer, den *StaticDataAcessor* an das zuständige DAO gesendet wird. Auch falls kein Entity Objekt erstellt werden muss (bei Select und Delete), wird der *Static Data Accessor* aufgerufen. Dies allerdings nur um die gewünschte Aktion weiterzuleiten, beispielsweise die ID für die Delete Operation zu übergeben.

Das DAO führt dann die aufgerufene Operation aus und sendet ein Entity Objekt bzw. eine Liste von Entity Objekten (Select) zurück. In der Businesslogik wird das Entity Objekt in eine Response verpackt und anschliessend über den Service Endpoint an SoapUI zurückgeschickt.

### Implementation

Bei der Implementation besteht die Aufgabe darin, die in der Design Phase beschlossene Lösung so umzusetzen, dass die Applikation einer in der Aufgabenstellung verlangten Lösung entspricht. Im folgenden Kapitel wird die gesamte Implementation des im Kapitel Design geschilderten Prozessablaufs erklärt.

#### Entity

Im Persistence Projekt dieser Applikation existiert bereits ein Entity Objekt mit dem Namen *MartketTime*. Diese Klasse wiederspiegelt jedoch nur die View *view\_MARKET\_TIME*, welche nicht alle Attribute beinhaltet, die für die zu erstellenden Services gebraucht werden. Darum wird wie geplant eine neue Entity Klasse implementiert, welche alle geforderten Attribute enthält. Jedoch wird mit dem Namen auf *MarketTimeOpening* ausgewichen.

**Klassendefinition**

Die Entity Klasse implementiert die Interfaces *PersistenceEntity* und *Serializable*. Der Grund für die Implementierung des *PersistenceEntity* Interfaces besteht darin, dass dies in der jetzigen Architektur zum Normalfall gehört. Durch das Interface wird die Entity Klasse dazu gezwungen die Methode *getId()* zu überschreiben. Das Interface *Serializable* wird implementiert um das Objekt später über HTTP bzw. HTTPS Sessions zu senden.

Zudem muss die Klasse mit mehreren Annotationen markiert werden, welche in Tabelle 2 erläutert werden.

|  |  |  |
| --- | --- | --- |
| Annotation | Bedeutung | Attribute |
| Entity | Die Klasse wird als Entity markiert |  |
| Table | Die zugehörige DB Tabelle wird definiert | name: der Name der Tabelle in der referenzierten Datenbank |
| Cacheable | Definiert ob diese Entity im Cache gespeichert werden soll, falls caching aktiviert ist. | **true** oder **false** |
| Cache | Definiert die Nutzung des Caches | usage: definiert die Zugriffsberechtigung auf diese Entity (Es wird **READ\_WRITE** verwendet, da sowohl gelesen als auch geschrieben werden soll.) |
| Access | Definition der Zugriffsart | AccessType.**FIELD** definiert den Zugriff über getter/setter. (Alternativ wäre **PROPERTY** möglich. Es würde direkt auf die Variable zugegriffen.) |

Tabelle 2: Entity Annotations

Im folgenden Code Snippet wird die Klassendefinition verdeutlicht. Dies beinhaltet sowohl die Anwendung der Annotationen als auch die Implementierung der bereits genannten Interfaces.

@Entity  
@Table(name="ref\_MARKET\_TIME")  
@Cacheable(**true**)  
@Cache(usage=CacheConcurrencyStrategy.***READ\_WRITE***)  
@Access(AccessType.***FIELD***)  
**public** **class** MarketTimeOpening **implements** PersistenceEntity<BigInteger>, Serializable {

**Attribute**

Die Attribute der Entity Klasse werden gemäss Tabelle 1 der Klasse hinzugefügt. Die Spalte *UPDATE\_BIN* wird wie bereits erwähnt nicht im Model implementiert.

Die Konvertierung von DB zu Java bezüglich Namen und Datentypen wird in Tabelle 3 aufgezeigt.

|  |  |  |  |
| --- | --- | --- | --- |
| DB Name | Java Name | DB Datentyp | Java Datentyp |
| TIME\_INTERNAL\_ID | timeInternalId | numeric | BigInteger |
| TIMEZONE\_INTERNAL\_ID | timezoneInternalId | numeric | BigInteger |
| HOME\_OPENING | homeOpening | datetime | TimeStamp |
| HOME\_CLOSING | homeClosing | datetime | TimeStamp |
| OPENING\_TIME | openingTime | datetime | TimeStamp |
| CLOSING\_TIME | closingTime | datetime | TimeStamp |
| LAST\_UPDATED\_BY\_USER | lastUpdatedByUser | varchar | String |
| LAST\_UPDATE | lastUpdate | datetime | TimeStamp |
| EXECPOOL\_SETTLE\_TIME | execpoolSettleTime | datetime | TimeStamp |

Tabelle 3: Konvertierung DB - Java

Die Attribute werden alle mit dem Zugriffsmodifikator private ausgestattet. Zudem werden auch sie wie die Klasse selber mit einer Annotation markiert. Die Annotation @Column markiert ein Attribut als Spalte in der DB Tabelle. Mit welchen Attributen für die bei dieser Annotation gearbeitet wird, wird in der Tabelle 4 beschrieben.

|  |  |
| --- | --- |
| Attribut | Bedeutung |
| name | Name der Spalte in der DB Tabelle |
| unique | boolean – darf ein Wert mehrmals in dieser Spalte vorkommen? |
| nullable | boolean – darf der Wert **null** in dieser Spalte vorhanden sein? |
| length | maximale Länge eines Wertes in dieser Spalte |
| insertable | boolean – ist der Wert im Insert Statement von Hibernate inbegriffen? |
| updatable | boolean – ist der Wert im Update Statement von Hibernate inbegriffen? |

Tabelle 4: @Column Attribute

Zudem muss die ID zusätzlich mit der Annotation @Id markiert werden. Und da die ID beim Insert Befehl automatisch generiert werden soll, muss auch noch die Annotation @GeneratedValue hinzugefügt werden.

Die Umsetzung der Variablen in der Entity Klasse sieht folgendermassen aus:

@Id  
@GeneratedValue(strategy=GenerationType.***AUTO***)  
@Column(name="TIME\_INTERNAL\_ID", unique=**true**, nullable=**false**, length=6, insertable=**true**, updatable=**true**)   
**private** BigInteger timeInternalId;  
  
@Column(name="TIMEZONE\_INTERNAL\_ID", unique=**false**, nullable=**true**, length=6, insertable=**true**, updatable=**true**)   
**private** BigInteger timezoneInternalId;  
  
@Column(name="HOME\_OPENING", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp homeOpening;  
  
@Column(name="HOME\_CLOSING", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp homeClosing;  
  
@Column(name="OPENING\_TIME", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp openingTime;  
  
@Column(name="CLOSING\_TIME", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp closingTime;  
  
@Column(name="LAST\_UPDATED\_BY\_USER", unique=**false**, nullable=**true**, length=30, insertable=**true**, updatable=**true**)   
**private** String lastUpdatedByUser;  
  
@Column(name="LAST\_UPDATE", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp lastUpdate;  
  
@Column(name="EXECPOOL\_SETTLE\_TIME", unique=**false**, nullable=**true**, length=8, insertable=**true**, updatable=**true**)   
**private** Timestamp execpoolSettleTime;

**NamedQueries**

Um die Suchabfragen einfacher aufzurufen, werden diese Queries direkt in der Entity Klasse als Strings definiert. Sie werden mit der Annotation @NamedQueries definiert, um später aus der DAO Klasse darauf zuzugreifen.

@NamedQueries({  
 @NamedQuery(name="MarketTimeOpening.findAll", query="SELECT e FROM MarketTimeOpening e"),  
 @NamedQuery(name="MarketTimeOpening.findById", query="SELECT e FROM MarketTimeOpening e WHERE e.timeInternalId = :timeInternalId")  
})

Zudem wird in diesem Entity auch die *toString* Methode überschrieben, um die Daten beim Loggen übersichtlicher darzustellen.

##### Timezone Entity

Da *Timezone* Objekte gesucht werden müssen um Fremdschlüssel zu überprüfen, wird das Entity *Timezone* ebenfalls implementier. Die Struktur entspricht jener von *MarketTimeOpening*. Es werden ausschliesslich die Spaltenwerte sowie deren Attribute angepasst. Zudem wird auf die Tabelle *ref\_TIMEZONE* referenziert.

##### persistence-jrave.xml

Jedes Entity Objekt muss bei Hibernate als solches registriert sein. Deshalb existiert in im Persistence Layer ein XML File namens *persistence-jrave.xml*, welches keine anderen Aufgaben besitzt, als die Entitätsklassen als solche zu markieren.

<persistence-unit name=*"jrave"* transaction-type=*"RESOURCE\_LOCAL"*>  
 <description>JRave Persistence</description>  
 <provider>org.hibernate.ejb.HibernatePersistence</provider>  
  
 <class>com.csg.street.persistence.MarketTimeOpening</class>  
 <class>com.csg.street.persistence.Timezone</class>  
  
 <exclude-unlisted-classes>true</exclude-unlisted-classes>  
 <shared-cache-mode>ENABLE\_SELECTIVE</shared-cache-mode>  
</persistence-unit>

Die gelb markierten *class* Attribute, stellen die in diesem Projekt erstellten Entity-Klassen dar. Zur besseren Übersicht wurden hier die anderen *class* Attribute weggelassen und nur die Konfigurationen hinzugefügt.

Die Konfigurationen wurden von der Architektur bereits bereitgestellt und nur die markierten zwei Zeilen wurden hinzugefügt.

#### Data Access Object

Das Data Access Object (DAO) dient dazu, den Zugriff auf das Entity Objekt zu abstrahieren. Deshalb werden aus der Businesslogik über eben dieses DAO Datenbankoperationen aufgerufen.

**Aufbau**

Das *MarketTimeOpeningDao* erweitert die Superklasse *JpaDao*. Diese beinhaltet den EntityManager, welcher die Entities verwaltet. Das in diesem Projekt genutzte Framework Spring automatisiert die Prozesse des EntityManagers so weit, dass Entwickler simpel per EntityManager Hibernate-Operationen, wie *merge* oder *remove*, aufrufen können.

Die Aufrufe der Operationen sind in der Klasse *JpaDao* bereits implementiert. Das bedeutet, dass die gewünschten Operationen im DAO nur noch weitergeleitet werden müssen.

**Inhalt**

Die Aufgabenstellung verlangt, dass sämtliche CRUD Operationen ausgeführt werden können, weshalb das DAO diese Operationen beinhaltet. Zusätzlich zur Aufgabenstellung, welche besagt, dass bei der Read Operation sämtliche Datensätze gelesen werden sollen, wird auch noch die Methode *findById(BigInteger id)* bereitgestellt. Diese Methode wird bei einem Update oder Delete Request benötigt um zu überprüfen ob der Datensatz, auf welchem die Operation ausgeführt werden soll, überhaupt existiert. Insgesamt werden damit die folgenden 5 Methoden implementiert:

* insert (**C**reate)
* findAll (**R**ead)
* update (**U**pdate)
* delete (**D**elete)
* findById (Hilfsmethode für Update und Delete)

Als Parameter dient jeweils ein Entity, wobei die Delete und die Select (Read) Methoden Ausnahmen bilden. Für die Delete Operation wird nur der Primärschlüssel als Parameter verlangt. Das dazugehörige Objekt wird direkt im DAO gesucht. Die Select Methode benötigt gar keine Parameter, da bei jedem Aufruf alle Datensätze gelesen werden.

**Umsetzung**

Wie bereits erwähnt, dient die DAO Klasse nur dazu, Datenbankoperationen weiterzuleiten. Dies funktioniert bei allen Operationen identisch, mit Ausnahme der Read Operationen.

Die normale Weiterleitung wird im folgenden Codeausschnitt anhand der Update Operation verdeutlicht. Der übergebene Parameter wird erst überprüft, damit es später nicht zu einem Error von Hibernate führt. Anschliessend wird die *merge* Methode im *JpaDao* aufgerufen. Diese erledigt den Rest. Anschliessend wird das neue bearbeitete Objekt geloggt und der Businesslogik übergeben.

**public** MarketTimeOpening update(MarketTimeOpening entity) {  
 **if**(entity == **null**) { // check to avoid exceptions in hibernate  
 **return** **null**;  
 }  
   
 entity = merge(entity);  
 logEntity("Updated:", entity);  
 **return** entity;  
}

Bei den Read Operationen werden NamedQueries benutzt. Diese wurden im Entity mit der HQL Syntax definiert. Im DAO wird der Aufruf folgendermassen getätigt.

**public** MarketTimeOpening findById(BigInteger id) {  
 **return** **super**.findOneByQuery(getEntityClass().getSimpleName() + ".findById", "timeInternalId", id);  
}

Die *findOneByQuery* Methode wird weist den EntityManager darauf hin, dass er ein einzelnes Objekt per NamedQuery finden muss. Mit dem Klassennamen und dem Querynamen („.findById“) findet der EntityManager das Query nun in der bereits per Generics gemappten Klasse.  
Zusätzlich dazu können Parameter übergeben werden. Im obigen Beispiel ist das der Parameter mit dem Namen *timeInternalId* und dem Wert *id*.

##### Timezone DAO

Das DAO für die Entity Klasse *Timezone* wird analog dem DAO *MarketTimeOpeningDao* implementiert. Allerdings werden ausschliesslich die Operationen *findAll* und *findById* implementiert.

##### spring-dao.xml

Wie die Entity-Klassen bei Hibernate müssen auch die DAO-Klassen bei Spring registriert werden, um darauf Zugriff zu bekommen. Diese werden allerdings nicht als *class* sondern als *bean* definiert, wie im folgenden Ausschnitt aus dem File zu sehen ist.

<bean id="marketTimeOpeningDao" class="com.csg.street.dao.MarketTimeOpeningDao"/>  
<bean id="timezoneDao" class="com.csg.street.dao.TimezoneDao"/>

Im oben stehenden Code sind nur die Einträge zu sehen, welche während diesem Projekt hinzugefügt wurden. Die restlichen ca. 100 Einträge sind in gleichem Stil eingetragen und nach Alphabet sortiert.

#### Static Data Accessor

Die Klasse *StaticDataAccessor* bildet die Schnittstelle zwischen Businesslogik (Actions) und Persistence Layer (DAO). Die Aufgabe dieser Klasse besteht einzig und allein darin, die Datenbankoperationen, welche die Businesslogik ausführen will, an das geeignete DAO weiterzuleiten.

**Methoden**

Für vier verschiedene Datenbankoperationen müssen natürlich auch vier DAO-Zugriffe implementiert werden, welche den CRUD Operationen entsprechen.

/\*

\* This StaticDataAccessor Class represents the interface between business   
\* layer and persistence layer.  
\* Every call on the DAO class should be done in this class.  
\*/  
**public** List<MarketTimeOpening> findAllMarketTimeOpenings() {  
 **return** marketTimeOpeningDao.findAll();  
}  
  
**public** MarketTimeOpening insertMarketTimeOpening(MarketTimeOpening entity) {  
 **return** marketTimeOpeningDao.insert(entity);  
}  
  
**public** MarketTimeOpening updateMarketTimeOpening(MarketTimeOpening entity) {  
 **return** marketTimeOpeningDao.update(entity);  
}  
  
/\*\*  
\* By calling this method the record with the same primary key as the passed   
\* id will be deleted.  
\* If the operation was successful the entity before it got deleted will be   
\* returned. Else the method returns null.  
\*/  
**public** MarketTimeOpening deleteMarketTimeOpening(BigInteger id) {  
 **return** marketTimeOpeningDao.delete(id);  
}

Wie im obige Codestück zu sehen ist, wird ausschliesslich ein (oder sogar kein) Parameter an die zuständige Methode im DAO Objekt übergeben. Zurückgegeben wird jeweils das bearbeitet Objekt. Auch bei der Delete Operation wird das gelöschte Objekt in seinem letzten Zustand zurückgegeben.

