HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

CREALOGIX

Terminvereinbarung für E-Banking

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2014

Autor(en):	Marcel Loop, Philipp Koster
Betreuer:	Prof. Hans Rudin
Projektpartner:	Crealogix AG - Zürich
Experte:	Prof. Hans Rudin

Erklärung der Eigenständigkeit

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 18.12.2014

Name, Unterschrift: Marcel Loop

Philipp Koster

1. Abstract

CREALOGIX entwickelt das E-Banking 2.0, welches immer mehr und komplexere Funktionalitäten anbietet. Eine davon ist die Beratung über den digitalen Kanal. Dafür wird ein Modul entwickelt, womit der Kunde bequem über das Internet einen Termin mit seinem Berater vereinbaren kann.

Der Kunde meldet sich in seinem E-Banking an und sieht wann sein persönlicher Berater Zeit für ihn hat. Er wählt einen Zeitraum aus und stellt seinem Berater damit eine Anfrage zu einem Termin. Der Kunde kann auswählen ob er einen Besuch in der Filiale, ein einfaches Telefongespräch oder ein Telefongespräch mit Co-Browsing wünscht.

Die Terminanfrage wird direkt auf dem Exchange Server der Bank erstellt, womit der Berater die Anfrage über seinen Outlook Client verwalten und beantworten kann. Daraus ergibt sich der Vorteil, dass der Termin direkt im Kalender des Beraters gespeichert ist.

Sobald der Berater den Termin beantwortet hat, sieht der Kunde die Antwort im E-Banking Modul. Er kann den Termin exportieren indem er ihn herunterlädt oder sich per E-Mail zuschicken lässt.

Zur Kommunikation von einem Web-Server mit dem Exchange-Server wurde ein API entwickelt, welches die erwähnte Funktionalität anbietet. Das API kann in einem späteren Schritt von CREALOGIX in ihr Produkt implementiert werden. Zusätzlich wurde im Rahmen dieser Arbeit ein Web-Client sowie Web-Server Prototyp zur Demonstration der API-Verwendung entwickelt.

2. Management Summary

2.1 Ausgangslage

Die Firma Crealogix entwickelt das neue E-Banking 2.0, das innovatives Design und neue Funktionen bietet, die dem Kunden helfen, seine Aufgaben effizienter zu bewältigen.
Für das neue E-Banking wurde ein neues Modul entwickelt, mit dem der Kunde innerhalb des E-Bankings einen Termin mit seinem Berater planen kann. Dabei soll die Arbeitszeit sowie der aktuelle Terminkalender des Beraters beachtet werden, damit keine Terminkollision entstehen können. Der Kunde soll zudem die Möglichkeit haben, zwischen einem Treffen, Telefongespräch oder Co-Browsing zu wählen.
Nach dem Absenden der Terminanfrage soll der Berater wie gewohnt über Microsoft Outlook einen Termin empfangen und ihn auch dort verwalten können. Die Antwort des Beraters, sowie deren Status soll der Kunde innerhalb seines E-Banking einsehen können.

In dieser Semesterarbeit wurde ein API in Java entwickelt, welches die Kommunikation zwischen Applikationsserver und Exchange vereinfacht.
Damit die Verwendung der API demonstriert werden kann, wurde zusätzlich eine Serverapplikation und Webclient entwickelt, die untereinander über eine REST Schnittstelle kommunizieren.
Zudem wurde die Arbeitsumgebung inklusive Active Directory und Exchange aufgesetzt.

2.2 Administratives

Das Management der Tickets und die Arbeitszeiterfassung wurden in Redmine administriert. Jede Woche wurden neue Tickets erstellt und alte abgearbeitet. Um den Entwicklungsprozess nachvollziehen zu können wurden wöchentlich Snapshots des Gant-Diagramms erstellt. Vor jeder Sitzung wurde ein Sitzungsprotokoll mit Traktanden vorbereitet und während des Meetings Beschlüsse niedergeschrieben. Die Snapshots, wie auch die Sitzungsprotokolle befinden sich auf der Abgabe CD.

2.3 Vorgehen

Die Arbeit wurde mit einem iterativen Ansatz geplant und durchgeführt, dabei Spiegeln die Meilensteine jeweils eine Iteration.

In einem ersten Schritt galt es die Anforderungen in Zusammenarbeit mit der Crealogix zu spezifizieren und nieder zu schreiben. Aus den Anforderungen wurden dann entsprechende Use Cases und nicht funktionale Anforderungen abgeleitet, welche den Umfang der Arbeit aufzeigten. Während der Arbeit kamen optionale Anforderungen dazu, die aber erst nach Abschluss der zu Beginn definierten Anforderungen umgesetzt werden sollten.

In einem nächsten Schritt wurden bestehende APIs evaluiert, die eine Kommunikation von Java zu Microsoft Exchange ermöglichen. Durch die Evaluation wurde das von Microsoft entwickelte ews-java-api ausgewählt, auf dem wir unser API aufbauen konnten.

Um eine gute Qualität des Codes zu gewährleisten wurde von Beginn an nach Test Driven Development gearbeitet. Zudem wurde jede Codeänderung vom anderen Entwickler überprüft.

Während der Elaborationsphase wurde wöchentlich eine Besprechung mit der Crealogix abgehalten, in welchen der aktuelle Stand sowie das weitere Vorgehen geplant wurde.

2.4 Technologien

Die Crealogix gab vor, dass das API in Java implementiert werden soll. API, Client und Server sind Stateless, da die Daten immer neu vom Exchange ausgelesen werden sollen. Die Server Applikation wurde mit Java und der Webclient mit Angular JS implementiert.