Zusätzlich zu diesen Methoden müssen noch zwei weitere Hilfsmethoden erstellt werden. Diese werden teilweise direkt zur Validierung der jeweiligen Requests benötigt, um die Existenz von Primär- oder Fremdschlüssel zu überprüfen.

**public** MarketTimeOpening findMarketTimeOpeningById(BigInteger id) {  
 **return** marketTimeOpeningDao.findById(id);  
}  
  
**public** Timezone findTimezoneById(BigInteger id) {  
 **return** timezoneDao.findById(id);  
}

Beide Hilfsmethoden rufen jeweils ein Query auf, welches einen einzelnen Record liest. Allerdings bezieht sich nur eine auf das *MarketTimeOpeningDao*. Die zweite ruft die Single-Select Methode im *TimezoneDao* auf. Diese wird verwendet, um bei Insert und Update Requests den Fremdschlüssel zur Tabelle *ref\_TIMEZONE* zu überprüfen.

##### spring-otex-facade.xml

Die Argumente des Konstruktors der *StaticDataAccessor* müssen in einem Konfigurationsfile definiert werden. Diese Funktion übernimmt das XML File *spring-otex-facade.xml*. Hier wird zudem auch noch die Konstruktor Signatur für die Klasse *BackendFacade* definiert, welche allerdings für diese IPA nicht verwendet wird.

Wie bereits erwähnt wird die Signatur als XML definiert. Dazu wird der Tag *bean* mit den Attributen *id* und *class* (die dazugehörige Klasse, in diesem Fall *StaticDataAccessor*) verwendet.

<bean id="staticDataAccessor" class="com.csg.otex.guibe.interop.StaticDataAccessor">  
  
 <constructor-arg ref="plainStockExchangeDao" />  
 <constructor-arg ref="brokerDao" />  
 <constructor-arg ref="productInfoService" />  
 <constructor-arg ref="brokerCustRuleDao" />  
 <constructor-arg ref="flipAccountDao" />  
 <constructor-arg ref="monAllBrokersDao" />  
 <constructor-arg ref="monAllClearingBrokersDao" />  
 <constructor-arg ref="monAllElectronicBrokersDao" />  
 <constructor-arg ref="monAllElectronicClearingBrokersDao" />  
 <constructor-arg ref="outgoingChannelDao" />  
 <constructor-arg ref="depositaryDao" />  
 <constructor-arg ref="depositaryIdDao" />  
 <constructor-arg ref="marketDao" />  
 <constructor-arg ref="aesMetaDataDao" />  
 <constructor-arg ref="deskDao" />  
 <constructor-arg ref="handelskreisDao" />  
 <constructor-arg ref="monRejectReasonDao" />  
 <constructor-arg ref="monCancelReasonDao" />  
 <constructor-arg ref="desk2teDao" />  
 <constructor-arg ref="locDestuserDao" />  
 <constructor-arg ref="monTradingAccountDao" />  
 <constructor-arg ref="externalUserDao" />  
 <constructor-arg ref="marketScanResetReasonDao" />  
 <constructor-arg ref="fillPriceCheckDao" />  
 <constructor-arg ref="externalBrokerDao" />  
 <constructor-arg ref="custodianValidationDao" />  
 <constructor-arg ref="tradingEntityDao" />  
 <constructor-arg ref="staticDataService" />  
 <constructor-arg ref="adhocRuleDao" />  
 <constructor-arg ref="marketTimeOpeningDao" />  
 <constructor-arg ref="timezoneDao"/>

</bean>

Das Codestück aus der dem besagten File zeigt, wie die Konstruktor Definition aussieht. Die gelb markierten Zeilen stellen jene zwei Zeilen dar, welche für die zwei DAO’s, welche im *StaticDataAccessor* verwendet werden, erweitert werden mussten.

#### XML Schemas (XSD)

Die XML Requests, welche schlussendlich an die Applikation gesendet werden, müssen sich von der Struktur her an Kontrakte halten, welche per XSD definiert werden.

##### Business Objects

Als Business Objects wurden jene Definitionen benannt, welche die eigentlichen Objekte wiederspiegeln, welche in der Businesslogik zur Ausführung der Operationen verwendet werden. Ihr Inhalt ist pro Operation unterschiedlich, wobei sich die geforderten Parameter bei einigen Operationen überschneiden. Die Business Objects sind im File *streetMonitorOperationsBusinessObjects.xsd* definiert.

**MarketTimeOpeningGeneralBOAttributes**

Die Gruppe *MarketTimeOpeningGeneralBOAttributes* wurde implementiert um die generellen Attribute der Tabelle zu gruppieren, da diese bei der Insert und Update Operation verwendet werden. Zudem werden diese Attribute auch beim Rückgabe-XML für die Select Operation verwendet.

Im Gegensatz zum Originalcode wurden die Elemente hier durch einen Zeilenumbruch getrennt, um den Code übersichtlicher zu gestalten.

<xsd:group name=*"MarketTimeOpeningGeneralBOAttributes"*>  
 <xsd:sequence>  
 <xsd:element name=*"timezoneInternalId" type="str:IdRestriction"* minOccurs=*"1"* maxOccurs=*"1"* nillable=*"false"*/>  
  
 <xsd:element name=*"homeOpening"* type=*"xsd:time"* minOccurs=*"1"* maxOccurs=*"1"* nillable=*"false"*/>  
  
 <xsd:element name=*"homeClosing"* type=*"xsd:time"* minOccurs=*"1"* maxOccurs=*"1"* nillable=*"false"*/>  
  
 <xsd:element name=*"openingTime"* type=*"xsd:time"* minOccurs=*"0"* maxOccurs=*"1"* nillable=*"false"*/>  
  
 <xsd:element name=*"closingTime"* type=*"xsd:time"* minOccurs=*"0"* maxOccurs=*"1"* nillable*="false"*/>  
  
 <xsd:element name=*"execpoolSettleTime"* type=*"xsd:time"* minOccurs=*"1"* maxOccurs=*"1"* nillable=*"false"*/>  
 </xsd:sequence>   
</xsd:group>

Diese Gruppe beinhaltet dieselben Attribute wie das Entity *MarketTimeOpening* mit Ausnahme des Primärschlüssels und den beiden Last Update Attributen, welche erst intern abgefüllt werden und daher beim Request noch keine Rolle spielen.

Für die Attribute, welche in der Datenbank als *TimeStamp* gespeichert werden wurde im Schema der Datentyp *xsd:tme* verwendet. Dieser verlangt vom gesendeten XML File eine Zeitangabe im Format *hh:mm:ss*.

Die Elemente *openingTime* und *closingTime* wurden auf *nillable=false* gesetzt, aber trotzdem mit *minOccurs=“0“* markiert. Der Grund dafür liegt darin, dass *not nillable* nur bedeutet, dass das Element im XML vorhanden sein muss, jedoch nicht abgefüllt. Bei *minOccurs=“1“* jedoch müsste auch noch ein valider Wert mitgegeben werden. Somit muss das Element nun im XML Request, welcher das Business Object beinhaltet, vorhanden sein. Es darf jedoch einen leeren Inhalt vorweisen.

Eine Besonderheit zeigt sich beim Element für den Fremdschlüssel, *timezoneInternalId*. Dieses muss nicht von einem XSD Datentyp stammen. Es wurde für die Primärschlüssel ein eigener Datentyp implementiert, welcher sicherstellt, dass der Wert im positiven Bereich liegt und nicht mehr als 10 Stellen lang ist.

<!-- Per default ID Attributes have a maximum length of 10 digits. -->  
<xsd:simpleType name=*"IdRestriction"*>  
 <xsd:restriction base=*"xsd:positiveInteger"*>  
 <xsd:totalDigits value=*"10"*/>  
 </xsd:restriction>  
</xsd:simpleType>

Der implementierte Datentyp erbt vom XSD Datentyp *positiveInteger*. Dieser stellt sicher, dass der Wert eine ganze Zahl ist und im positiven Bereich liegt. Zudem wurde noch eine XSD Restriction namens *totalDigits* hinzugefügt, welche sicherstellt, dass der Wert nicht mehr als 10 Stellen lang ist.

**MarketTimeOpeningGeneralBO**

Die Gruppe *MarketTimeOpeningGeneralBOAttributes* wird nur vom *complexType* mit dem Namen *MarketTimeOpeningGeneralBO* verwendet. Dieser ComplexType beinhaltet allerdings sonst keine anderen Elemente. Der Grund für die Erstellung dieses Typs liegt darin, dass JAXB aus einer Gruppe keine Klasse generiert, aus einem ComplexType jedoch schon. Und mit einem eigenen Objekt für diese Attribute verringert sich der doppelte Code bei der Konvertierung des Businessobjekts in eine Java Entity, sowie umgekehrt. Die Gruppe mit den Attributen wurde nicht zusätzlich als ComplexType definiert, da dies bei der Verwendung eines Attributs jedes Mal den Aufruf eines weiteren Getters zur Folge hätte. Mit der jetzigen Implementierung sind die Attribute jedoch direkt im *MarketTimeOpeningGeneralBO* verfügbar. Zudem kann das *MarketTimeOpeningSelectBO* direkt auf die Attribute zugreifen, was nötig ist, damit in der Response alle Elemente auf einer Ebene dargestellt werden.

**MarketTimeOpeningInsertBO**

Für den Insert sind keine zusätzlichen Attribute zu den der *MarketTimeOpeningGeneralBOAttributes* Gruppe erforderlich. Deshalb wurde auch nur ein Verweis auf diese Gruppe implementiert.

<xsd:complexType name=*"MarketTimeOpeningInsertBO"*>  
 <xsd:sequence>  
 <xsd:element name=*"marketTimeOpening"* type=*"str:MarketTimeOpeningGeneralBO"* minOccurs=*"1"* maxOccurs=*"1"*/>  
 </xsd:sequence>  
</xsd:complexType>

Das einzige verlangte Element, in Tat und Wahrheit eine Gruppe, muss jedoch vorhanden sein und zwar komplett. Dies wird durch die Attribute *minOccurs* und *maxOccurs* sichergestellt.

**MarketTimeOpeningUpdateBO**

Der einzige Unterschied zwischen dem Update und dem Insert Business Object liegt im verlangten Primärschlüssel. Um einen Datensatz zu updaten muss dieser als Referenz vorhanden sein. Im Schema wird er mit dem Datentyp *IdRestriction* eingefügt und ist zwingend erforderlich.

<xsd:element name=*"timeInternalId"* type=*"str:IdRestriction"* minOccurs=*"1"* maxOccurs=*"1"*/>

**MarketTimeOpeningSelectBO**

Im Gegensatz zu den anderen Business Objects wird das *MarketTimeOpeningSelectBO* nicht beim Request sondern beim Response verwendet. Dies ist deshalb der Fall, weil beim Request gar keine Parameter benötigt werden, weil immer alle Records zurückgegeben werden. Bei der Response jedoch muss jedes Objekt in per XML gesendet werden, weshalb der Kontrakt in die andere Richtung nötig ist.

<xsd:complexType name=*"MarketTimeOpeningSelectBO"*>  
 <xsd:sequence>  
 <xsd:element name=*"timeInternalId"* type*="str:IdRestriction"* minOccurs=*"1"* maxOccurs=*"1"*/>  
  
 <xsd:group ref=*"str:MarketTimeOpeningGeneralBOAttributes"*/>  
  
 <xsd:element name=*"lastUpdatedByUser"* type=*"xsd:string"* minOccurs=*"1"* maxOccurs=*"1"*/>  
  
 <xsd:element name=*"lastUpdate"* type=*"xsd:dateTime"* minOccurs=*"1"* maxOccurs=*"1"*/>  
  
 </xsd:sequence>  
</xsd:complexType>

Zusätzlich zu den *MarketTimeOpeningGeneralBOAttributes* wird aus der Datenbank der Primärschlüssel sowie die beiden Attribute *lastUpdatedByUser* und *lastUpdate* zurückgesendet. Die beiden „last Update“ Attribute werden weder beim Insert noch beim Update Befehl dem Request mitgegeben. Dies ist der Fall, weil diese Attribute in der Businesslogik direkt verarbeitet werden.

##### Requests

Grundsätzlich wird für jede Operation ein Request/Services implementiert. Diese werden im File *streetMonitorOperationsServices.xsd* implementiert. Jedoch werden die Services für Insert und Update Operationen durch Business Objects erweitert. Der Basis-Request *OperationServiceRequest* stand bereits in der Applikation bereit und sieht folgendermassen aus.

<xsd:complexType name=*"OperationServiceRequest"* abstract=*"true"*>  
 <xsd:complexContent>  
 <xsd:extension base=*"str:RequestItem"*>  
 <xsd:attribute name=*"serviceId"* type=*"xsd:string"* use=*"required"* />  
 </xsd:extension>  
 </xsd:complexContent>  
</xsd:complexType>

Dem Basis-Request muss also bloss das Attribut *serviceId*, sowie das vom *RequestItem* geerbte Attribut *ItemId* mitgegeben werden.

**SelectMarketTimeOpening**

Dieser Request braucht keine Parameter, da laut Aufgabenstellung immer alle Records gelesen werden sollen. Der Request *SelectMarketTimeOpening* erbt also alles vom *OperationServiceRequest* und erweitert diesen nicht. Es muss trotzdem ein neuer Typ erstellt werden, da beim Ausführen des Request nicht geprüft werden kann, ob der Request ein *OperationServiceRequest* ist, weil alle Requests darauf ansprechen würden.

**DeleteMarketTimeOpening**

Die Delete Operation benötigt ausser dem Primärschlüssel keine weiteren Angaben, weshalb dieser Service auch nicht durch ein Business Object erweitert wird, sondern das Element für den Primärschlüssel einfach direkt in den Service implementiert.

<xsd:element name=*"recordId"* type=*"str:IdRestriction"* minOccurs*="1"* maxOccurs=*"1"* />

Da das Element einen Primärschlüssel abbilden soll, wird es wird mit dem Typ *IdRestriction* definiert, da dieser sicherstellt, dass der Primärschlüssel im positiven Bereich und maximal 10 Stellen lang ist.

**Insert-/UpdateMarketTimeOpening**

Die Requests für die Insert und die Update Operation werden durch ihre jeweiligen Business Objects erweitert und auch mit dem Namen *businessObject* dem Element angefügt.

##### Responses

Die Kontrakte für die Responses stellen jene Schema Kontrakte dar, welche bei der Übermittlung der Resultate benutzt werden. Auch hier bestand bereits vor dieser IPA ein Basis Typ, welcher grösstenteils in diesem Zustand benutzt wurde.

<xsd:complexType name=*"OperationServiceResponse"*>  
 <xsd:sequence>  
 <xsd:element name=*"status*" type=*"str:Status"* minOccurs*="1"*/>  
  
 <xsd:element name=*"errors"* type=*"xsd:string"* minOccurs=*"0"*   
 maxOccurs=*"unbounded"* />  
  
 <xsd:element name=*"recordNumber"* type=*"xsd:integer"* minOccurs=*"0"* maxOccurs=*"1"*/>  
  
 <xsd:element name=*"serviceId"* type=*"xsd:string"* />  
 </xsd:sequence>  
</xsd:complexType>

Die wichtigsten Elemente bei der Basis Response bilden *status, errors* und *recordNumber*.  
Das Element *errors* liefert alle Fehlermeldungen, welche während der Validierung entstanden sind und ist darum auch als *maxOccurs=“unbound“* definiert. Das bedeutet, dass in der generierten Java Klasse das Attribut *errors* als List<String> definiert wird.