2.5 Ergebnisse

Das Endprodukt der Arbeit ist ein Java API, welches einem beliebigen Client erlaubt mit Microsoft Exchange zu kommunizieren.
Das API bietet die Möglichkeit freie Termine zu suchen, Terminanfragen zu erstellen, bestehende Termine zu ändern, herunterzuladen oder sich per Mail zuzusenden.
Neben dem API wurde eine vollständige Umgebung erstellt, welche die Funktionsweise des APIs demonstriert. Dafür wurden eine Server Applikation und ein Webclient entwickelt. Die Server Applikation bietet eine REST Schnittstelle, die der Webclient mit HTTP Requests verwendet.
Die folgende Systemübersicht veranschaulicht grob das Zusammenspiel der Komponenten:

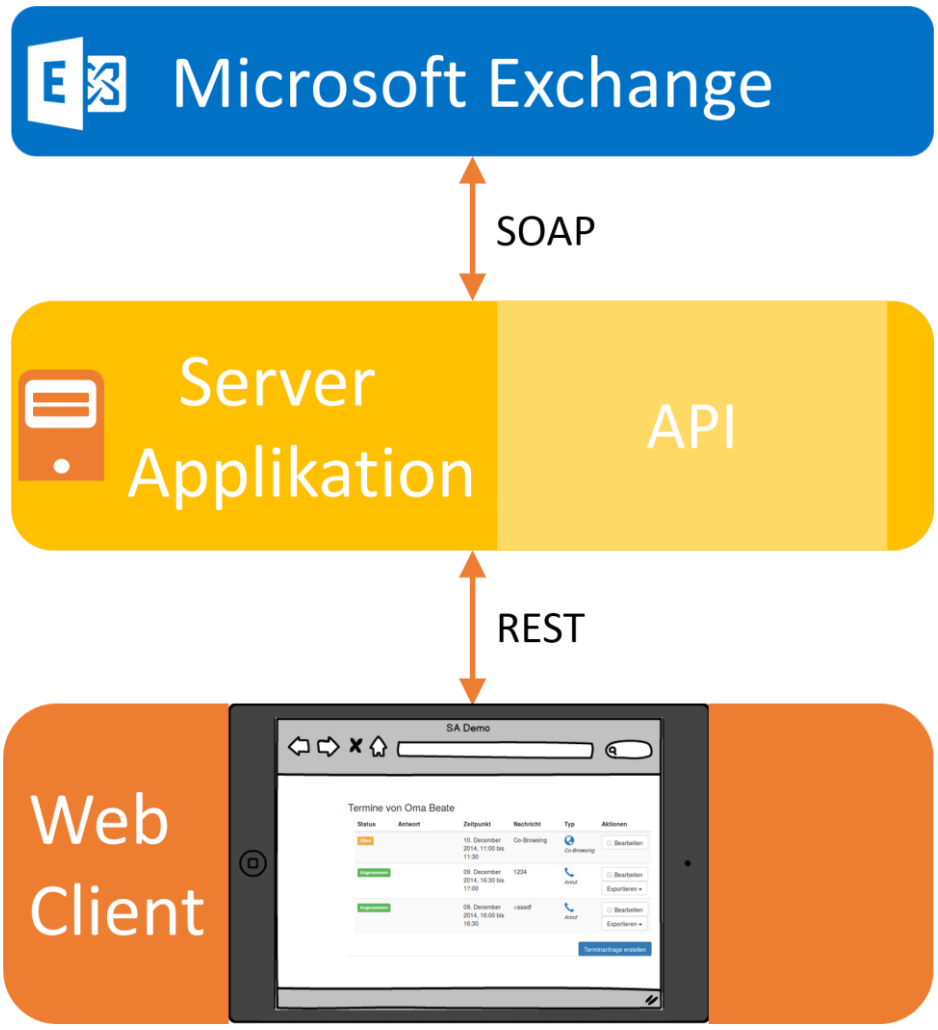


Abbildung 1: Systemübersicht

3. Anforderungsanalyse

In der Anforderungsanalyse werden die Anforderungen an das API aufgenommen, durchdacht und genau beschrieben. Dabei sollen auch alle Spezialfälle durchdacht werden, womit man grössere Anpassungen in der Implementationsphase vermeiden kann.
Die Anforderungen bieten in einem späteren Schritt die Grundlagen für Evaluationen, die Architektur und Implementierung.

3.1 Use Cases

Folgende Use Cases wurden mit dem Auftraggeber ausgearbeitet, wobei graue Use Cases optional sind.