Das Element mit dem Namen *recordNumber* liefert einen Integer, welcher den Primärschlüssel des bearbeiteten Records beinhaltet.

Im Status wird ein Wert der Enumeration *Status* verlangt. Dieser zeigt, ob die Operation fehlerfrei verlaufen ist, oder fehl schlug. Die Enumeration wird aus einem SimpleType im File *streetMonitorOperationsResult.xsd* generiert.

<xsd:simpleType name=*"Status"*>  
 <xsd:restriction base=*"xsd:string"*>  
 <xsd:enumeration value=*"OK"* />  
 <!-- Everything was OK -->  
  
 <xsd:enumeration value=*"SEVERE\_ERROR"* />  
 <!-- Severe technical problem in backend -->  
  
 <xsd:enumeration value=*"SOFT\_ERROR"* />  
 <!-- Soft Error: Errors in validation, but can be forced -->  
  
 <xsd:enumeration value=*"HARD\_ERROR"* />  
 <!-- Hard Error: Errors in validation, not forceable, must be changed -->  
 </xsd:restriction>

</xsd:simpleType>

In den Kommentaren zur Status Enumeration wird ersichtlich für was die einzelnen Werte stehen. Für die Webservices, welche in dieser IPA erstellt werden, werden nur die Werte *OK* und *HARD\_ERROR* benutzt.

**SelectMarketTimeOpeningResponse**

Für die Select Operation musste die Basis Response erweitert werden, da eine Liste von Objekten zurück gegeben werden muss. Die Erweiterung wird im selben File implementiert und trägt den Namen *SelectMarketTimeOpeningResponse*.

<xsd:element name=*"marketTimeOpenings"* type=*"str:MarketTimeOpeningSelectBO"* minOccurs=*"0"* maxOccurs*="unbounded"*/>

Der ComplexType wird wie gewöhnlich implementiert (zu sehen bei Business Objects). Als BaseType wird *OperationServiceResponse* verwendet. Als einziges Element wird eine Liste (*maxOccurs=“unbound“*) von *MarketTimeOpeningSelectBO* definiert.

#### Actions

Die Klasse *MarketTimeOpeningActions* beinhaltet die Business Logik, verarbeitet also die Requests und ruft den StaticDataAccessor auf, um die Operationen auszuführen. Zu Beginn der Verarbeitung wird ausserdem die Validierung durchgeführt. Da diese in einer separaten Klasse stattfindet, wird sie in diesem Dokument auch in einem eigenen Kapitel (3.2.3.6 Validierung) beschrieben.

Die Klasse ist so aufgebaut, dass eine *public* Methode den Request entgegen nimmt und an eine interne private Methode weiterleitet. Dies wird so implementiert, weil es in der bisherigen Architektur als Standard angewendet wird.

**Attribute**

Die Klasse besitzt nur die nötigsten Attribute. Dazu gehören ein Logger, sowie eine Instanz des *StaticDataAccessor,* eine Instanz der Validierungsklasse *MarketTimeOpeningValidation* und eine Liste mit Strings für die Fehlermeldungen.

**private** **static** Logger *LOGGER* =Logger.*getLogger*(MarketTimeOpeningActions.**class**);  
**private** StaticDataAccessor staticDataAccessor;  
  
**private** MarketTimeOpeningValidation validation;  
**private** List<String> errorMessages = **new** ArrayList<String>();

Der Logger wird dabei von der **log4j** Library von Apache verwendet. Die Attribute, welche nicht direkt mit einer neuen Instanz abgefüllt werden, werden noch im Konstruktor abgefüllt.

**Konstruktor**

Der Konstruktor wird aus dem Webservice aufgerufen, nachdem dieser den Request dieser Action Klasse zugeordnet hat. Als Parameter wird nur ein Objekt der *BackendFacade*, welche unter anderem eine Referenz zum *StaticDataAccessor* beinhaltet, benötigt.

**public** MarketTimeOpeningActions(BackendFacade backendInteraction) {  
 **super**(backendInteraction);  
 **this**.staticDataAccessor = backendInteraction.getStaticDataAccessor();  
 **this**.validation = **new** MarketTimeOpeningValidation(staticDataAccessor);  
}

Im Konstruktor wird zuerst die Superklasse *Actions* mit einem *BackendFacade* Objekt instanziiert. Anschliessend wird eine Referenz zum *StaticDataAccessor* für diese Klasse bereitgestellt, sowie die Vaidierungsklasse instanziiert.

**Methoden allgemein**

Alle Methoden, mit Ausnahme der Select Action, werden mit denselben zwei Parametern aufgerufen. Der *OperationHeader* beinhaltet Informationen über den gesendeten Request, wie zum Beispiel der Username des Absenders. Als zweiter Parameter wird der effektive Request übergeben, welcher dann weiter verarbeitet wird. Für die Select Action werden keine Parameter benötigt, da der Request an sich keinen Parameter beinhaltet und keine Infos, wie Username, benötigt werden, da der User die Tabelle nicht bearbeitet.

Der Ablauf der Verarbeitung erfolgt jeweils in drei Schritten:

1. Validierung
2. Ausführung der Operation
3. Zusammenstellen der Response

Danach wird die Response an den Webservice Endpoint zurückgeschickt.

##### Insert

Die Verarbeitung des Insert Requests findet in der Methode *insertMarketTimeOpeningInternal* statt. Sie findet in denselben drei Schritten statt wie in Kapitel 3.2.3.5 bereits erwähnt wurde.

Die Validierung sendet eine Liste mit allfälligen Fehlermeldungen. Auf dieser Liste wird auf ihren Inhalt geprüft. Wenn keine Validierungsfehler aufgetaucht sind, ist die Liste leer.

**if**(errorMessages.size() == 0) { // list is empty if validation is successful

Wenn die Liste *errorMessages* keine Einträge beinhaltet, kann mit der Ausführung der Operation begonnen werden. Nach einer erfolgreichen Validierung wird in einem ersten Schritt der Request zu einem Entity Objekt umgewandelt. Dies erfolgt über die Methode *convertInsertRequestToEntity*.

// convert request values to insertable entity  
MarketTimeOpening entity = convertInsertRequestToEntity(header, request);

In der aufgerufenen Methode werden die technischen Attribute *lastUpdate* und *lastUpdatedByUser* abgefüllt. Zudem werden die restlichen Attribute, welche im XSD Kontrakt in der Gruppe *MarketTimeOpeningGeneralBOAttributes* definiert wurden in der Methode *convertGeneralBOToEntity*, welche im Kapitel 3.2.3.5.5.1 beschrieben wird, in das Entity Objekt übertragen. Als Rückgabewert dient ein Entity Objekt der Klasse *MarketTimeOpening*.

/\*\*

\* Converts the business object in the insert request to an entity object.

\*/ **private** MarketTimeOpening convertInsertRequestToEntity(OperationHeader header, InsertMarketTimeOpening request) {  
 MarketTimeOpening entity = convertGeneralBOToEntity(request.getBusinessObject().getMarketTimeOpening());  
  
 entity.setLastUpdate(**new** Timestamp(**new** Date().getTime()));  
 entity.setLastUpdatedByUser(header.getUsername());  
 **return** entity;  
}

Im Code Ausschnitt ist zu sehen, dass das Attribut *lastUpdatedByUser* mit dem Username abgefüllt wird, welcher bereits mit dem Request im OperationHeader übergeben wurde.

Das Attribut *lastUpdate* wird ebenfalls bereits in der Businesslogik gesetzt. Alternativ hätte es auch erst im DAO, also zum letztmöglichen Zeitpunkt, mit dem *TimeStamp* versehen werden können. Allerdings wurde entschieden, dass das DAO ausschliesslich dazu dient, Transaktionen auf die Datenbank weiterzuleiten.

Als nächster Schritt wird der *StaticDataAccessor* aufgerufen um die effektive Operation durchzuführen. Dazu wird die Insert-Methode mit dem konvertierten Entity als Parameter aufgerufen.

MarketTimeOpening execResult = staticDataAccessor.insertMarketTimeOpening(entity);

Als Resultat wird das Objekt erwartet, welches in die Datenbank eingefügt wurde. Falls die Transaktion fehlgeschlagen ist, wird ein der Wert *null* zurückgegeben.

**if**(execResult!= **null**) { // if insert fails, null will be returned

Falls diese Abfrage tatsächlich zutrifft, handelt es sich um einen Spezialfall, da die Validierung bereits ergeben hat, dass der Request zulässig ist. In diesem Fall wird eine Fehlermeldung der Liste *errorMessages* angehängt, welcher besagt, man müsse das den Log überprüfen.

**else** {  
errorMessages.add(MarketTimeOpening.class.toString() + " object could not be inserted. Check log. " + entity.toString());  
}

Beim Normalfall, also wenn das eingefügte Objekt zurückgeschickt wird, wird anschliessen die Response mit dem Status *OK* versehen und der Primärschlüssel des eingefügten Datensatzes zur Response hinzugefügt, damit ersichtlich ist, welcher Record bearbeitet wurde. Zu guter Letzt wird das eingefügte Objekt mit seiner *toString* Methode geloggt. Anschliessend wird die Response zum Webservice Endpoint zurück gesendet.

response.setStatus(Status.***OK***);

response.setRecordNumber(execResult.getTimeInternalId());

*LOGGER*.info("Inserted: " + execResult.toString());

**return** response;

Falls allerdings die Validierung fehlschlägt oder vom DAO kein Objekt zurückgeschickt wird, werden alle Meldungen, welche sich in der Liste *errorMessages* befinden, geloggt und der Response hinzugefügt.

response.getErrors().add("Failed to insert " + MarketTimeOpening.**class**.toString());  
**for**(String em : errorMessages) {  
 response.getErrors().add(em);  
 *LOGGER*.info(em);  
}

Zusätzlich wird der Status in der Response auf *HARD\_ERROR* gesetzt bevor die Response an den Webservice Endpoint zurück gesendet wird.

##### Update

Da sich die Insert und Update Operationen im Allgemeinen sehr ähnlich sind, schneiden sich auch die Verarbeitung der beiden verschiedenen Requests grösstenteils, weshalb in diesem Kapitel nur die Differenz aufgezeigt wird.

Direkt nach einer erfolgreichen Validierung wird der Record, welcher bearbeitet werden soll, mit seinem Primärschlüssel als Entity Objekt aus der Datenbank geholt.

MarketTimeOpening oldEntity = staticDataAccessor.findMarketTimeOpeningById(request.getBusinessObject()  
.getTimeInternalId());

Der Ausschnitt oben stellt eine einzige Zeile dar und soll aufzeigen, wie über den *StaticDataAccessor* das Objekt mit seinem Primärschlüssel selektiert wird. Der Grund, wieso dies so gehandhabt wird, liegt in der Aufgabenstellung. Bim Update muss das Objekt vor und nach der Mutation geloggt werden, weshalb in der Variablen *oldEntity* die bisherige Version gespeichert wird.

Als nächster Schritt wird, wie bei der Verarbeitung eines Insert Requests, ein Entity aus dem Request generiert. Allerdings muss hier eine eigene Methode implementiert werden, da sich der Request *UpdateMarketTimeOpening* vom *InsertMarketTimeOpening* Request darin unterscheidet, dass zusätzlich eine ID mitgesendet wird. Deshalb mussten zwei verschiedene Methoden implementiert werden, welche allerdings für die Konvertierung der *GeneralBOAttributes* auf dieselbe Methode *convertGeneralBOToEntity* zugreifen.

/\*\*  
 \* Converts the business object in the update request to an entity object.  
 \*/  
**private** MarketTimeOpening convertUpdateRequestToEntity(OperationHeader header, UpdateMarketTimeOpening request) {  
  
 MarketTimeOpening entity = convertGeneralBOToEntity(request.getBusinessObject().getMarketTimeOpening());  
  
 entity.setLastUpdate(**new** Timestamp(**new** Date().getTime()));  
 entity.setLastUpdatedByUser(header.getUsername());  
  
 entity.setTimeInternalId(request.getBusinessObject()  
 .getTimeInternalId());  
 **return** entity;

}

Die gelb markierte Zeile im Code Ausschnitt, stellt die Zeile dar, in der sich die Methode *convertUpdateRequestToEntity* von der Methode *convertInsertRequestToEntity* unterscheidet.

Der letzte Unterschied zur Verarbeitung einer Insert Operation liegt darin, dass nach einer erfolgreichen Mutation des Datensatzes der Status vor und nach der Mutation geloggt wird.

*LOGGER*.info("Updated: " + MarketTimeOpening.**class**.toString());  
*LOGGER*.info(oldEntity.toString()); // log before update  
*LOGGER*.info(execResult.toString()); // log after update

Die restliche Verarbeitung mit dem Setzen des Status und dem Verarbeiten der Fehlermeldungen findet exakt gleich, wie beim Insert Request statt.

##### Delete

Die Methode zur Verarbeitung des Delete Requests kann einiges kürzer gehalten werden, als jene für die Operationen Insert und Update. Grund dafür ist, dass beim Delete keine Entity erstellt werden muss, weder beim Request noch bei der Response. Das Entity für das Logging wird direkt beim Ausführen der Operation als Return Wert mitgegeben. Sprich, wird ein Record gelöscht muss das DAO diesen erst mittels Primärschlüssel suchen und anschliessend löschen. Bei einer erfolgreichen Durchführung wird das vorher gesuchte Objekt zurückgesendet. Das ganz passiert nachdem die Validierung erfolgreich abgeschlossen wurde.

MarketTimeOpening execResult = staticDataAccessor.deleteMarketTimeOpening(request.getRecordId());

Der Rest, sprich die Zusammensetzung der Response findet genau wie beim Insert Request, im Kapitel 3.2.3.5.1 beschrieben, statt.

##### Select

Die Verarbeitung eines Select Requests stellt einen Spezialfall dar. Einerseits weil keine Parameter mitgegeben werden müssen, andererseits, weil keine Entity Konvertierung nötig ist und ein spezieller Responsetyp nötig ist.

/\*\*  
 \* Business logic for selecting all existing MarketTimeOpening records.  
 \* @return An {@link OperationServiceResponse} with the {@link Status} of the   
 \* Operation (OK, HARD\_ERROR) and a list of all MarketTimeOpening objects.  
 \*/  
**private** OperationServiceResponse selectMarketTimeOpeningsInternal() {  
 SelectMarketTimeOpeningResponse response = **new** SelectMarketTimeOpeningResponse();  
   
 List<MarketTimeOpening> queryResult = staticDataAccessor.findAllMarketTimeOpenings();  
  
 **for**(MarketTimeOpening entity : queryResult) {  
 response.getMarketTimeOpenings().add(convertEntityToSelectBO(entity));  
 }  
  
 *LOGGER*.info("Successfully selected all records from " + MarketTimeOpening.**class**);  
 response.setStatus(Status.***OK***);  
 **return** response;  
}

Die Response Klasse *SelectMarketTimeOpeningResponse*, welche durch ein XML Schema generiert wurde, beinhaltet im Gegensatz zur Basis-Response eine Liste von *MarketTimeOpening* Objekten, welche zurückgesendet werden können.

Die Resultate müssen zu einem Select Business Object konvertiert werden, was in der Methode *convertEntityToSelectBO* erledigt wird.

Einen Alternativ Flow muss hier nicht implementiert werden, denn wenn die gemappte Tabelle nicht vorhanden ist, würde Hibernate eine werfen, welche aber bereits durch das allgemeine DAO in der bereitgestellten Architektur (*JpaDao*) abgefangen wird.

Der Response müssen anschliessend der Status *OK* und sämtliche Resultate angehängt werden, bevor sie zurück gesendet werden kann.

##### Hilfsmethoden

Beim Verarbeiten der Requests können viele Funktionalitäten bei mehreren Request Typen angewendet werden. Diese werden in eigene *private* Methoden ausgelagert. Diese Hilfsmethoden greifen jedoch teilweise auch auf weiter Hilfsmethoden zu, was die Übersicht ein wenig erschwert. In diesem Kapitel werden sie vorgestellt.

###### convertGeneralBOToEntity

Die komplexeste Hilfsmethode ist jene mit dem Namen *convertGeneralBOToEntity*, welche sowohl von der Insert wie auch von der Update Verarbeitung benutzt wird. Sie übernimmt, wie der Name schon sagt, die allgemeinen Attribute im *MarketTimeOpeningGeneralBOAttributes* und speichert sie in ein neu erstelltes Entity.

Die Attribute, welche auch einen *null* Wert beinhalten können, wie *openingTime* und *closingTime* werden folgendermassen übernommen.

XMLGregorianCalendar closingTime = bo.getClosingTime();  
**if**(closingTime != **null**) {  
 setCalendarDate(closingTime);  
 entity.setClosingTime(**new** Timestamp(closingTime.toGregorianCalendar().getTimeInMillis()));  
}

Das Attribut muss nur übernommen werden, wenn sein Wert nicht *null* beträgt. Die Methode *setCalendarDate* wird in Kapitel 3.2.3.5.5.4 beschrieben.

Zum Schluss muss aus dem *XMLGregorianCalendar*, welcher aus dem *xsd:time* Datentyp von JAXB erstellt wurde, ein *Timestamp* erstellt und dem Entity hinzugefügt werden.

Die zwingend erforderlichen Methoden werden auf dem exakt selben Weg dem Entity zugefügt, ausser die Abfrage ob das Attribut nicht *null* ist, wird weggelassen, da dies bereits per XSD sichergestellt wird.

Der Fremdschlüssel *timezoneInternalId* ist das einzige Attribut, welches keine Zeit darstellen soll. Es ist sowohl im Business Object, wie auch im Entity Objekt als *BigInteger* gespeichert und muss darum nur eins zu eins übernommen werden.

###### convertEntityToSelectBO

Diese Methode wird ausschliesslich während der Verarbeitung des Select Requests ausgeführt. Sie übernimmt die Attributwerte vom Entity und erstellt daraus ein neues Business Object für die Select Response. Da mehrere *TimeStamps* im Entity verwendet werden, welche zum Datentyp *XMLGregorianCalendar* umgewandelt werden müssen, wird eine weiter Hilfsmethode, *convertTimestampToCalendar* verwendet.

###### convertTimestampToCalendar

Der Methode wird die simple Aufgabe zugetragen, einen *TimeStamp* zu einem *XMLGregorianCalendar* zu konvertieren. Dies geschieht wie im Code Snippet unten dargestellt.

GregorianCalendar cal = **new** GregorianCalendar();  
cal.setTimeInMillis(timestamp.getTime());  
**return** **new** XMLGregorianCalendarImpl(cal);

Da die Klasse *XMLGregorianCalendarImpl* von der abstrakten Klasse *XMLGregorianCalendar* erbt, kann sie im Business Object verwendet werden.

###### setCalendarDate

In Insert und Update Requests werden ausschliesslich Zeiten und keine Daten übergeben. Die Aufgabenstellung besagt jedoch, dass in der Datenbank jeweils das Datum 1.1.1900 verwendet werden muss. Die Methode *setCalendarDate* setzt dieses geforderte Datum bei der Konvertierung von Business Object zum Entity.

/\*\*  
 \* Dates in DB have always to be 01.01.1900, excluding lastUpdated.  
 \* Only times will be set in request for opening times.  
 \*/

**private** XMLGregorianCalendar setCalendarDate(XMLGregorianCalendar calendar) {  
 calendar.setDay(1);  
 calendar.setMonth(1);  
 calendar.setYear(1900);  
// No setting of seconds, because xsd:time requires specification of hh:mm:ss  
 **return** calendar;  
}

#### Validierung

Die Validierung wird in einer separaten Klasse namens *MarketTimeOpeningValidation* durchgeführt. Es werden drei *public* Methoden als Schnittstelle zu den Actions bereitgestellt. Die drei Methoden beziehen sich je auf einen Request (Insert, Update, Delete). Da der Select Request keine Parameter beinhaltet, wird auch die Überprüfung solcher unnötig.

Intern, als *private* gekennzeichnet, werden nur zwei Methoden bereitgestellt.

1. Die Validierung von *GeneralBOAttributes* wird bei Insert und Update Requests benötigt. Diese Überprüft die Richtigkeit der Zeitattribute, sowie den Fremdschlüssel.
2. Die Validierung des Primärschlüssels wird bei Update und Delete Requests benötigt. Da beim Insert der Primärschlüssel erst noch generiert wird, wird diese Überprüfung unnötig.

##### General Attributes

Bei der Validierung der generellen Attribute wird ein *MarketTimeOpeningGeneralBO* als Parameter benötigt. Es werden sämtliche Attribute dieses Business Objects validiert. Die Restrictions, welche im XSD erstellt wurden, werden zur Sicherheit nochmals überprüft. Für jeden Fehler, der gefunden wird, wird einer Liste mit Strings eine Fehlermeldung hinzugefügt. Als Beispiel die Überprüfung, ob das Attribut *homeOpening* einen *null* Wert beinhaltet:

**if**(businessObject.getHomeOpening() == **null**) {  
 errorMessages.add("Attribute HomeOpening is null.");  
}

Bei der Überprüfung des Fremdschlüssels wird zuerst überprüft ob ein Wert vorhanden ist. Falls ja, wird das dazugehörige Objekt über die Hilfsmethode im *StaticDataAccessor* gesucht. Es wird die selbe Methodik angewandt wie beim Überprüfung des Primärschlüssels in Kapitel 3.2.3.6.2.

Schlussendlich wird die Liste mit allen Fehlermeldungen der aufrufenden Methode der Action Klasse zurück gesendet.

##### ID

Beim Update und Delete Request muss der Primärschlüssel auf seine Existenz überprüft werden. Dafür wird die *findById* Methode für das *MarketOpeningTimeDao* im *StaticDataAcessor* verwendet um den dazugehörigen Record zu finden. Liefert die Suche einen *null* Wert, wird eine Fehlermeldung erstellt (unten gelb markiert).

**if**(id == **null**) {  
 errormessages.add("Attribute TimeInternalId is null.");  
} **else** {  
 **if**(staticDataAccessor.findMarketTimeOpeningById(id) == **null**) {  
 errormessages.add("MarketTimeOpening record with ID=" + id + " does not exist.");  
 }  
}

#### SoapUI Requests

SoapUI Requests werden einerseits zum Testen aller Komponenten implementiert, andererseits kann dadurch auch ein grafischer Output geliefert werden. Sie bilden also für diese IPA den Startpunkt der Applikation.

Der Request besteht grundsätzlich immer aus einem *header* und einem *operationServiceRequest*. Der *header* sieht jeweils folgendermassen aus:

<str:header>  
 <str:requestId>123</str:requestId>  
 <str:traceId>123</str:traceId>  
 <str:time>17:03:2016 07:56:01</str:time>  
 <str:username>looslir</str:username>  
</str:header>

Das einzige relevante Element für die Verarbeitung ist der *username*. Dieser wird letztendlich in die Datenbanktabelle *LAST\_UPDATED\_BY\_USER* gespeichert.

Die jeweiligen XML Requests unterscheiden sich je nach Operationen, die sie ausführen sollen. Generell muss das XML mit dem XML Schema des auszuführenden Services übereinstimmen. Als Beispiel wird hier der *UpdateMarketTimeOpening Request* aufgeführt.

<str:operationServiceRequest xsi:type="str:UpdateMarketTimeOpening" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" str:serviceId="123">  
 <str:businessObject>  
 <str:timeInternalId>49</str:timeInternalId>  
 <str:marketTimeOpening>  
 <str:timezoneInternalId>12</str:timezoneInternalId>  
 <str:homeOpening>08:00:00</str:homeOpening>  
 <str:homeClosing>19:30:00</str:homeClosing>  
 <str:openingTime>08:15:00</str:openingTime>  
 <str:closingTime>15:29:00</str:closingTime>  
 <str:execpoolSettleTime>22:20:00</str:execpoolSettleTime>  
 </str:marketTimeOpening>  
 </str:businessObject>  
</str:operationServiceRequest>

Das wichtigste neben der Struktur des Requests, ist der angegebene Request Typ (gelb markiert). Dieser muss mit dem Namen der zu instanziierenden Request Klasse übereinstimmen.

Das Resultat wird dann strukturiert gemäss XSD zurückgegeben. Im oberen Teil wird der *header* genauso aufgelistet wie er bei der Ausführung gesendet wurde. Das Resultat bzw. die *operationServiceResponse* wird je nach Ergebnis strukturiert. Die folgende Abbildung zeigt das Resultat eines korrekt ausgeführten Requests.

<ns2:operationServiceResponse>  
 <ns2:status>OK</ns2:status>  
 <ns2:recordNumber>12</ns2:recordNumber>  
 <ns2:serviceId>123</ns2:serviceId>  
</ns2:operationServiceResponse>

Eine Fehlerhafte Response beinhaltet neben dem *HARD\_ERROR* Status auch noch die Liste mit allen Fehlermeldungen.

<ns2:operationServiceResponse>  
 <ns2:status>HARD\_ERROR</ns2:status>  
  
 <ns2:errors>Failed to delete class com.csg.street.persistence.MarketTimeOpening</ns2:errors>  
  
 <ns2:errors>MarketTimeOpening record with ID=20000262 does not exist.</ns2:errors>  
  
 <ns2:serviceId>123</ns2:serviceId>  
</ns2:operationServiceResponse>

Somit können Fehlermeldungen direkt aus der XML Response gelesen werden und müssen nicht im Log der Applikation gesucht werden, wo sie ja ebenfalls geschrieben werden.

### Testing

Das Testing wird mit verschiedenen Methoden durchgeführt. Einerseits werden DAO’s und Business Logik mit dem JUnit 4 Framework getestet. Persistence Layer und Business Layer werden jeweils separat voneinander getestet. Da das Ergebnis beim Business Layer nicht durch den Persistence Layer verfälscht werden soll, wird dieser mit dem bereits im Projekt verwendeten Mockito Framework gemockt. Zudem werden Test Request von SoapUI geschickt um den Gesamtablauf zu testen.

Die Testfälle der Persistenz Schicht sollen die Funktionalität von Hibernate und Spring bei den verschiedenen Operationen testen. Ausserdem wird für jede Operation in der Businesslogik eine Validiermethode erstellt. Der wichtigste Punkt beim der Validierung der mitgegebenen Werte besteht darin, zu überprüfen ob ein angegebener Fremdschlüssel auch existiert.

#### Testprotokoll Persistence Layer

In Tabelle 5 werden ausschliesslich die Tests der DAO Methoden protokolliert. Diese werden mit JUnit Tests durchgeführt, welche gegen eine H2DB laufen gelassen werden.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Testfall | Zu testende Merkmale und Vorgehen | Erwartetes Resultat | Beobachtetes Resultat | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\cross.png or C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 1 | DAO Insert | Aufruf der insert Methode im DAO mit einem Objekt welches nur Fremdschlüssel enthält. (Alle anderen Attribute dürfen auf der DB null enthalten) | MarketTime Objekt inklusive recordId | Insert wurde geloggt.  recordId = 6 | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 2 | DAO Insert mit zu langem Wert | Aufruf der insert Methode im DAO. Das Attribut *lastUpdatedByUser* beinhaltet mehr als 30 Zeichen | PersistenceException (Test Annotation enthält Attribut expected = PersistenceException) | PersistenceException | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 4 | DAO Update | Aufruf der update Methode im DAO mit einem Objekt welches nur Fremdschlüssel enthält. (Alle anderen Attribute dürfen auf der DB null enthalten) | MarketTime Objekt mit neuen Werten. | Fremdschlüssel wurde von 69 auf 100 geändert. | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 6 | DAO Select | Aufruf der findAll Methode im DAO. | Liste mit allen MarketTime Objekten | 2 Objekte wurden dem DAO hinzugefügt. 2 Objekte wurden gefunden. | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 7 | DAO Delete | Aufruf der delete Methode im DAO mit vorhandener recordId | Rückgabewert gelöschtes Objekt | Rückgabewert gelöschtes Objekt  Primärschlüssel konnte nach Ausführung der Operation nicht mehr gefunden werden. | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 8 | DAO Delete mit falscher recordId | Aufruf der delete Methode im DAO mit inexistenter recordId | Rückgabewert null | Rückgabewert null | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |

Tabelle 5: Testprotokoll Persistence Layer

#### Testprotokoll Business Layer

Tabelle 6 beschreibt den Testverlauf bezüglich Business Layer. Die Schnittstelle zum Persistence Layer wird gemockt. Die Tests werden als JUnit Tests umgesetzt und ausgeführt.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Testfall | Zu testende Merkmale und Vorgehen | Erwartetes Resultat | Beobachtetes Resultat | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\cross.png or C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 1 | Action Insert | Aufruf von *insertMarketTimeInternal* mittels InsertMarketTime Request mit vollständigen und korrekten Parametern:  TIMEZONE\_INTERNAL\_ID=367  HOME\_OPENING=10:00:00  HOME\_CLOSING=19:00:00  OPENING\_TIME=null  CLOSING\_TIME=null  EXEXPOOL\_SETTLE\_TIME=22:30:00 | Response mit Status OK und leerer Error Liste  Inhalt des Records wird geloggt  ID des erzeugten Records wird der Response mitgegeben. | Status OK  Liste mit Errors ist leer  ID wurde erzeugt | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 2 | Action Insert ohne HOME\_CLOSING | Aufruf von *insertMarketTimeInternal* mittels InsertMarketTime Request mit folgenden Parametern:  TIMEZONE\_INTERNAL\_ID=367  HOME\_OPENING=10:00:00  HOME\_CLOSING=null  EXEXPOOL\_SETTLE\_TIME=22:30:00 | Response mit Status HARD\_ERROR und Fehler in Error Liste | Status HARD\_ERROR  Liste mit Errors hat Einträge  ID in Response = null | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 3 | Action Update | Aufruf von *updateMarketTimeInternal* mittels UpdateMarketTime Request mit vollständigen und korrekten Parametern:  RECORD\_ID=1  TIMEZONE\_INTERNAL\_ID=367  HOME\_OPENING=10:00:00  HOME\_CLOSING=19:00:00  EXEXPOOL\_SETTLE\_TIME=22:30:00 | Response mit Status OK und leerer Error Liste  Alter und neuer Inhalt des Objektes wird geloggt | Status OK  leere Error Liste  alter und neuer Inhalt wird geloggt  lastUpdatedByUser:  vorher: old testuser  nachher: unittester | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 4 | Action Update mit falschem Fremdschlüssel | Aufruf von *updateMarketTimeInternal* mittels UpdateMarketTime Request mit vollständigen und korrekten Parametern:  RECORD\_ID=1  TIMEZONE\_INTERNAL\_ID=777  HOME\_OPENING=10:00:00  HOME\_CLOSING=19:00:00  EXEXPOOL\_SETTLE\_TIME=22:30:00 | Fehlermeldung „Foreign Key does not exist”  Response mit Status HARD\_ERROR und Fehlermeldung in Error Liste | Fehlermeldung „Foreign Key does not exist“  Status HARD\_ERROR  Fehlermeldung geloggt und in Liste | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 5 | Action Select | Aufruf von selectMarketTimeInternal mittels SelectMarketTime Request (keine Parameter notwendig) | SelectMarketTimeResult mit allen MarketTime Objekten und Status OK | Alle 3 Einträge werden gefunden  Status OK | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 6 | Action Delete | Aufruf von deleteMarketTimeInternal mittels DeleteMarketTimeRequest mit vorhandener recordId | Response mit Status OK und leerer Error Liste  Der letzte Stand des gelöschten Records wird geloggt | Status OK  leere Error Liste  letzter Stand wurde gelöscht | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 7 | Action Delete mit falscher recordId | Aufruf von deleteMarketTimeInternal mittels DeleteMarketTimeRequest mit nicht vorhandener recordId | Response mit Status HARD\_ERROR und Fehlermeldung in Error Liste | Status HARD\_ERROR  Fehlermeldung „record with this ID does not exist“ | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |