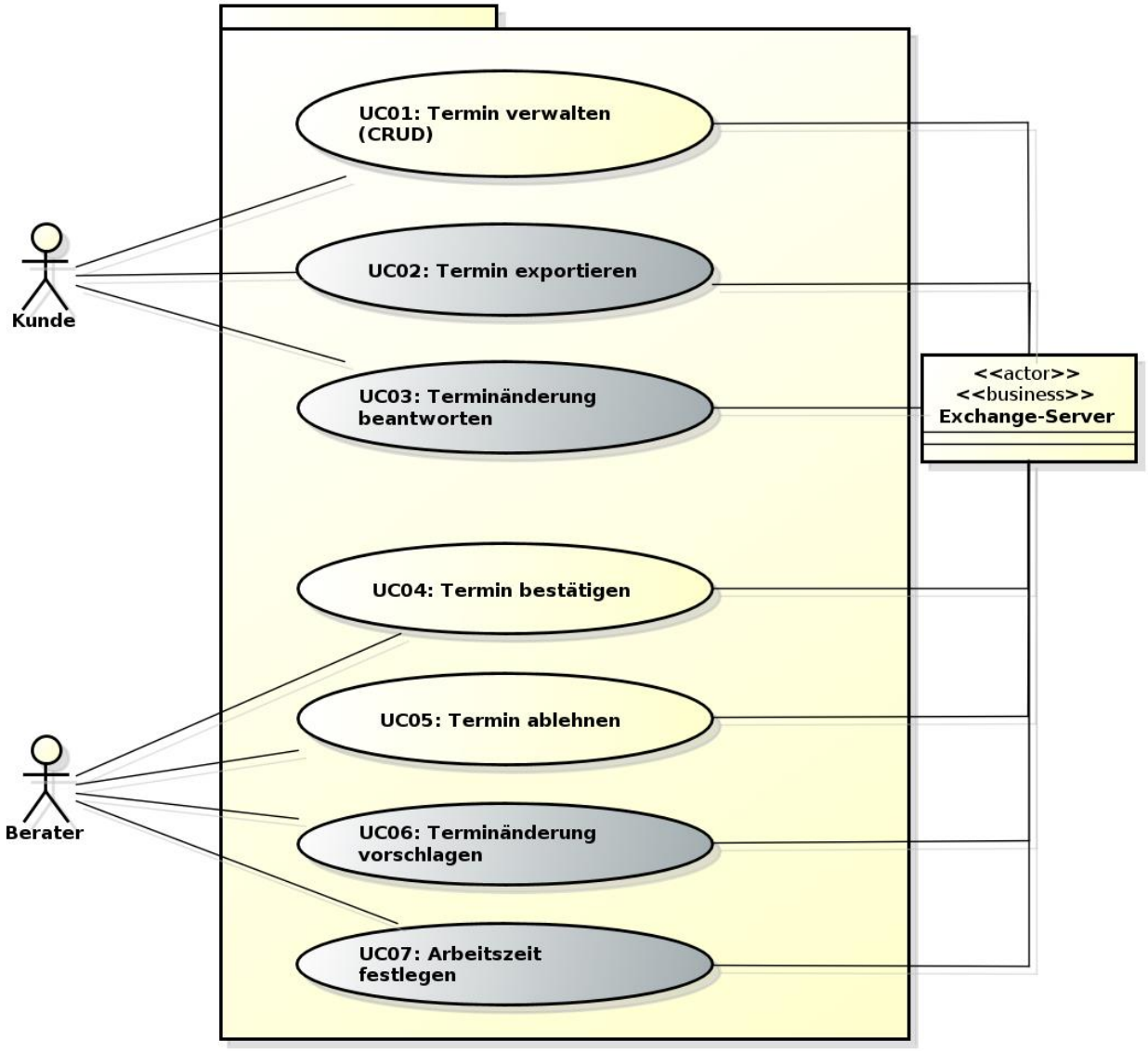


Abbildung 3-1:Use Case

3.2 Use Cases Brief

Nachfolgend eine kurze Beschreibung zu jedem Use Case.

UC01: Termin verwalten (CRUD)

Der Kunde erstellt einen Termin mit seinem Berater. Der Termin kann bearbeitet und gelöscht werden.

UC02: Termin exportieren (Optional)

Der Kunde exportiert einen Termin, indem er sich ihn per Mail zuschicken lässt oder herunterlädt.

UC03: Terminänderung beantworten (optional)

Der Kunde nimmt eine vorgeschlagene Terminänderung an oder lehnt diese ab.

UC04: Termin bestätigen

Der Berater bestätigt die Terminanfrage eines Kunden.

UC05: Termin ablehnen

Der Berater lehnt die Terminreservation eines Kunden mit einer Begründung ab.

UC06: Terminänderung vorschlagen (optional)

Der Berater schlägt einen anderen Zeitpunkt für den Termin vor.

UC07: Arbeitszeit festlegen (optional)

Der Berater kann seine Arbeitszeit festlegen, womit dem Kunden nur Termine in dieser vorgeschlagen werden.

3.3 Use Cases Fully Dressed

Nachfolgend werden die Use Cases genauer beschrieben, wobei auch Spezialfälle abgedeckt werden.

3.3.1 UC01: Termin verwalten (CRUD)

Primary Actor: Kunde

Create

Main Success Scenario:

1. (Kunde betrachtet Kalender seines Beraters)
2. Kunde wählt freien Zeitabschnitt im Kalender, an dem er einen Termin reservieren möchte (Datum, Startzeit, Endzeit).
3. Kunde gibt Daten des Appointments ein (siehe Domain Model).
4. System validiert Daten und erstellt Termin.
5. Benutzer sieht erstellten Termin im Kalender.
6. System erstellt Termin im Kalender von Berater (Exchange), wodurch dieser eine Benachrichtigungsmail erhält.
7. (Termin erstellen abgeschlossen)

Extensions (or Alternative Flows):

4a. Invalide Daten

1. Benutzer sieht Meldung über falsche Eingabe.

Update

Main Success Scenario:

1. (Kunde betrachtet seine bestehende Termine)
2. Kunde wählt bereits erstellten Termin.
3. Kunde ändert Daten des Appointment (siehe Domain Model).
4. System validiert Daten und ändert Termin in Exchange, wodurch Berater Mail der Änderungen erhält.
5. (Termin ändern abgeschlossen)

Extensions (or Alternative Flows):

4a. Termin bereits bestätigt und Datum geändert

1. System setzt Termin in Exchange auf offen, wodurch Berater Mail der Änderungen erhält.

Delete

Main Success Scenario:

1. (Kunde betrachtet seine bestehende Termine)
2. Kunde wählt bereits erstellten Termin.
3. Kunde sagt erstellten Termin ab.
4. System löscht Termin aus Exchange, wodurch Berater ein Mail mit der Information erhält, dass Termin abgesagt worden ist.
5. (Termin löschen abgeschlossen)

3.3.2 UC02: Termin exportieren (optional)

Primary Actor: Kunde

Postconditions:

- Reservierter Termin im System vorhanden

Main Success Scenario:

1. (Kunde betrachtet seine bestehende Termine)
2. Kunde wählt bereits reservierten Termin.
3. Kunde klickt auf Termin exportieren.
4. Kunde gibt Mail ein und bestätigt Export.
5. System schickt Kunde Mail mit gewähltem Termin.
6. Kunde sieht Meldung, dass Termin verschickt worden ist.
7. (Termin exportieren abgeschlossen)

3.3.3 UC03: Terminänderung beantworten (optional)

Primary Actor: Kunde

Postconditions:

- Berater hat Terminänderung vorgeschlagen

Main Success Scenario:

1. (Kunde betrachtet Kalender seines Beraters)
2. Kunde sieht Meldung, dass Terminänderung vorgeschlagen wurde.
3. Kunde bestätigt Terminänderung.
4. System setzt Termin in Exchange auf bestätigt.
5. System schickt Berater Mail, dass Terminänderung angenommen wurde.
6. (Terminänderung beantworten abgeschlossen)

Extensions (or Alternative Flows):

3a. Kunde lehnt Terminänderung ab.

1. System schickt Berater Mail, dass Terminänderung abgelehnt worden ist.
2. System löscht Termin von Exchange.
3. (Terminänderung beantworten abgeschlossen)

3.3.4 UC04: Termin bestätigen

Primary Actor: Berater

Postconditions:

- Kunde hat Termin reserviert

Main Success Scenario:

1. (Berater betrachtet Termin in Outlook-Kalender)
2. Berater bestätigt Termin in Outlook-Kalender.
3. System erkennt dass Termin bestätigt worden ist.
4. Kunde erhält Benachrichtigung, dass Termin bestätigt worden ist, wenn er eingeloggt ist oder sich das nächste Mal einloggt.
5. (Termin bestätigen abgeschlossen)

3.3.5 UC05: Termin ablehnen

Primary Actor: Berater

Postconditions:

- Kunde hat Termin reserviert

Main Success Scenario:

1. (Berater betrachtet Termin in Outlook-Kalender)
2. Berater lehnt Termin in Outlook-Kalender ab (Optimal mit Begründung).
3. System erkennt, dass Termin abgelehnt worden ist.
4. Kunde erhält Benachrichtigung, dass Termin abgelehnt worden ist, wenn er eingeloggt ist oder sich das nächste Mal einloggt.
5. (Termin ablehnen abgeschlossen)

3.3.6 UC06: Terminänderung vorschlagen (optional)

Primary Actor: Berater

Postconditions:

- Kunde hat Termin reserviert

Main Success Scenario:

1. (Berater betrachtet Termin in Outlook-Kalender)
2. Berater schlägt in Outlook-Kalender anderen Zeitpunkt für Termin vor.
3. System speichert Terminänderungsvorschlag in Exchange.
4. System benachrichtigt Kunde über Terminänderung, wenn er eingeloggt ist oder sich das nächste Mal einloggt.
5. (Terminänderung vorschlagen)

3.3.7 UC07: Arbeitszeit festlegen(optional)

Primary Actor: Berater

Main Success Scenario:

1. (Berater legt Arbeitszeit im Outlook-Kalender fest)
2. System berücksichtigt die Arbeitszeit beim Erstellen von Terminvorschlägen.
3. (Arbeitszeit festlegen abgeschlossen)

3.4 Nichtfunktionale Anforderungen (Gestützt auf ISO 9126)

Die nichtfunktionalen Anforderungen legen fest welche Eigenschaften das Produkt hat.

3.4.1 Funktionalität

Richtigkeit

Es wird gewährleistet, dass der dargestellte Stand des Kalenders höchstens eine Minute alt ist. Das heisst, wenn der Berater z.B. eine Abwesenheit in seinem Kalender erfasst, sieht dies der Kunde in seinem bereits geöffneten Kalender spätestens nach einer Minute.

Interoperabilität

Die Business Logik der Applikation wird über eine REST Schnittstelle angesteuert, womit beliebige Clients verwendet werden können. Der mitgelieferte Client ist in JavaScript implementiert.

Sicherheit

Man kann nur mit den festgelegten Berechtigungen auf einen Kalender zugreifen.

3.4.2 Zuverlässigkeit

Fehlertoleranz

Fehler werden abgefangen und der Benutzer angemessen benachrichtigt.
Tritt ein Fehler in der Business Logik auf, wird dieser zusätzlich geloggt und eine Exception mit dem passenden HTML Status Code ausgelöst.

3.4.3 Benutzbarkeit

Verständlichkeit

Durch ein einfach gehaltenes Interface ist die Applikation für den Benutzer leicht verständlich und intuitiv zu verwenden.

3.4.4 Übertragbarkeit

Anpassbarkeit

Der Server-Teil kann auf jedem Java Applikationsserver laufen.
Der Client setzt lediglich einen Browser voraus.

Installierbarkeit

Der Server-Teil kann per git heruntergeladen und maven gebuildet werden. Um das ews-java-api einzubinden steht ein README im git Repository zur Verfügung.

4. Evaluation

Da das API mit dem Exchange-Server kommunizieren muss, wird eine Schnittstelle gesucht, die verwendet werden kann. Die verschiedenen Optionen werden in der Evaluation geprüft.

4.1 Vorgehen

In einem ersten Schritt sollen verschiedene Exchange API für Java gesucht und beschrieben werden. Anschliessend wird eine API ausgewählt und eine Detailanalyse durchgeführt welche die Funktionalität prüft. Werden alle Anforderungen abgedeckt, verwenden wir dieses API für unsere Arbeit.

4.2 Umgebung

Der Prototyp wird mit einem **Exchange 2013** implementiert. Dieser soll in einem späteren Schritt erweitert werden können um **weitere Versionen** zu unterstützen.

4.3 Funktionalität

Aus den Use Cases ergeben sich folgende Anforderungen an die Funktionalität des APIs:

Anforderung	Beschreibung	Betreffende Use Cases
Freie / Belegte Zeiten abrufen	Es muss die Möglichkeit geben die Zeiten abzurufen, wann der Berater Zeit hat oder bereits beschäftigt ist.	UC01
Termin erstellen	Ein Termin muss erstellt werden können. Dabei muss folgendes möglich sein: ● Art des Termins beschreiben ● Nachricht an Termin anhängen	UC01
Termine auflisten	Termine müssen aufgelistet werden können, dabei soll es beispielsweise möglich sein nur die Termine einer bestimmten Person zu finden.	UC01
Termininformationen abrufen	Folgende Informationen eines Termins müssen abgerufen werden können: ● Zeitpunkt, Dauer und Datum ● Teilnehmer ● Hat Teilnehmer Termin bestätigt oder abgelehnt ● Art des Termins ● Nachricht des Termins	UC01, UC03
Termin ändern	Ein bestehender Termin muss bearbeitet werden können. Dabei muss folgendes möglich sein: ● Zeitpunkt und Datum ändern ● Nachricht des Termins bearbeiten (evt. Nachricht anhängen) ● Art des Termins ändern	UC01

Termin löschen	Ein bestehender Termin muss gelöscht werden können.	UC01
Termin exportieren (optional)	Der Termin soll in einem passenden Format exportiert werden können. Optional, da sich Export auch selber zusammensetzen lässt, wenn alle Informationen des Termins abgerufen werden können.	UC02
Arbeitszeit (optional)	Beim Abfragen der freien Terminslots soll die eingestellte Arbeitszeit berücksichtigt werden.	UC07

Tabelle 4-1: Funktionalität

4.4 Evaluierung Exchange API

Die wichtigsten Faktoren des API wurden durch die Crealogix vorgegeben.

- Java
- Open Source
- Aktuell
- Unterstützung ab Exchange 2010

Folgende APIs wurden evaluiert:

- ews-java-api
- jwebservices
- j-XChange
- JEC -Java Exchange Connector
- Java Bridge to Exchange

Nachfolgende detailliertere Informationen zu den einzelnen APIs.