Tabelle 6: Testprotokoll Business Layer

#### Testprotokoll Integrationstests

Die Integrationstests werden mittels SoapUI durchgeführt und testen das komplexe Zusammenspiel zwischen Business Layer und Persistence Layer. Die Testfälle sowie deren Ergebnisse sind in Tabelle 7 zu sehen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Testfall | Zu testende Merkmale und Vorgehen | Erwartetes Resultat | Beobachtetes Resultat | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\cross.png or C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 1 | Insert | Sendung von InsertMarketTime Request mit folgenden Parametern:  TIMEZONE\_INTERNAL\_ID=367  HOME\_OPENING=07:00:00  HOME\_CLOSING=18:00:00  OPENING\_TIME=06:30:00  CLOSING\_TIME=18:00:00  EXEXPOOL\_SETTLE\_TIME=22:30:00 | XML Response mit Status OK und keinen Fehlermeldungen | Status OK  Leere Error Liste  Eingefügtes Objekt wurde geloggt | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 2 | Update | Sendung von UpdateMarketTime Request mit folgenden Parametern:  TIME\_INTERNAL\_ID=1  TIMEZONE\_INTERNAL\_ID=367  HOME\_OPENING=05:45:00  HOME\_CLOSING=16:00:00  OPENING\_TIME=08:30:00  CLOSING\_TIME=19:00:00  EXEXPOOL\_SETTLE\_TIME=22:30:00 | XML RESPONSE mit Status OK und keinen Fehlermeldungen | Status OK  Leere Error Liste  Vorher und Nachher wurde geloggt | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 3 | Select | Sendung von SelectMarketTime Request, welcher keine Parameter braucht. | XML RESPONSE mit Status OK und keinen Fehlermeldungen | Status OK  Liste mit allen Einträgen  Leere Error Liste | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |
| 4 | Delete | Sendung von DeleteMarketTime Request mit folgenden Parametern:  RECORD\_ID=RECORD\_ID von Testfall #1 | XML RESPONSE mit Status OK und keinen Fehlermeldungen | Status OK  Leere Error Liste  letzter Stand wurde geloggt | C:\data\projects\AURARTJ-7.5.0\AURA_WebApp\web\yof2Web\webapp\images\tick.png |

Tabelle 7: Testprotokoll Integrationstest

### Integration / Wartung

Nach der IPA werden die Services auf verschiedenen Ebenen deployd. Dazu wird, wie in der Aufgabenstellung erwähnt, ein GUI zur Bearbeitung der Öffnungszeiten entwickelt. Dies erleichtert die Neuerfassung bzw. Aktualisierung der Daten erheblich.

# Schlusswort

Während zehn sehr anspruchsvollen Tagen habe ich meine IPA erfolgreich absolviert. Ich konnte sämtliche Ziele der Aufgabenstellung erfüllen, wobei nicht immer alles reibungslos verlief. Ich habe während dieser Zeit auch wieder dazugelernt. Bei diesem Projekt habe ich beispielsweise bemerkt, dass die Zeiteinteilung extrem wichtig ist. Diesen Aspekt habe ich beim Ausführen von Alltagsaspekt bisher nicht so deutlich wahrgenommen. Deshalb dachte ich zu Beginn dieser Arbeit auch, dass die Erstellung eines Zeitplans zur Farce verkommt. Im Verlauf dieser zehn Tage war ich jedoch sehr froh, dass ich mir eine solche Übersicht über die zeitlichen Aspekte zusammengestellt habe. Im Allgemeinen konnte ich meine Arbeitspakete meist sogar vor dem geplanten Termin fertigstellen, jedoch würde ich mir bei einer erneuten Durchführung mehr Zeit für die Erstellung des Zeitplans nehmen. Die rasche Umsetzung des Projekts, welche keine grossen Probleme hervorgerufen haben, hat mich zwischenzeitlich selber überrascht und stolz gemacht. Ich hoffe dieser schnelle Ablauf wird sich in zukünftigen Projekten wiederholen.

Die Anwendung der Technologien stellte für mich keine grössere Hürde dar, da ich diese bereits im Arbeitsalltag mehrfach eingesetzt habe und daher schon damit vertraut war. Grössere Schwierigkeiten dagegen bereitete mir die Dokumentation, da ich einfach nicht gerne viel schreibe. Ich musste mich stets wieder neu motivieren und aufpassen, dass ich nicht den Fokus verliere.

Im Allgemeinen hat mir die Realisation dieses Projektes trotz der Stolpersteine und den weniger spassigen Teilen eine grosse Freude bereitet und ich bin stolz mein erarbeitetes Wissen in einem Projekt einzubringen, welches anschliessend auch produktiv verwendet wird. Ich freue mich, dass in Zukunft Anpassungen zu Marktöffnungszeiten nicht mehr in einem SQL gescriptet werden müssen, sondern ganz einfach meine Webservices aufgerufen werden, später sogar mit GUI.

Während der Realisation dieses Projektes habe ich mich sehr angestrengt, die optimale Leistung zu erbringen und ich hoffe meine Bemühungen werden belohnt.

# Glossar & Verzeichnisse

## Glossar

Das Glossar beschreibt Abkürzungen und Fachausdrücke, welche in diesem Dokument verwendet werden. Sämtliche Internetquellen basieren auf dem Stand des 23. März 2016.

|  |  |  |
| --- | --- | --- |
| **Begriff** | **Quelle** | **Erklärung** |
| CRUD | <https://de.wikipedia.org/wiki/CRUD> | Create, Read, Update, Delete – grundlegende Datenbankoperationen |
| DAO | <http://tutorials.jenkov.com/java-persistence/dao-design-pattern.html> | Data Access Object: Abstrahiert den Zugriff auf die Persistenz Objekte |
| HQL | https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html | Hibernate Query Language |
| JAXB | https://jaxb.java.net/ | Java Architecture for XML Binding |
| log4j | <http://logging.apache.org/log4j/2.x/> | Logging Tool |
| SVN | <https://subversion.apache.org/> | Subversion: Repository um Daten zu sichern und versionieren. |

## Abbildungsverzeichnis

[Abbildung 1: Zeitplan 11](#_Toc446485708)

[Abbildung 2: Wasserfallmodell 23](#_Toc446485709)

[Abbildung 3: Benötigte Sub-Projekte aus dem Gesamtprojekt 24](#_Toc446485710)

[Abbildung 4: ERD involvierte Tabellen 25](#_Toc446485711)

[Abbildung 5: Prozessablauf 27](#_Toc446485712)

## Tabellenverzeichnis

[Tabelle 1: ref\_MARKET\_TIME Spaltenbeschreibung 26](#_Toc446485713)

[Tabelle 2: Entity Annotations 29](#_Toc446485714)

[Tabelle 3: Konvertierung DB - Java 30](#_Toc446485715)

[Tabelle 4: @Column Attribute 30](#_Toc446485716)

[Tabelle 5: Testprotokoll Persistence Layer 54](#_Toc446485717)

[Tabelle 6: Testprotokoll Business Layer 56](#_Toc446485718)

[Tabelle 7: Testprotokoll Integrationstest 58](#_Toc446485719)

## Quellenverzeichnis

|  |  |
| --- | --- |
| **Name** | **URL** |
| Hibernate Documentation | <http://hibernate.org/orm/documentation/5.1/> |
| Java 6 API | <http://docs.oracle.com/javase/6/docs/api/> |
| JUnit 4 | <http://junit.sourceforge.net/javadoc/> |
| Sybase Infocenter | <http://infocenter.sybase.com/archive/index.jsp?topic=/com.sybase.help.ase_15.0.tables/html/tables/tables22.htm> |
| W3Schools | <http://www.w3schools.com/xml/xml_schema.asp> |

# Anhang

## SoapUI Requests

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:str="http://www.credit-suisse.com/streetside">

<soapenv:Header/>

<soapenv:Body>

<str:operationRequest>

<str:header>

<str:requestId>123</str:requestId>

<str:traceId>123</str:traceId>

<str:time>17:03:2016 07:56:01</str:time>

<str:username>looslir</str:username>

</str:header>

<str:operationServiceRequest xsi:type="str:InsertMarketTimeOpening" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" str:serviceId="123">

<str:businessObject>

<str:marketTimeOpening>

<str:timezoneInternalId>367</str:timezoneInternalId>

<str:homeOpening>07:00:00</str:homeOpening>

<str:homeClosing>18:00:00</str:homeClosing>

<str:openingTime>06:30:00</str:openingTime>

<str:closingTime>18:00:00</str:closingTime>

<str:execpoolSettleTime>22:30:00</str:execpoolSettleTime>

</str:marketTimeOpening>

</str:businessObject>

</str:operationServiceRequest>

</str:operationRequest>

</soapenv:Body>

</soapenv:Envelope>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:str="http://www.credit-suisse.com/streetside">

<soapenv:Header/>

<soapenv:Body>

<str:operationRequest>

<str:header>

<str:requestId>123</str:requestId>

<str:traceId>123</str:traceId>

<str:time>10:03:2016 07:56:01</str:time>

<str:username>looslir</str:username>

</str:header>

<str:operationServiceRequest xsi:type="str:UpdateMarketTimeOpening" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" str:serviceId="123">

<str:businessObject>

<str:timeInternalId>1</str:timeInternalId>

<str:marketTimeOpening>

<str:timezoneInternalId>367</str:timezoneInternalId>

<str:homeOpening>05:45:00</str:homeOpening>

<str:homeClosing>16:00:00</str:homeClosing>

<str:openingTime>08:30:00</str:openingTime>

<str:closingTime>19:00:00</str:closingTime>

<str:execpoolSettleTime>22:30:00</str:execpoolSettleTime>

</str:marketTimeOpening>

</str:businessObject>

</str:operationServiceRequest>

</str:operationRequest>

</soapenv:Body>

</soapenv:Envelope>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:str="http://www.credit-suisse.com/streetside">

<soapenv:Header/>

<soapenv:Body>

<str:operationRequest>

<str:header>

<str:requestId>123</str:requestId>

<str:traceId>123</str:traceId>

<str:time>10:03:2016 07:56:01</str:time>

<str:username>looslir</str:username>

</str:header>

<str:operationServiceRequest xsi:type="str:SelectMarketTimeOpening" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" str:serviceId="123">

</str:operationServiceRequest>

</str:operationRequest>

</soapenv:Body>

</soapenv:Envelope>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:str="http://www.credit-suisse.com/streetside">

<soapenv:Header/>

<soapenv:Body>

<str:operationRequest>

<str:header>

<str:requestId>123</str:requestId>

<str:traceId>123</str:traceId>

<str:time>10:03:2016 07:56:01</str:time>

<str:username>looslir</str:username>

</str:header>

<str:operationServiceRequest xsi:type="str:DeleteMarketTimeOpening" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" str:serviceId="123">

<str:recordId>20000269</str:recordId>

</str:operationServiceRequest>

</str:operationRequest>

</soapenv:Body>

</soapenv:Envelope>

## StreetMonitorOperationsServices.xsd

<!-- Request for selecting all MarketTimeOpening objects from the database. -->

<xsd:complexType name="SelectMarketTimeOpening">

<xsd:complexContent>

<xsd:extension base="str:OperationServiceRequest">

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Request for inserting a new MarketTimeOpening object into the database. -->

<xsd:complexType name="InsertMarketTimeOpening">

<xsd:complexContent>

<xsd:extension base="str:OperationServiceRequest">

<xsd:sequence>

<xsd:element name="businessObject" type="str:MarketTimeOpeningInsertBO" minOccurs="1" maxOccurs="1" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Request for updating an existing MarketTimeOpening object on the database. -->

<xsd:complexType name="UpdateMarketTimeOpening">

<xsd:complexContent>

<xsd:extension base="str:OperationServiceRequest">

<xsd:sequence>

<xsd:element name="businessObject" type="str:MarketTimeOpeningUpdateBO" minOccurs="1" maxOccurs="1" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<!-- Request for deleting an existing MarketTimeOpening object from the database. -->

<xsd:complexType name="DeleteMarketTimeOpening">

<xsd:complexContent>

<xsd:extension base="str:OperationServiceRequest">

<xsd:sequence>

<xsd:element name="recordId" type="str:IdRestriction" minOccurs="1" maxOccurs="1" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

## StreetMonitorOperationsBusinessObjects.xsd

<!-- Business Object used by the InsertMarketTimeOpening Request. Business object contains all attributes which can be

manually filled in the request.-->

<xsd:complexType name="MarketTimeOpeningInsertBO">

<xsd:sequence>

<xsd:element name="marketTimeOpening" type="str:MarketTimeOpeningGeneralBO" minOccurs="1" maxOccurs="1"/>

</xsd:sequence>

</xsd:complexType>

<!-- Business Object used by the UpdateMarketTimeOpening Request. Business object contains all attributes which can be

manually filled in the request and a timeInternalId which is the record id in the database.-->

<xsd:complexType name="MarketTimeOpeningUpdateBO">

<xsd:sequence>

<xsd:element name="timeInternalId" type="str:IdRestriction" minOccurs="1" maxOccurs="1"/>

<xsd:element name="marketTimeOpening" type="str:MarketTimeOpeningGeneralBO" minOccurs="1" maxOccurs="1"/>

</xsd:sequence>

</xsd:complexType>

<!-- Business Object used by the SelectMarketTimeOpening Request. It's needed for the Response XML Business object contains all attributes which can be

manually filled in the request and a timeInternalId which is the record id in the database.-->

<xsd:complexType name="MarketTimeOpeningSelectBO">

<xsd:sequence>

<xsd:element name="timeInternalId" type="str:IdRestriction" minOccurs="1" maxOccurs="1"/>

<xsd:group ref="str:MarketTimeOpeningGeneralBOAttributes"/>

<xsd:element name="lastUpdatedByUser" type="xsd:string" minOccurs="1" maxOccurs="1"/>

<xsd:element name="lastUpdate" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>

</xsd:sequence>

</xsd:complexType>

<!-- Business Object used by both, the UpdateMarketTimeOpening and InsertMarketTimeOpening Request, as well as SelectMarketTimeOpening Response.

Business object contains all attributes which can be manually filled in an insert or update request. -->

<xsd:complexType name="MarketTimeOpeningGeneralBO">

<xsd:sequence>

<xsd:group ref="str:MarketTimeOpeningGeneralBOAttributes"/>

</xsd:sequence>

</xsd:complexType>

<xsd:group name="MarketTimeOpeningGeneralBOAttributes">

<xsd:sequence>

<xsd:element name="timezoneInternalId" type="str:IdRestriction" minOccurs="1" maxOccurs="1" nillable="false"/>

<xsd:element name="homeOpening" type="xsd:time" minOccurs="1" maxOccurs="1" nillable="false"/>

<xsd:element name="homeClosing" type="xsd:time" minOccurs="1" maxOccurs="1" nillable="false"/>

<xsd:element name="openingTime" type="xsd:time" minOccurs="0" maxOccurs="1" nillable="false"/>

<xsd:element name="closingTime" type="xsd:time" minOccurs="0" maxOccurs="1" nillable="false"/>

<xsd:element name="execpoolSettleTime" type="xsd:time" minOccurs="1" maxOccurs="1" nillable="false"/>

</xsd:sequence>

</xsd:group>

<!-- Per default ID Attributes have a maximum length of 10 digits. -->

<xsd:simpleType name="IdRestriction">

<xsd:restriction base="xsd:positiveInteger">

<xsd:totalDigits value="10"/>

</xsd:restriction>

</xsd:simpleType>

## StreetMonitorOperationsResult.xsd

<!-- This object is used as definition for the MarketTimeOpening Select response -->

<xsd:complexType name="SelectMarketTimeOpeningResponse">

<xsd:complexContent>

<xsd:extension base="str:OperationServiceResponse">

<xsd:sequence>

<xsd:element name="marketTimeOpenings" type="str:MarketTimeOpeningSelectBO" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

## MonitorServiceEndpoint.java

else if(command instanceof SelectMarketTimeOpening) {

result = new MarketTimeOpeningActions(this.backendInteraction).selectMarketTimeOpening(header, (SelectMarketTimeOpening) command);

} else if(command instanceof InsertMarketTimeOpening) {

result = new MarketTimeOpeningActions(this.backendInteraction).insertMarketTimeOpening(header, (InsertMarketTimeOpening) command);

} else if(command instanceof UpdateMarketTimeOpening) {

result = new MarketTimeOpeningActions(this.backendInteraction).updateMarketTimeOpening(header, (UpdateMarketTimeOpening) command);

} else if(command instanceof DeleteMarketTimeOpening) {

result = new MarketTimeOpeningActions(this.backendInteraction).deleteMarketTimeOpening(header, (DeleteMarketTimeOpening) command);

}

## MarketTimeOpeningActions.java

package com.csg.otex.guibe.impl.commands;

import java.sql.Timestamp;

import java.util.ArrayList;

import java.util.Date;

import java.util.GregorianCalendar;

import java.util.List;

import javax.xml.datatype.XMLGregorianCalendar;

import org.apache.log4j.Logger;

import com.csg.otex.guibe.api.commands.ops.DeleteMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.InsertMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningGeneralBO;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningSelectBO;

import com.csg.otex.guibe.api.commands.ops.OperationHeader;

import com.csg.otex.guibe.api.commands.ops.OperationServiceResponse;

import com.csg.otex.guibe.api.commands.ops.SelectMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.SelectMarketTimeOpeningResponse;

import com.csg.otex.guibe.api.commands.ops.Status;

import com.csg.otex.guibe.api.commands.ops.UpdateMarketTimeOpening;

import com.csg.otex.guibe.interop.BackendFacade;

import com.csg.otex.guibe.interop.StaticDataAccessor;

import com.csg.otex.guibe.validation.impl.MarketTimeOpeningValidation;

import com.csg.street.persistence.MarketTimeOpening;

import com.sun.org.apache.xerces.internal.jaxp.datatype.XMLGregorianCalendarImpl;

public class MarketTimeOpeningActions extends Actions {

private static Logger LOGGER = Logger.getLogger(MarketTimeOpeningActions.class);

private StaticDataAccessor staticDataAccessor;

private MarketTimeOpeningValidation validation;

private List<String> errorMessages = new ArrayList<String>();

public MarketTimeOpeningActions(BackendFacade backendInteraction) {

super(backendInteraction);

this.staticDataAccessor = backendInteraction.getStaticDataAccessor();

this.validation = new MarketTimeOpeningValidation(staticDataAccessor);

}

public OperationServiceResponse insertMarketTimeOpening(OperationHeader header, InsertMarketTimeOpening request) {

errorMessages.clear();

return insertMarketTimeOpeningInternal(header, request);

}

public OperationServiceResponse updateMarketTimeOpening(OperationHeader header, UpdateMarketTimeOpening request) {

errorMessages.clear();

return updateMarketTimeOpeningInternal(header, request);

}

public OperationServiceResponse deleteMarketTimeOpening(OperationHeader header, DeleteMarketTimeOpening request) {

errorMessages.clear();

return deleteMarketTimeOpeningInternal(header, request);

}

public OperationServiceResponse selectMarketTimeOpening(OperationHeader header, SelectMarketTimeOpening request) {

errorMessages.clear();

return selectMarketTimeOpeningsInternal();

}

/\*\*

\* Business logic for inserting a new MarketTimeOpening record.

\* Request will be validated and, if valid, inserted.

\* The inserted entity will be logged.

\* @param header Provides general infos to the request, like username of the sender.

\* @param request Provides data for the insert Query.

\* @return An {@link OperationServiceResponse} with the {@link Status} of the Operation (OK, HARD\_ERROR) and all error messages.

\*/

private OperationServiceResponse insertMarketTimeOpeningInternal(OperationHeader header, InsertMarketTimeOpening request) {

OperationServiceResponse response = new OperationServiceResponse();

errorMessages = validation.validateInsertRequest(request);

if(errorMessages.size() == 0) { // list is empty if validation is successfull

MarketTimeOpening entity = convertInsertRequestToEntity(header, request); // convert request values to insertable entity

MarketTimeOpening execResult = staticDataAccessor.insertMarketTimeOpening(entity);

if(execResult != null) { // if insert fails, null will be returned

response.setStatus(Status.OK);

response.setRecordNumber(execResult.getTimeInternalId());

LOGGER.info("Inserted: " + execResult.toString());

return response;

} else {

errorMessages.add(MarketTimeOpening.class.toString() + " object could not be inserted. Check log. " + entity.toString());

}

}

response.getErrors().add("Failed to insert " + MarketTimeOpening.class.toString());

for(String em : errorMessages) {

response.getErrors().add(em);

LOGGER.info(em);

}

response.setStatus(Status.HARD\_ERROR);

return response;

}

/\*\*

\* Business logic for updating an existing MarketTimeOpening record.

\* Request will be validated and, if valid, the record will be updated.

\* The updated entity will be logged in the states before and after the update.

\* @param header Provides general infos to the request, like username of the sender.

\* @param request Provides data for the update Query.

\* @return An {@link OperationServiceResponse} with the {@link Status} of the Operation (OK, HARD\_ERROR) and all error messages.

\*/

private OperationServiceResponse updateMarketTimeOpeningInternal(OperationHeader header, UpdateMarketTimeOpening request) {

OperationServiceResponse response = new OperationServiceResponse();

errorMessages = validation.validateUpdateRequest(request);

if(errorMessages.size() == 0) {

MarketTimeOpening oldEntity = staticDataAccessor.findMarketTimeOpeningById(request.getBusinessObject().getTimeInternalId());

MarketTimeOpening newEntity = convertUpdateRequestToEntity(header, request);

MarketTimeOpening execResult = staticDataAccessor.updateMarketTimeOpening(newEntity);

if(execResult != null) { // if update fails, null will be returned

response.setStatus(Status.OK);

response.setRecordNumber(execResult.getTimeInternalId());

LOGGER.info("Updated: " + MarketTimeOpening.class.toString());

LOGGER.info(oldEntity.toString()); // log before update

LOGGER.info(execResult.toString()); // log after update

return response;

} else {

errorMessages.add(MarketTimeOpening.class.toString() + " object could not be updated. Check log. " + newEntity.toString());

}

}

response.getErrors().add("Failed to update " + MarketTimeOpening.class.toString());

for(String em : errorMessages) {

response.getErrors().add(em);

LOGGER.info(em);

}

response.setStatus(Status.HARD\_ERROR);

return response;

}

/\*\*

\* Business logic for deleting an existing MarketTimeOpening record.

\* Request will be validated and, if valid, the record will be deleted.

\* The deleted entity will be logged in the state before the deletion.

\* @param header Provides general infos to the request, like username of the sender.

\* @param request Provides recordId for the delete Query.

\* @return An {@link OperationServiceResponse} with the {@link Status} of the Operation (OK, HARD\_ERROR) and all error messages.

\*/

private OperationServiceResponse deleteMarketTimeOpeningInternal(OperationHeader header, DeleteMarketTimeOpening request) {

OperationServiceResponse response = new OperationServiceResponse();

errorMessages = validation.validateDeleteRequest(request);

if(errorMessages.size() == 0) {

MarketTimeOpening execResult = staticDataAccessor.deleteMarketTimeOpening(request.getRecordId());

if(execResult != null) { // if delete fails, null will be returned

response.setStatus(Status.OK);

response.setRecordNumber(execResult.getTimeInternalId());

LOGGER.info("Deleted: " + execResult.toString());

return response;

} else {

errorMessages.add(MarketTimeOpening.class.toString() + " object could not be deleted. ID=" + request.getRecordId());

}

}

response.getErrors().add("Failed to delete " + MarketTimeOpening.class.toString());

for(String em : errorMessages) {

response.getErrors().add(em);

LOGGER.info(em);

}

response.setStatus(Status.HARD\_ERROR);

return response;

}

/\*\*

\* Business logic for selecting all existing MarketTimeOpening records.

\* @return An {@link OperationServiceResponse} with the {@link Status} of the Operation (OK, HARD\_ERROR) and a list of all MarketTimeOpening objects.

\*/

private OperationServiceResponse selectMarketTimeOpeningsInternal() {

SelectMarketTimeOpeningResponse response = new SelectMarketTimeOpeningResponse();

List<MarketTimeOpening> queryResult = staticDataAccessor.findAllMarketTimeOpenings();

for(MarketTimeOpening entity : queryResult) {

response.getMarketTimeOpenings().add(convertEntityToSelectBO(entity));

}

LOGGER.info("Successfully selected all records from " + MarketTimeOpening.class);

response.setStatus(Status.OK);

return response;

}

/\*\*

\* Converts the business object in the insert request to an entity object.

\*/

private MarketTimeOpening convertInsertRequestToEntity(OperationHeader header, InsertMarketTimeOpening request) {

MarketTimeOpening entity = convertGeneralBOToEntity(request.getBusinessObject().getMarketTimeOpening());

entity.setLastUpdate(new Timestamp(new Date().getTime()));

entity.setLastUpdatedByUser(header.getUsername());

return entity;

}

/\*\*

\* Converts the business object in the update request to an entity object.

\*/

private MarketTimeOpening convertUpdateRequestToEntity(OperationHeader header, UpdateMarketTimeOpening request) {

MarketTimeOpening entity = convertGeneralBOToEntity(request.getBusinessObject().getMarketTimeOpening());

entity.setLastUpdate(new Timestamp(new Date().getTime()));

entity.setLastUpdatedByUser(header.getUsername());

entity.setTimeInternalId(request.getBusinessObject().getTimeInternalId());

return entity;

}

/\*\*

\* Converts a general business object to an entity object.

\*/

private MarketTimeOpening convertGeneralBOToEntity(MarketTimeOpeningGeneralBO bo) {

MarketTimeOpening entity = new MarketTimeOpening();

XMLGregorianCalendar closingTime = bo.getClosingTime();

if(closingTime != null) {

setCalendarDate(closingTime);

entity.setClosingTime(new Timestamp(closingTime.toGregorianCalendar().getTimeInMillis()));

}

XMLGregorianCalendar openingTime = bo.getOpeningTime();

if(openingTime != null) {

setCalendarDate(openingTime);

entity.setOpeningTime(new Timestamp(openingTime.toGregorianCalendar().getTimeInMillis()));

}

XMLGregorianCalendar homeOpening = bo.getHomeOpening();

setCalendarDate(homeOpening);

XMLGregorianCalendar homeClosing = bo.getHomeClosing();

setCalendarDate(homeClosing);

XMLGregorianCalendar execpoolSettleTime = bo.getExecpoolSettleTime();

setCalendarDate(execpoolSettleTime);

entity.setExecpoolSettleTime(new Timestamp(execpoolSettleTime.toGregorianCalendar().getTimeInMillis()));

entity.setHomeClosing(new Timestamp(homeClosing.toGregorianCalendar().getTimeInMillis()));

entity.setHomeOpening(new Timestamp(homeOpening.toGregorianCalendar().getTimeInMillis()));

entity.setTimezoneInternalId(bo.getTimezoneInternalId());

return entity;

}

/\*\*

\* For returning a select business object, entity has to be convert through this method.

\*/

private MarketTimeOpeningSelectBO convertEntityToSelectBO(MarketTimeOpening entity) {

MarketTimeOpeningSelectBO bo = new MarketTimeOpeningSelectBO();

bo.setClosingTime(convertTimestampToCalendar(entity.getClosingTime()));

bo.setExecpoolSettleTime(convertTimestampToCalendar(entity.getExecpoolSettleTime()));

bo.setHomeClosing(convertTimestampToCalendar(entity.getHomeClosing()));

bo.setHomeOpening(convertTimestampToCalendar(entity.getHomeOpening()));

bo.setLastUpdate(convertTimestampToCalendar(entity.getLastUpdate()));

bo.setLastUpdatedByUser(entity.getLastUpdatedByUser());

bo.setOpeningTime(convertTimestampToCalendar(entity.getOpeningTime()));

bo.setTimeInternalId(entity.getTimeInternalId());

bo.setTimezoneInternalId(entity.getTimezoneInternalId());

return bo;

}

/\*\*

\* Converts TimeStamp to XMLGregorianCalendar object, which is needed in response.

\*/

private XMLGregorianCalendar convertTimestampToCalendar(Timestamp timestamp) {

if(timestamp == null) {

return null;

}

GregorianCalendar cal = new GregorianCalendar();

cal.setTimeInMillis(timestamp.getTime());

return new XMLGregorianCalendarImpl(cal);

}

/\*\*

\* Dates in DB have always to be 01.01.1900, excluding lastUpdated.

\* Only times will be set in request for opening times.