4.4.1 ews-java-api

Hersteller	Microsoft
Lizenz	Open source (https://github.com/OfficeDev/ews-java-api)
Beschreibung	Das EWS Java API wurde am 4. Juni 2014 von Microsoft veröffentlicht und wird bis heute fortlaufend weiter entwickelt. Im Hintergrund werden SOAP Requests ausgeführt um mit Exchange zu kommunizieren
Vorteile	Open Source Fortlaufende Weiterentwicklung Ablösung für MAPI
Nachteile	Unbeantwortete Fragen in Foren, da EWS Java API noch neu ist. Unterstützt bis jetzt nur Funktionen von Exchange 2007, 2010 SP2. Bis eine Erweiterung für Exchange 2013 entwickelt und veröffentlich wird ist unbekannt
Link	https://github.com/OfficeDev/ews-java-api

Tabelle 4-2:Evaluierung ews-java-api

4.4.2 jwebservices for exchange

Hersteller	Independent Soft
Lizenz	Kostenpflichtig
Beschreibung	Java API für Microsoft Exchange. Das API bietet die komplette Exchange Web Service Funktionalität inklusive erstellen, ändern und suchen von Items. Zudem können Termine erstellt und verwaltet werden.
Vorteile	• Know How vorhanden, da die Firma mehrere Produkte für Java - Microsoft anbietet • Unterstützt Exchange 2007,2010, 2013
Nachteile	Kostenpflichtig • 1 Entwickler → 299 € • Unlimitierte Anzahl Entwickler → 699 € • Blueprint Edition (source Code) → 3999 € Unbeantwortete Fragen in Foren
Link	www.independentsoft.de

Tabelle 4-3: Evaluierung jwebservices for exchange

4.4.3 j-XChange

Hersteller	J-interop
Lizenz	Open Source
Beschreibung	J-XChange ist eine Java Implementation welche mit Hilfe von Collaboration Data Objects (CDO 1.21) mit Microsoft Exchange Server kommuniziert.
Vorteile	• Open Source
Nachteile	• API ist nicht gut dokumentiert und man findet nur wenig im Internet • Scheint nicht viel verwendet zu werden • Keine Beschreibung, Beispiele etc im Internet zu finden. • Keine Angaben zu unterstützen Exchange Versionen • CDO scheint laut Microsoft veraltet zu sein. In der momentanen Version wird vor allem SOAP für die Kommunikation verwendet
Link	http://sourceforge.net/projects/j-xchange/

Tabelle 4-4: Evaluierung j-XChange

4.4.4 JEC -Java Exchange Connector

Hersteller	NetComps Ltd
Lizenz	Kostenpflichtig
Beschreibung	JEC verwendet EWSJ um mit Exchange zu kommunizieren
Vorteile	• Umfangreiche Dokumentation zum Produkt • Nötige Kalender Funktionen enthalten • Unterstützt Exchange 2007, 2010 und 2013
Nachteile	• Kostenpflichtig • EWSJ Kommerzielle Lizenz → 599 USD
Link	http://www.javaexchangeconnector.com/

Tabelle 4-5: Evaluierung Java Exchange Connector

4.4.5 Java Bridge to Exchange

Hersteller	Moyosoft
Lizenz	Kostenpflichtig
Beschreibung	Java Bridge verwendet das Standard EWS Interface und erlaubt diverse Operationen an
Vorteile	• Sehr gut dokumentiert mit API Reference(Java Doc) • Keine externe Abhängigkeiten zu anderen Libraries • Verwendet EWS API • Unterstützt Exchange 2007, 2010, 2013

Nachteile	• Kostenpflichtig
	  • Base Edition → 199 € für 1 Entwickler
	  • Professional Edition → 497 € für 3 Entwickler
	  • Team Edition → 970 € Unlimitiert
	• In der Base Edition für 199 € werden Funktionen wie Events nicht mitgeliefert

Link	http://www.moyobase.com/jbex/

Tabelle 4-6: Evaluierung Java Bridge to Exchange

4.5 Auswahl

Das Open Source Projekt ews-java-api von Microsoft ist aktuell und wird aktiv weiterentwickelt, was man an den commits im git Respoitory (https://github.com/OfficeDev/ews-java-api) sehen kann.
JWebservices, JEC und Java Bridge to Exchange unterstützen zwar Funktionen von Exchange 2013, jedoch sind alle proprietär und gewünscht wird eine Open-Source Lösung. Aus diesen Gründen werden wir ews-java-api für die weitere Detailanalyse verwenden und gegebenenfalls als Basis API einsetzen. Falls wir nicht alle Anforderungen abdecken können bleibt immer noch die Möglichkeit die EWS-Schnittstelle direkt mit SOAP Requests auszusprechen.
Da das ews-java-api nur die Exchange Versionen bis 2010 SP2 unterstützt, setzen wir auf die Rückwärtskompatibilität der neueren Versionen. Das heisst ein Exchange Server mit der Version 2013 versteht Requests die für einen Exchange Server mit der Version 2010 erstellt worden sind.

4.6 Detailanalyse ews-java-api

In der Detailanalyse wird überprüft ob mit dem ews-java-api alle Anforderungen abgedeckt werden können. Dabei werden Erkenntnisse und Beispiele festgehalten.

Symbol	Anforderung
	Überprüft und funktioniert
	Nicht erfüllt

Tabelle 4-7: Symbol Legende

Quelle Icons: https://www.iconfinder.com/icons

Nachfolgende Tabelle führt auf welche Anforderungen aus der Evaluation erfüllt werden.

Anforderung	Beschreibung	Erfüllt?
Freie / Belegte Zeiten abrufen	Es muss die Möglichkeit geben die Zeiten abzurufen, wann der Berater Zeit hat oder bereits beschäftigt ist.	
Termin erstellen	Ein Termin muss erstellt werden können. Dabei muss folgendes möglich sein: ● Art des Termins beschreiben Nachricht an Termin anhängen	
Termine auflisten	Termine müssen aufgelistet werden können, dabei soll es beispielsweise möglich sein nur die Termine einer bestimmten Person zu finden.	
Termininformationen abrufen	Folgende Informationen eines Termins müssen abgerufen werden können: ● Zeitpunkt, Dauer und Datum ● Teilnehmer ● Hat Teilnehmer Termin bestätigt oder abgelehnt ● Art des Termins ● Nachricht des Termins ● Status der Teilnehmer Antwort von Teilnehmer, wenn beantwortet	
Termin ändern	Ein bestehender Termin muss bearbeitet werden können. Dabei muss folgendes möglich sein: ● Zeitpunkt und Datum ändern ● Nachricht des Termins bearbeiten (evt. Nachricht anhängen) Art des Termins ändern	
Termin löschen	Ein bestehender Termin muss gelöscht werden können.	