\*/

private XMLGregorianCalendar setCalendarDate(XMLGregorianCalendar calendar) {

calendar.setDay(1);

calendar.setMonth(1);

calendar.setYear(1900);

// No setting of seconds, because xsd:time requires specification of hh:mm:ss

return calendar;

}

}

## MarketTimeOpeningValidation.java

package com.csg.otex.guibe.validation.impl;

import java.math.BigInteger;

import java.util.ArrayList;

import java.util.List;

import javax.xml.datatype.DatatypeConstants;

import com.csg.otex.guibe.api.commands.ops.DeleteMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.InsertMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningGeneralBO;

import com.csg.otex.guibe.api.commands.ops.UpdateMarketTimeOpening;

import com.csg.otex.guibe.interop.StaticDataAccessor;

public class MarketTimeOpeningValidation {

private StaticDataAccessor staticDataAccessor;

public MarketTimeOpeningValidation(StaticDataAccessor staticDataAccessor) {

this.staticDataAccessor = staticDataAccessor;

}

public List<String> validateInsertRequest(InsertMarketTimeOpening request) {

return validateGeneralAttributes(request.getBusinessObject().getMarketTimeOpening());

}

public List<String> validateUpdateRequest(UpdateMarketTimeOpening request) {

List<String> errorMessages = new ArrayList<String>();

errorMessages.addAll(validateGeneralAttributes(request.getBusinessObject().getMarketTimeOpening()));

errorMessages.addAll(validateId(request.getBusinessObject().getTimeInternalId()));

return errorMessages;

}

public List<String> validateDeleteRequest(DeleteMarketTimeOpening request) {

return validateId(request.getRecordId());

}

private List<String> validateGeneralAttributes(MarketTimeOpeningGeneralBO businessObject) {

List<String> errorMessages = new ArrayList<String>();

if(businessObject.getHomeOpening() == null) {

errorMessages.add("Attribute HomeOpening is null.");

} else {

if(businessObject.getHomeOpening().getHour() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Hour defined in HomeOpening. Value=" + businessObject.getHomeOpening().getHour());

}

if(businessObject.getHomeOpening().getMinute() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Minute defined in HomeOpening. Value=" + businessObject.getHomeOpening().getMinute());

}

}

if(businessObject.getHomeClosing() == null) {

errorMessages.add("Attribute HomeClosing is null.");

} else {

if(businessObject.getHomeClosing().getHour() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Hour defined in HomeClosing. Value=" + businessObject.getHomeClosing().getHour());

}

if(businessObject.getHomeClosing().getMinute() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Minute value defined in HomeClosing. Value=" + businessObject.getHomeClosing().getMinute());

}

}

if(businessObject.getExecpoolSettleTime() == null) {

errorMessages.add("Attribute ExecPoolSettleTime is null.");

} else {

if(businessObject.getExecpoolSettleTime().getHour() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Hour defined in ExecPoolSettleTime. Value=" + businessObject.getExecpoolSettleTime().getHour());

}

if(businessObject.getExecpoolSettleTime().getMinute() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Minute defined in ExecPoolSettleTime. Value=" + businessObject.getExecpoolSettleTime().getMinute());

}

}

if(businessObject.getOpeningTime() != null) {

if(businessObject.getOpeningTime().getHour() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Hour defined in OpeningTime. Value=" + businessObject.getOpeningTime().getHour());

}

if(businessObject.getOpeningTime().getMinute() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Minute defined in OpeningTime. Value=" + businessObject.getOpeningTime().getMinute());

}

}

if(businessObject.getClosingTime() != null) {

if(businessObject.getClosingTime().getHour() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Hour defined in ClosingTime. Value=" + businessObject.getClosingTime().getHour());

}

if(businessObject.getClosingTime().getMinute() == DatatypeConstants.FIELD\_UNDEFINED) {

errorMessages.add("No Minute defined in ClosingTime. Value=" + businessObject.getClosingTime().getMinute());

}

}

if(businessObject.getTimezoneInternalId() == null) {

errorMessages.add("Attribute TimezoneInternalId is null.");

} else {

if(staticDataAccessor.findTimezoneById(businessObject.getTimezoneInternalId()) == null) {

errorMessages.add("The given value for the Foreign Key TimeZoneInternalId doesn't exist. Value=" + businessObject.getTimezoneInternalId());

}

}

return errorMessages;

}

private List<String> validateId(BigInteger id) {

List<String> errormessages = new ArrayList<String>();

if(id == null) {

errormessages.add("Attribute TimeInternalId is null.");

} else {

if(staticDataAccessor.findMarketTimeOpeningById(id) == null) {

errormessages.add("MarketTimeOpening record with ID=" + id + " does not exist.");

}

}

return errormessages;

}

}

## StaticDataAccessor.java

/\*

\* This StaticDataAccessor Class represents the interface between business layer and persistence layer.

\* Every call on the DAO class should be done in this class.

\*/

public List<MarketTimeOpening> findAllMarketTimeOpenings() {

return marketTimeOpeningDao.findAll();

}

public MarketTimeOpening findMarketTimeOpeningById(BigInteger id) {

return marketTimeOpeningDao.findById(id);

}

public MarketTimeOpening insertMarketTimeOpening(MarketTimeOpening entity) {

return marketTimeOpeningDao.insert(entity);

}

public MarketTimeOpening updateMarketTimeOpening(MarketTimeOpening entity) {

return marketTimeOpeningDao.update(entity);

}

/\*\*

\* By calling this method the record with the same primary key as the passed id will be deleted.

\* If the operation was successful the entity before it got deleted will be returned. Else the method returns null.

\*/

public MarketTimeOpening deleteMarketTimeOpening(BigInteger id) {

return marketTimeOpeningDao.delete(id);

}

public Timezone findTimezoneById(BigInteger id) {

return timezoneDao.findById(id);

}

## MarketTimeOpeningDao.java

package com.csg.street.dao;

import java.math.BigInteger;

import java.util.List;

import javax.persistence.EntityManager;

import org.apache.log4j.Logger;

import com.csg.street.dao.generic.JpaDao;

import com.csg.street.persistence.MarketTimeOpening;

public class MarketTimeOpeningDao extends JpaDao<BigInteger, MarketTimeOpening> {

private static final Logger LOGGER = Logger.getLogger(MarketTimeOpeningDao.class);

public MarketTimeOpeningDao() {

}

public MarketTimeOpeningDao(EntityManager entityManager) {

super(entityManager);

}

/\*\*

\* Select all records from entity.

\*/

public List<MarketTimeOpening> findAll() {

return super.findManyByQuery(getEntityClass().getSimpleName() + ".findAll");

}

/\*\*

\* Selects a single record from entity.

\*/

public MarketTimeOpening findById(BigInteger id) {

return super.findOneByQuery(getEntityClass().getSimpleName() + ".findById", "timeInternalId", id);

}

/\*\*

\* Insert an entity object.

\* @param entity to insert

\* @return the inserted entity, inclusive a generated primary key

\*/

public MarketTimeOpening insert(MarketTimeOpening entity) {

if(entity == null) { // check to avoid exceptions in hibernate

return null;

}

entity = merge(entity);

logEntity("Inserted:", entity);

return entity;

}

/\*\*

\* Update an entity object.

\* @param entity to update

\* @return the updated entity

\*/

public MarketTimeOpening update(MarketTimeOpening entity) {

if(entity == null) {

return null;

}

entity = merge(entity);

logEntity("Updated:", entity);

return entity;

}

/\*\*

\* Removes an entity from the database.

\* @param id the primary key of the record to be removed.

\* @return the entity object before it was removed or null if remove failed.

\*/

public MarketTimeOpening delete(BigInteger id) {

MarketTimeOpening entity = findById(id);

if(entity == null) {

return null;

}

remove(entity);

logEntity("Removed:", entity);

return entity;

}

}

## TimezoneDao.java

package com.csg.street.dao;

import java.math.BigInteger;

import java.util.List;

import javax.persistence.EntityManager;

import com.csg.street.dao.generic.JpaDao;

import com.csg.street.persistence.Timezone;

public class TimezoneDao extends JpaDao<BigInteger, Timezone>{

public TimezoneDao() {

}

public TimezoneDao(EntityManager entityManager) {

super(entityManager);

}

public List<Timezone> findAll() {

return super.findManyByQuery(getEntityClass().getSimpleName() + ".findAll");

}

public Timezone findById(BigInteger id) {

Timezone result = super.findOneByQuery(getEntityClass().getSimpleName() + ".findById", "timezoneInternalId", id);

return result;

}

}

## MarketTimeOpening.java

package com.csg.street.persistence;

import java.io.Serializable;

import java.math.BigInteger;

import java.sql.Timestamp;

import javax.persistence.Access;

import javax.persistence.AccessType;

import javax.persistence.Cacheable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import com.csg.street.model.PersistenceEntity;

@NamedQueries({

@NamedQuery(name="MarketTimeOpening.findAll", query="SELECT e FROM MarketTimeOpening e"),

@NamedQuery(name="MarketTimeOpening.findById", query="SELECT e FROM MarketTimeOpening e WHERE e.timeInternalId = :timeInternalId")

})

@Entity

@Table(name="ref\_MARKET\_TIME")

@Cacheable(true)

@Cache(usage=CacheConcurrencyStrategy.READ\_WRITE)

@Access(AccessType.FIELD)

public class MarketTimeOpening implements PersistenceEntity<BigInteger>, Serializable {

public static final long serialVersionUID = 1;

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name="TIME\_INTERNAL\_ID", unique=true, nullable=false, length=6, insertable=true, updatable=true)

private BigInteger timeInternalId;

@Column(name="TIMEZONE\_INTERNAL\_ID", unique=false, nullable=true, length=6, insertable=true, updatable=true)

private BigInteger timezoneInternalId;

@Column(name="HOME\_OPENING", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp homeOpening;

@Column(name="HOME\_CLOSING", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp homeClosing;

@Column(name="OPENING\_TIME", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp openingTime;

@Column(name="CLOSING\_TIME", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp closingTime;

@Column(name="LAST\_UPDATED\_BY\_USER", unique=false, nullable=true, length=30, insertable=true, updatable=true)

private String lastUpdatedByUser;

@Column(name="LAST\_UPDATE", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp lastUpdate;

@Column(name="EXECPOOL\_SETTLE\_TIME", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp execpoolSettleTime;

public BigInteger getTimeInternalId() {

return timeInternalId;

}

public void setTimeInternalId(BigInteger timeInternalId) {

this.timeInternalId = timeInternalId;

}

public BigInteger getTimezoneInternalId() {

return timezoneInternalId;

}

public void setTimezoneInternalId(BigInteger timezoneInternalId) {

this.timezoneInternalId = timezoneInternalId;

}

public Timestamp getHomeOpening() {

return homeOpening;

}

public void setHomeOpening(Timestamp homeOpening) {

this.homeOpening = homeOpening;

}

public Timestamp getHomeClosing() {

return homeClosing;

}

public void setHomeClosing(Timestamp homeClosing) {

this.homeClosing = homeClosing;

}

public Timestamp getOpeningTime() {

return openingTime;

}

public void setOpeningTime(Timestamp openingTime) {

this.openingTime = openingTime;

}

public Timestamp getClosingTime() {

return closingTime;

}

public void setClosingTime(Timestamp closingTime) {

this.closingTime = closingTime;

}

public String getLastUpdatedByUser() {

return lastUpdatedByUser;

}

public void setLastUpdatedByUser(String lastUpdatedByUser) {

this.lastUpdatedByUser = lastUpdatedByUser;

}

public Timestamp getLastUpdate() {

return lastUpdate;

}

public void setLastUpdate(Timestamp lastUpdate) {

this.lastUpdate = lastUpdate;

}

public Timestamp getExecpoolSettleTime() {

return execpoolSettleTime;

}

public void setExecpoolSettleTime(Timestamp execpoolSettleTime) {

this.execpoolSettleTime = execpoolSettleTime;

}

@Override

public BigInteger getId() {

return getTimeInternalId();

}

@Override

public String toString() {

String buffer = "MarketTimeOpening : {";

buffer += "timeInternalId: " + getId() + ", ";

buffer += "timezoneInternalId: " + getTimezoneInternalId() + ", ";

buffer += "homeOpening: " + getHomeOpening() + ", ";

buffer += "homeClosing: " + getHomeClosing() + ", ";

buffer += "openingTime: " + getOpeningTime() + ", ";

buffer += "closingTime: " + getClosingTime() + ", ";

buffer += "lastUpdatedByUser: " + getLastUpdatedByUser() + ", ";

buffer += "lastUpdate: " + getLastUpdate() + ", ";

buffer += "execpoolSettleTime: " + getExecpoolSettleTime();

buffer += "}";

return buffer;

}

}

## Timezone.java

package com.csg.street.persistence;

import java.io.Serializable;

import java.math.BigInteger;

import java.sql.Timestamp;

import javax.persistence.Access;

import javax.persistence.AccessType;

import javax.persistence.Cacheable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.NamedQueries;

import javax.persistence.NamedQuery;

import javax.persistence.Table;

import org.hibernate.annotations.Cache;

import org.hibernate.annotations.CacheConcurrencyStrategy;

import com.csg.street.model.PersistenceEntity;

@NamedQueries({

@NamedQuery(name="Timezone.findAll", query="SELECT e FROM Timezone e"),

@NamedQuery(name="Timezone.findById", query="SELECT e FROM Timezone e WHERE e.timezoneInternalId = :timezoneInternalId")

})

@Entity

@Table(name="ref\_TIMEZONE")

@Cacheable(true)

@Cache(usage=CacheConcurrencyStrategy.READ\_WRITE)

@Access(AccessType.FIELD)

public class Timezone implements PersistenceEntity<BigInteger>, Serializable{

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

@Column(name="TIMEZONE\_INTERNAL\_ID", unique=true, nullable=false, length=6, insertable=true, updatable=true)

private BigInteger timezoneInternalId;

@Column(name="TIMEZONE\_ID", unique=false, nullable=true, length=4, insertable=true, updatable=true)

private Integer timezoneId;

@Column(name="JAVA\_TIMEZONE", unique=false, nullable=true, length=64, insertable=true, updatable=true)

private String javaTimezone;

@Column(name="ZONE\_DESCRIPTION", unique=false, nullable=true, length=64, insertable=true, updatable=true)

private String zoneDescription;

@Column(name="DES\_TIMEZONE\_ABBR", unique=false, nullable=true, length=6, insertable=true, updatable=true)

private String desTimezoneAbbr;

@Column(name="LAST\_UPDATED\_BY\_USER", unique=false, nullable=true, length=30, insertable=true, updatable=true)

private String lastUpdatedByUser;

@Column(name="LAST\_UPDATE", unique=false, nullable=true, length=8, insertable=true, updatable=true)

private Timestamp lastUpdate;

public BigInteger getTimezoneInternalId() {

return timezoneInternalId;

}

public void setTimezoneInternalId(BigInteger timezoneInternalId) {

this.timezoneInternalId = timezoneInternalId;

}

public Integer getTimezoneId() {

return timezoneId;

}

public void setTimezoneId(Integer timezoneId) {

this.timezoneId = timezoneId;

}

public String getJavaTimezone() {

return javaTimezone;

}

public void setJavaTimezone(String javaTimezone) {

this.javaTimezone = javaTimezone;

}

public String getZoneDescription() {

return zoneDescription;

}

public void setZoneDescription(String zoneDescription) {

this.zoneDescription = zoneDescription;

}

public String getDesTimezoneAbbr() {

return desTimezoneAbbr;

}

public void setDesTimezoneAbbr(String desTimezoneAbbr) {

this.desTimezoneAbbr = desTimezoneAbbr;

}

public String getLastUpdatedByUser() {

return lastUpdatedByUser;

}

public void setLastUpdatedByUser(String lastUpdatedByUser) {

this.lastUpdatedByUser = lastUpdatedByUser;

}

public Timestamp getLastUpdate() {

return lastUpdate;

}

public void setLastUpdate(Timestamp lastUpdate) {

this.lastUpdate = lastUpdate;

}

@Override

public BigInteger getId() {

return getTimezoneInternalId();

}

@Override

public String toString() {

String buffer = "Timezone : {";

buffer += "timezoneInternalId: " + getId() + ", ";

buffer += "timezoneId: " + getTimezoneId() + ", ";

buffer += "javaTimezone: " + getJavaTimezone() + ", ";

buffer += "zoneDescription: " + getZoneDescription() + ", ";

buffer += "desTimezoneAbbr: " + getDesTimezoneAbbr() + ", ";

buffer += "lastUpdatedByUser: " + getLastUpdatedByUser() + ", ";

buffer += "lastUpdate: " + getLastUpdate();

buffer += "}";

return buffer;

}

}

## spring-otex-façade.xml

<constructor-arg ref="marketTimeOpeningDao" />

<constructor-arg ref="timezoneDao"/>

## spring-dao.xml

<bean id="marketTimeOpeningDao" class="com.csg.street.dao.MarketTimeOpeningDao"/>

<bean id="timezoneDao" class="com.csg.street.dao.TimezoneDao"/>

## persistence-jrave.xml

<class>com.csg.street.persistence.MarketTimeOpening</class>

<class>com.csg.street.persistence.Timezone</class>

## MarketTimeOpeningDaoTest.java

package com.csg.street.dao;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertNull;

import java.math.BigInteger;

import java.util.List;

import javax.annotation.Resource;

import javax.persistence.PersistenceException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import com.csg.street.common.AbstractPersistenceTest;

import com.csg.street.persistence.MarketTimeOpening;

public class MarketTimeOpeningDaoTest extends AbstractPersistenceTest {

@Resource

private MarketTimeOpeningDao dao;

private MarketTimeOpening insertEntity = new MarketTimeOpening();

private MarketTimeOpening beforeUpdate = new MarketTimeOpening();

private MarketTimeOpening afterUpdate = new MarketTimeOpening();

@Before

public void init() {

dao.removeAll();

insertEntity.setTimezoneInternalId(new BigInteger("68"));

beforeUpdate.setTimeInternalId(new BigInteger("10"));

beforeUpdate.setTimezoneInternalId(new BigInteger("69"));

afterUpdate.setTimeInternalId(new BigInteger("10"));

afterUpdate.setTimezoneInternalId(new BigInteger("100"));

}

@Test

public void testInsert() {

MarketTimeOpening inserted = dao.insert(insertEntity);

// ID gets generated during the insert.