Tabelle 4-8: Ergebnisse Detailanalyse

4.6.1 Analyse Anforderungen

Freie / Belegte Zeiten abrufen (UC01)

Erkenntnisse

Durch die Klasse “GetUserAvailability” lassen sich die freien sowie belegte Termine in Form von “*CalenderEvents*” auslesen.

Anwendungsbeispiel

```
    List < AttendeeInfo > attendees = new ArrayList < AttendeeInfo > ();
     attendees.add(new AttendeeInfo("pkoster@acs.local"));

     SimpleDateFormat formatter = new SimpleDateFormat("yyyy/MM/dd");
     Date start = formatter.parse("2014/10/10");
     Date end = formatter.parse("2014/10/20");

     GetUserAvailabilityResults results = service.getUserAvailability(
     attendees, new TimeWindow(start, end),
     AvailabilityData.FreeBusyAndSuggestions);

     int attendeeIndex = 0;

     for (AttendeeAvailability attendeeAvailability: results.getAttendeesAvailability())
{
         System.out.println("Availability for " + attendees.get(attendeeIndex));
         if (attendeeAvailability.getErrorCode() == ServiceError.NoError) {
             for (CalendarEvent calendarEvent: attendeeAvailability.getCalendarEvents())
{

                 System.out.println("Calendar event");
                 System.out.println("  Start time: " +
calendarEvent.getStartTime().toString());
                 System.out.println("  End time: " +
calendarEvent.getEndTime().toString());

                 if (calendarEvent.getDetails() != null) {
                     System.out.println("  Subject: " +
calendarEvent.getDetails().getSubject());
                 }
             }
         }
         attendeeIndex++;
     }
```

Code Snippet 4-1: ews-exchange-api, freie / belegte Zeit lesen

Termine auflisten (UC01)

Erkenntnisse

Um die Suche mittels Filter einzugrenzen muss die Methode findItems und nicht findAppointments verwendet werden.

Anwendungsbeispiel

Termine im Kalender des Benutzers können mit folgender Query gefunden werden:

```
CalendarFolder cf = CalendarFolder.bind(service, WellKnownFolderName.Calendar);
FindItemsResults<Appointment> findResults = cf.findAppointments(new
CalendarView(startDate, endDate));
```

Code Snippet 4-2: ews-exchange-api, Termine lesen

Alternativ kann folgende Methode verwendet werden, womit man die Resultate mittels Filter eingrenzen kann (in diesem Fall werden nur die Appointments zurückgegeben, dessen Organisator mloop@acs.local ist).

```
ItemView view = new ItemView(100);
FindItemsResults<Item> findItems = service.findItems(WellKnownFolderName.Calendar, new
SearchFilter.ContainsSubstring(AppointmentSchema.Organizer, "mloop@acs.local"), view);
```

Code Snippet 4-3: ews-exchange-api, Termine lesen

Termininformationen abrufen (UC01)

Erkenntnisse

Um die Suche mittels Filter einzugrenzen muss die Methode *findItems* und nicht *findAppointments* verwendet werden.

ResponseStatus von Attendees lässt sich auslesen, aber für den Response-Text muss die entsprechende Mail gesucht & geparset werden.

Anwendungsbeispiel

Die Termininformationen können wie folgt ausgelesen werden

```
FindItemsResults < Appointment > results = findCalendarItems();
service.loadPropertiesForItems(results, PropertySet.FirstClassProperties);
for (Appointment appointment: results.getItems()) {
    System.out.println("Subject: " + appointment.getSubject());
    System.out.println("Start: " + appointment.getStart().toString());
    System.out.println("End: " + appointment.getStart().toString());
    System.out.println("Body: " + appointment.getBody());


     for (Attendee attendee : appointment.getRequiredAttendees()) {
     System.out.println("     Name: " + attendee.getName());
     System.out.println("     Response: " + attendee.getResponseType());
    }
```

Code Snippet 4-4: ews-exchange-api, Termininformationen lesen

Termin ändern (UC03)

Fälle

1. Terminänderung nachdem Empfänger bestätigt hat
2. Terminänderung bevor Empfänger bestätigt hat

Erkenntnisse

- Beim ersten Testfall wird eine neue E-Mail an den Empfänger gesandt, dieser kann wiederum entscheiden ob er den neuen Termin annehmen oder ablehnen will
- Termin wird nur als nicht angenommen gesetzt, wenn sich das Datum oder die Zeit ändert.
- Beim zweiten Testfall kriegt der Empfänger eine neue Einladung. Die alte wird als gelöscht angezeigt

Anwendungsbeispiel

```
Appointment appointment = Appointment.bind(service, new ItemId(uniqueId));
       appointment.setBody(MessageBody.getMessageBodyFromText("Update Termin"));
appointment.setStart(startDate);
appointment.setEnd(endDate);
appointment.setSubject("Appointement UPDATE");
appointment.update(ConflictResolutionMode.AutoResolve);
```

Code Snippet 4-5: ews-exchange-api, Termin ändern

Termin löschen und absagen (UC01)

Erkenntnisse

Achtung: Delete entspricht nicht Cancel! D.h. über das Outlook-Webinterface ist ein Delete nur möglich, wenn Meeting bereits abgesagt.

Anwendungsbeispiel

Delete:

```
findCalendarItems().getItems().get(0).delete(DeleteMode.HardDelete);
```

Code Snippet 4-6: ews-exchange-api, Termin löschen

Cancel:

```
appointment.cancelMeeting("Ich habe Bauchschmerzen");
```

Code Snippet 4-7: ews-exchange-api, Termin absagen

4.7 Detailanalyse EWS SOAP

Um die Möglichkeit offen zu halten direkt SOAP Requests zu verschicken, falls das ews-java-api gewisse Funktionen nicht oder fehlerhaft implementiert, werden nachfolgend die Anforderungen mit SOAP Requests evaluiert.
Bei SOAP handelt es sich um ein Netzwerkprotokoll zum Austausch von Daten zwischen Systemen. Dargestellt wird es im XML Format. Die Microsoft Web Services lassen sich direkt mit SOAP Requests ansteuern. Bereitgestellte EWS-APIs generieren ebenfalls nur SOAP Requests.

4.7.1 Konfiguration

Zur Evaluation kann SoapUI verwendet werden. Folgendes sind die wichtigen Konfigurationsparameter:

EntryPoint:

https://sinv-56040.edu.hsr.ch/EWS/Exchange.asmx

Im Entry Point wird auf die SOAP Schnittstelle verwiesen, welche Exchange zur Verfügung stellt.
Im folgenden Beispiel wird die “GetUserAvailability” aus der Serviceschnittstelle dargestellt.

```
<!-- GetUserAvailability -->
−<wsdl:operation name="GetUserAvailability">
   <soap:operation soapAction="http://schemas.microsoft.com/exchange/services/2006/messages/GetUserAvailability"/>
  −<wsdl:input>
      <soap:header message="tns:GetUserAvailabilitySoapIn" part="Impersonation" use="literal"/>
      <soap:header message="tns:GetUserAvailabilitySoapIn" part="TimeZoneContext" use="literal"/>
      <soap:header message="tns:GetUserAvailabilitySoapIn" part="RequestVersion" use="literal"/>
      <soap:body parts="GetUserAvailabilityRequest" use="literal"/>
      <soap:header message="tns:GetUserAvailabilitySoapIn" part="RequestVersion" use="literal"/>
   </wsdl:input>
  −<wsdl:output>
      <soap:body parts="GetUserAvailabilityResult" use="literal"/>
      <soap:header message="tns:GetUserAvailabilitySoapOut" part="ServerVersion" use="literal"/>
   </wsdl:output>
</wsdl:operation>
```

Code Snippet 4-8: Exchange WSDL

Authentication:

NTLM mit Username & Password sowie acs.local als Domain.

SOAP-Header

Im Soap-Header soll die Version auf Exchange 2010 gesetzt werden:

```
<soap:Header>
 <t:RequestServerVersion Version="Exchange2010"/>
</soap:Header>
```

Code Snippet 4-9: SAOP Header

Bei den nachfolgenden Beispielen werden lediglich die relevanten Teile der Requests abgebildet.

4.7.2 Anforderungen

Freie / Belegte Zeiten abrufen

Erkenntnisse

Mit der Operation ‘GetUserAvailability’ kann abgefragt werden wann ein Benutzer zur Verfügung steht oder beschäftigt ist. Dafür wird im Request eine Zeitspanne von max. 62 Tagen angegeben, welche man überprüfen möchte.
Damit erhält man alle CalenderEvents mit den jeweiligen Zeiten und dem BusyType.

Anwendungsbeispiel

Benutzer: mloop@acs.local
Zeitraum: 1. Oktober 2014 bis 30. Oktober 2014

```
<MailboxDataArray>
<t:MailboxData>
  <t:Email>
    <t:Address>mloop@acs.local</t:Address>
  </t:Email>
  <t:AttendeeType>Required</t:AttendeeType>
  <t:ExcludeConflicts>false</t:ExcludeConflicts>
</t:MailboxData>
     </MailboxDataArray>
     <t:FreeBusyViewOptions>
<t:TimeWindow>
  <t:StartTime>2014-10-01T00:00:00</t:StartTime>
  <t:EndTime>2014-10-30T23:59:59</t:EndTime>
</t:TimeWindow>
```

Code Snippet 4-10: SOAP, freie / belegte Zeiten lesen

Antwort:

```
<CalendarEventArray xmlns="http://schemas.microsoft.com/exchange/services/2006/types">
 <CalendarEvent>
    <StartTime>2014-10-02T22:00:00</StartTime>
    <EndTime>2014-10-03T22:00:00</EndTime>
    <BusyType>Free</BusyType>
 </CalendarEvent>
 <CalendarEvent>
    <StartTime>2014-10-07T09:00:00</StartTime>
    <EndTime>2014-10-07T14:30:00</EndTime>
    <BusyType>Tentative</BusyType>
 </CalendarEvent>
 <CalendarEvent>
    <StartTime>2014-10-25T12:00:00</StartTime>
    <EndTime>2014-10-25T13:00:00</EndTime>
    <BusyType>Busy</BusyType>
 </CalendarEvent>
 <CalendarEvent>
    <StartTime>2014-10-26T12:00:00</StartTime>
    <EndTime>2014-10-26T13:00:00</EndTime>
    <BusyType>Busy</BusyType>
 </CalendarEvent>
</CalendarEventArray>
```

Code Snippet 4-11: SOAP, Antwort freie / belegte Zeiten

Termin erstellen

Erkenntnisse

Mit <Subject>*Ereignis* </Subject> kann angegeben werden ob es sich beim Termin um einen Telefonanruf, Gespräch oder auch Co Browsing handelt.
Die Beschreibung kommt in das Feld <Body BodyType="Text"> *Weitere Beschreibung des Termins*</Body>

Anwendungsbeispiel

Erstellung eines Termins mit folgenden Eigenschaften:

- Datum: 26.10.2014
- Von 13:00 - 14:00
- Ereignis: Telefonanruf von Kunde Muster Hans
- Beschreibung: Muster Hans möchte mehr über das Produkt x erfahren
- Empfänger: moop@acs.local

```
<Items>
   <t:CalendarItem xmlns="http://schemas.microsoft.com/exchange/services/2006/types">
       <Subject>Telefonanruf von Kunde Muster Hans</Subject>
       <Body BodyType="Text">Muster Hans möchte mehr über das Produkt x
           erfahren
       </Body>
       <Start>2014-10-26T14:00:00</Start>
       <End>2014-10-26T15:00:00</End>
       <RequiredAttendees>
           <Attendee>
               <Mailbox>
                   <EmailAddress>mloop@acs.local</EmailAddress>
               </Mailbox>
           </Attendee>
       </RequiredAttendees>
   </t:CalendarItem>
</Items>
```

Code Snippet 4-12: SOAP, Termin erstellen

Terminformationen abrufen

Erkenntnisse

Um an die Termininformationen zu gelangen braucht es zwei Schritte.