// If an ID is avalaible, the insert worked correctly.

assertNotNull(inserted.getTimeInternalId());

}

@Test

public void testFindAll() {

dao.insert(insertEntity);

dao.insert(beforeUpdate);

List<MarketTimeOpening> results = dao.findAll();

assertEquals(2, results.size());

}

@Test(expected=PersistenceException.class)

public void testInsertFailDataTooLong() {

// provoke PersistenceException by setting a too long username (lastUpdatedByUser)

MarketTimeOpening failObject = insertEntity;

failObject.setLastUpdatedByUser("Dies ist ein ganz langer Test aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");

dao.insert(failObject);

}

@Test

public void testUpdate() {

MarketTimeOpening old = dao.insert(beforeUpdate);

MarketTimeOpening updated = dao.update(afterUpdate);

Assert.assertNotSame(old.getTimezoneInternalId(), updated.getTimezoneInternalId());

}

@Test

public void testDelete() {

MarketTimeOpening inserted = dao.insert(insertEntity);

MarketTimeOpening deleted = dao.delete(inserted.getId());

assertEquals(inserted.toString(), deleted.toString());

assertNull(dao.findById(deleted.getId()));

}

@Test

public void testDeleteFailWrongId() {

MarketTimeOpening inserted = dao.insert(insertEntity);

MarketTimeOpening deleted = dao.delete(inserted.getId().add(new BigInteger("1"))); // add 1 to id for wrong id

assertNull(deleted);

assertNotNull(dao.findById(inserted.getId()));

}

}

## MarketTimeOpeningTest.java

package com.csg.otex.guibe.impl.commands.markettimeopening;

import static org.mockito.Matchers.any;

import static org.mockito.Matchers.eq;

import static org.mockito.Mockito.when;

import java.math.BigInteger;

import java.sql.Timestamp;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.GregorianCalendar;

import java.util.List;

import junit.framework.Assert;

import org.junit.Before;

import org.junit.Test;

import com.csg.otex.guibe.api.commands.ops.DeleteMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.InsertMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningGeneralBO;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningInsertBO;

import com.csg.otex.guibe.api.commands.ops.MarketTimeOpeningUpdateBO;

import com.csg.otex.guibe.api.commands.ops.OperationServiceResponse;

import com.csg.otex.guibe.api.commands.ops.SelectMarketTimeOpening;

import com.csg.otex.guibe.api.commands.ops.SelectMarketTimeOpeningResponse;

import com.csg.otex.guibe.api.commands.ops.Status;

import com.csg.otex.guibe.api.commands.ops.UpdateMarketTimeOpening;

import com.csg.otex.guibe.impl.AbstractGuiBETest;

import com.csg.otex.guibe.impl.commands.MarketTimeOpeningActions;

import com.csg.street.persistence.MarketTimeOpening;

import com.csg.street.persistence.Timezone;

import com.sun.org.apache.xerces.internal.jaxp.datatype.XMLGregorianCalendarImpl;

public class MarketTimeOpeningTest extends AbstractGuiBETest{

private MarketTimeOpeningActions actions;

private Timezone timezone;

private MarketTimeOpening marketTimeOpening;

@Before

public void init() {

actions = new MarketTimeOpeningActions(backendInteractionMock);

timezone = new Timezone();

marketTimeOpening = new MarketTimeOpening();

}

@Test

public void testInsert() {

InsertMarketTimeOpening request = new InsertMarketTimeOpening();

request.setBusinessObject(createInsertBO(10, 0, 19, 0, 22, 30, new BigInteger("367")));

marketTimeOpening = createEntityFromBO(request.getBusinessObject().getMarketTimeOpening());

timezone.setTimezoneInternalId(marketTimeOpening.getTimezoneInternalId());

when(backendInteractionMock.getStaticDataAccessor().insertMarketTimeOpening(any(MarketTimeOpening.class))).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findTimezoneById(eq(timezone.getTimezoneInternalId()))).thenReturn(timezone);

OperationServiceResponse response = actions.insertMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.OK, response.getStatus());

Assert.assertEquals(0, response.getErrors().size());

Assert.assertEquals(marketTimeOpening.getTimeInternalId(), response.getRecordNumber());

verifyStaticDataAccessThreetimes();

}

@Test

public void testInsertHomeClosingNull() {

InsertMarketTimeOpening request = new InsertMarketTimeOpening();

request.setBusinessObject(createInsertBO(10, 0, -1, -1, 22, 30, new BigInteger("367")));

request.getBusinessObject().getMarketTimeOpening().setHomeClosing(null);

marketTimeOpening = createEntityFromBO(request.getBusinessObject().getMarketTimeOpening());

timezone.setTimezoneInternalId(marketTimeOpening.getTimezoneInternalId());

when(backendInteractionMock.getStaticDataAccessor().insertMarketTimeOpening(any(MarketTimeOpening.class))).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findTimezoneById(eq(timezone.getTimezoneInternalId()))).thenReturn(timezone);

OperationServiceResponse response = actions.insertMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.HARD\_ERROR, response.getStatus());

Assert.assertNotSame(marketTimeOpening.getTimeInternalId(), response.getRecordNumber());

Assert.assertTrue(response.getErrors().size() > 0);

verifyStaticDataAccessThreetimes();

}

@Test

public void testUpdate() {

UpdateMarketTimeOpening request = new UpdateMarketTimeOpening();

request.setBusinessObject(createUpdateBO(10, 0, 19, 0, 22, 30, new BigInteger("367")));

request.getBusinessObject().setTimeInternalId(new BigInteger("1"));

marketTimeOpening = createEntityFromBO(request.getBusinessObject().getMarketTimeOpening());

marketTimeOpening.setTimeInternalId(request.getBusinessObject().getTimeInternalId());

timezone.setTimezoneInternalId(marketTimeOpening.getTimezoneInternalId());

MarketTimeOpening oldEntity = createEntityFromBO(request.getBusinessObject().getMarketTimeOpening());

oldEntity.setLastUpdatedByUser("old testuser");

when(backendInteractionMock.getStaticDataAccessor().findMarketTimeOpeningById(request.getBusinessObject().getTimeInternalId())).thenReturn(oldEntity);

when(backendInteractionMock.getStaticDataAccessor().updateMarketTimeOpening(any(MarketTimeOpening.class))).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findTimezoneById(eq(timezone.getTimezoneInternalId()))).thenReturn(timezone);

OperationServiceResponse response = actions.updateMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.OK, response.getStatus());

Assert.assertEquals(marketTimeOpening.getTimeInternalId(), response.getRecordNumber());

Assert.assertEquals(0, response.getErrors().size());

verifyStaticDataAccessTimes(4);

}

@Test

public void testUpdateWrongForeignKey() {

UpdateMarketTimeOpening request = new UpdateMarketTimeOpening();

request.setBusinessObject(createUpdateBO(10, 0, 19, 0, 22, 30, new BigInteger("777")));

request.getBusinessObject().setTimeInternalId(new BigInteger("1"));

marketTimeOpening = createEntityFromBO(request.getBusinessObject().getMarketTimeOpening());

marketTimeOpening.setTimeInternalId(request.getBusinessObject().getTimeInternalId());

timezone.setTimezoneInternalId(new BigInteger("8888"));

MarketTimeOpening oldEntity = marketTimeOpening;

oldEntity.setLastUpdatedByUser("old testuser");

when(backendInteractionMock.getStaticDataAccessor().findMarketTimeOpeningById(request.getBusinessObject().getTimeInternalId())).thenReturn(oldEntity);

when(backendInteractionMock.getStaticDataAccessor().updateMarketTimeOpening(any(MarketTimeOpening.class))).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findTimezoneById(eq(timezone.getTimezoneInternalId()))).thenReturn(timezone);

OperationServiceResponse response = actions.updateMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.HARD\_ERROR, response.getStatus());

Assert.assertNotSame(marketTimeOpening.getTimeInternalId(), response.getRecordNumber());

Assert.assertTrue(response.getErrors().size() > 0);

verifyStaticDataAccessTimes(4);

}

@Test

public void testSelect() {

SelectMarketTimeOpening request = new SelectMarketTimeOpening();

MarketTimeOpening one = new MarketTimeOpening();

MarketTimeOpening two = new MarketTimeOpening();

MarketTimeOpening three = new MarketTimeOpening();

List<MarketTimeOpening> results = new ArrayList<MarketTimeOpening>();

results.add(one);

results.add(two);

results.add(three);

when(backendInteractionMock.getStaticDataAccessor().findAllMarketTimeOpenings()).thenReturn(results);

OperationServiceResponse response = actions.selectMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.OK, response.getStatus());

Assert.assertTrue(response.getErrors().size() < 1);

Assert.assertEquals(3, ((SelectMarketTimeOpeningResponse) response).getMarketTimeOpenings().size());

verifyStaticDataAccessTwice();

}

@Test

public void testDelete() {

DeleteMarketTimeOpening request = new DeleteMarketTimeOpening();

request.setRecordId(new BigInteger("1"));

marketTimeOpening = new MarketTimeOpening();

marketTimeOpening.setTimeInternalId(request.getRecordId());

when(backendInteractionMock.getStaticDataAccessor().deleteMarketTimeOpening(request.getRecordId())).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findMarketTimeOpeningById(eq(request.getRecordId()))).thenReturn(marketTimeOpening);

OperationServiceResponse response = actions.deleteMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.OK, response.getStatus());

Assert.assertEquals(0, response.getErrors().size());

Assert.assertEquals(request.getRecordId(), response.getRecordNumber());

verifyStaticDataAccessTimes(3);

}

@Test

public void testDeleteWrongId() {

DeleteMarketTimeOpening request = new DeleteMarketTimeOpening();

request.setRecordId(new BigInteger("1"));

marketTimeOpening = new MarketTimeOpening();

marketTimeOpening.setTimeInternalId(new BigInteger("3"));

when(backendInteractionMock.getStaticDataAccessor().deleteMarketTimeOpening(request.getRecordId())).thenReturn(marketTimeOpening);

when(backendInteractionMock.getStaticDataAccessor().findMarketTimeOpeningById(request.getRecordId())).thenReturn(null);

OperationServiceResponse response = actions.deleteMarketTimeOpening(operationHeader, request);

Assert.assertEquals(Status.HARD\_ERROR, response.getStatus());

Assert.assertNotSame(0, response.getErrors().size());

Assert.assertNotSame(request.getRecordId(), response.getRecordNumber());

verifyStaticDataAccessTimes(3);

}

private MarketTimeOpeningInsertBO createInsertBO(int hourHomeOpening, int minuteHomeOpening, int hourHomeClosing, int minuteHomeClosing, int hourExecpoolSettleTime, int minuteExecpoolSettleTime, BigInteger timezoneInternalId) {

MarketTimeOpeningInsertBO updateBo = new MarketTimeOpeningInsertBO();

updateBo.setMarketTimeOpening(createGeneralBO(hourHomeOpening, minuteHomeOpening, hourHomeClosing, minuteHomeClosing, hourExecpoolSettleTime, minuteExecpoolSettleTime, timezoneInternalId));

return updateBo;

}

private MarketTimeOpeningUpdateBO createUpdateBO(int hourHomeOpening, int minuteHomeOpening, int hourHomeClosing, int minuteHomeClosing, int hourExecpoolSettleTime, int minuteExecpoolSettleTime, BigInteger timezoneInternalId) {

MarketTimeOpeningUpdateBO insertBo = new MarketTimeOpeningUpdateBO();

insertBo.setMarketTimeOpening(createGeneralBO(hourHomeOpening, minuteHomeOpening, hourHomeClosing, minuteHomeClosing, hourExecpoolSettleTime, minuteExecpoolSettleTime, timezoneInternalId));

return insertBo;

}

private MarketTimeOpeningGeneralBO createGeneralBO(int hourHomeOpening, int minuteHomeOpening, int hourHomeClosing, int minuteHomeClosing, int hourExecpoolSettleTime, int minuteExecpoolSettleTime, BigInteger timezoneInternalId) {

MarketTimeOpeningGeneralBO bo = new MarketTimeOpeningGeneralBO();

GregorianCalendar homeClosing = new GregorianCalendar();

Calendar c = Calendar.getInstance();

c.set(Calendar.HOUR\_OF\_DAY, hourHomeClosing);

c.set(Calendar.MINUTE, minuteHomeClosing);

homeClosing.setTime(c.getTime());

GregorianCalendar homeOpening = new GregorianCalendar();

c = Calendar.getInstance();

c.set(Calendar.HOUR\_OF\_DAY, hourHomeOpening);

c.set(Calendar.MINUTE, minuteHomeOpening);

homeOpening.setTime(c.getTime());

GregorianCalendar execpoolSettleTime = new GregorianCalendar();

c = Calendar.getInstance();

c.set(Calendar.HOUR\_OF\_DAY, hourExecpoolSettleTime);

c.set(Calendar.MINUTE, minuteExecpoolSettleTime);

bo.setExecpoolSettleTime(new XMLGregorianCalendarImpl(execpoolSettleTime));

bo.setHomeClosing(new XMLGregorianCalendarImpl(homeClosing));

bo.setHomeOpening(new XMLGregorianCalendarImpl(homeOpening));

bo.setTimezoneInternalId(timezoneInternalId);

return bo;

}

private MarketTimeOpening createEntityFromBO(MarketTimeOpeningGeneralBO bo) {

MarketTimeOpening entity = new MarketTimeOpening();

entity.setTimeInternalId(new BigInteger("1"));

// all attributes must be checked if null, because of testing

if(bo.getClosingTime() != null) {

entity.setClosingTime(new Timestamp(bo.getClosingTime().toGregorianCalendar().getTimeInMillis()));

}

if(bo.getOpeningTime() != null) {

entity.setOpeningTime(new Timestamp(bo.getOpeningTime().toGregorianCalendar().getTimeInMillis()));

}

if(bo.getExecpoolSettleTime() != null) {

entity.setExecpoolSettleTime(new Timestamp(bo.getExecpoolSettleTime().toGregorianCalendar().getTimeInMillis()));

}

if(bo.getHomeClosing() != null) {

entity.setHomeClosing(new Timestamp(bo.getHomeClosing().toGregorianCalendar().getTimeInMillis()));

}

if(bo.getHomeOpening() != null) {

entity.setHomeOpening(new Timestamp(bo.getHomeOpening().toGregorianCalendar().getTimeInMillis()));

}

if(bo.getTimezoneInternalId() != null) {

entity.setTimezoneInternalId(bo.getTimezoneInternalId());

}

entity.setLastUpdate(new Timestamp(new Date().getTime()));

entity.setLastUpdatedByUser(operationHeader.getUsername());

return entity;

}

}