1. calenderID des Aufrufers abfragen mit GetFolder
2. calenderID und Zeitraum (max 2 Jahre) angeben. Um die eingetragenen Termine zu bekommen mit findItem

Anwendungsbeispiel

Abrufen der calenderID:

```
<m:GetFolder>
	<m:FolderShape>
	<t:BaseShape>IdOnly</t:BaseShape>
	</m:FolderShape>
	<m:FolderIds>
	<t:DistinguishedFolderId Id="calendar"/>
	</m:FolderIds>
</m:GetFolder>
```

Code Snippet 4-13: SOAP, calendarID abrufen

Termine auslesen mit calenderID:

```
<m:FindItem Traversal="Shallow">
	<m:ItemShape>
	<t:BaseShape>IdOnly</t:BaseShape>
	<t:AdditionalProperties>
	<t:FieldURI FieldURI="item:Subject"/>
	<t:FieldURI FieldURI="calendar:Start"/>
	<t:FieldURI FieldURI="calendar:End"/>
	</t:AdditionalProperties>
	</m:ItemShape>
	<m:CalendarView StartDate="2014-10-01T17:30:24.127Z" EndDate="2014-10-
30T17:30:24.127Z" />
	<m:ParentFolderIds>
	<t:FolderId
Id="AAMkAGUzNTIwM2I1LTUwMDQtNDAxMS1iYTk5LTgzZDU0YWE0ZGZiYwAuAAAAAAAZ/3qbWm6vS7k6NrlVnqESAQAz
/sVlQbGRQbPifGQ+1I2pAAAAAAEOAAA=" ChangeKey="AgAAABYAAAAz/sVlQbGRQbPifGQ+1I2pAAAAAAy9"/>
	</m:ParentFolderIds>
</m:FindItem>
```

Code Snippet 4-14: SOAP, Termininformationen mit calendarID

Antwort

Rückgabe der calenderID

```
<m:GetFolderResponse xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
	<m:ResponseMessages>
	<m:GetFolderResponseMessage ResponseClass="Success">
	<m:ResponseCode>NoError</m:ResponseCode>
	<m:Folders>
		<t:CalendarFolder>
			<t:FolderId ChangeKey="AgAAABYAAADh8+8GX2M7SJg5LkWx0FSaAAAAAADV"
Id="AQMkAGZkYjBmYjFlLTkyZjktNDQBMy1iN2VjLWVlATVjNTJkODFkMAAuAAADl6yb7i9slkWaVH07K4dm+AEA4fPv
Bl9jO0iYOS5FsdBUmgAAAgEOAAAA"/>
		</t:CalendarFolder>
	</m:Folders>
	</m:GetFolderResponseMessage>
	</m:ResponseMessages>
</m:GetFolderResponse>
```

Code Snippet 4-15: SOAP, Antowort calendarID

Rückgabe der Termine

```
<m:FindItemResponse xmlns:m="http://schemas.microsoft.com/exchange/services/2006/messages"
xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types">
	<m:ResponseMessages>
	<m:FindItemResponseMessage ResponseClass="Success">
	<m:ResponseCode>NoError</m:ResponseCode>
	<m:RootFolder IncludesLastItemInRange="true" TotalItemsInView="5">
		<t:Items>
			<t:CalendarItem>
			<t:ItemId ChangeKey="DwAAABYAAADh8+8GX2M7SJg5LkWx0FSaAAAAAAzf"
Id="AQMkAGZkYjBmYjFlLTkyZjktNDQBMy1iN2VjLWVlATVjNTJkODFkMABGAAADl6yb7i9slkWaVH07K4dm+AcA4fPv
Bl9jO0iYOS5FsdBUmgAAAgEOAAAA4fPvBl9jO0iYOS5FsdBUmgAAAgziAAAA"/>
			<t:Subject>Crealogix arbeiten</t:Subject>
			<t:Start>2014-10-02T22:00:00Z</t:Start>
			<t:End>2014-10-03T22:00:00Z</t:End>
			</t:CalendarItem>
			<t:CalendarItem>
			<t:ItemId ChangeKey="DwAAABYAAADh8+8GX2M7SJg5LkWx0FSaAAAAAAzg"
Id="AQMkAGZkYjBmYjFlLTkyZjktNDQBMy1iN2VjLWVlATVjNTJkODFkMABGAAADl6yb7i9slkWaVH07K4dm+AcA4fPv
Bl9jO0iYOS5FsdBUmgAAAgEOAAAA4fPvBl9jO0iYOS5FsdBUmgAAAgzjAAAA"/>
			<t:Subject>Arbeiten</t:Subject>
			<t:Start>2014-10-07T09:00:00Z</t:Start>
```

```
                    <t:End>2014-10-07T14:30:00Z</t:End>
                    </t:CalendarItem>
                    <t:ItemId ChangeKey="DwAAABYAAADh8+8GX2M7SJg5LkWx0FSaAAAAAAzk"
Id="AQMkAGZkYjBmYjFlLTkyZjktNDQBMy1iN2VjLWVlATVjNTJkODFkMABGAAADl6yb7i9slkWaVH07K4dm+AcA4fPv
Bl9jO0iYOS5FsdBUmgAAAgEOAAAA4fPvBl9jO0iYOS5FsdBUmgAAAgzkAAAA"/>
                    <t:Subject>Planning Meeting</t:Subject>
                    <t:Start>2014-10-25T12:00:00Z</t:Start>
                    <t:End>2014-10-25T13:00:00Z</t:End>
                </t:Items>
        </m:RootFolder>
        </m:FindItemResponseMessage>
        </m:ResponseMessages>
</m:FindItemResponse>
```

Code Snippet 4-16: SOAP, Antwort Termine

Termin ändern

Erkenntnisse
Im Anwendungsbeispiel werden alle aus den Anforderungen relevanten Attribute eines Termins geändert.

Anwendungsbeispiel:

```
<m:UpdateItem ConflictResolution="AlwaysOverwrite"
SendMeetingInvitationsOrCancellations="SendToAllAndSaveCopy">
   <m:ItemChanges>
       <t:ItemChange>
           <t:ItemId ChangeKey="DwAAABYAAAAz/sVlQbGRQbPifGQ+1I2pAAAAABDI"
Id="AAMkAGUzNTIwM2I1LTUwMDQtNDAxMS1iYTk5LTgzZDU0YWE0ZGZiYwBGAAAAAAAZ/3qbWm6vS7k6NrlVnqESBwAz
/sVlQbGRQbPifGQ+1I2pAAAAAAEOAAAz/sVlQbGRQbPifGQ+1I2pAAAAABDKAAA="/>
           <t:Updates>
               <t:SetItemField>
                   <t:FieldURI FieldURI="item:Subject"/>
                   <t:CalendarItem>
                       <t:Subject>CRX arbeiten</t:Subject>
                   </t:CalendarItem>
               </t:SetItemField>
               <t:SetItemField>
                   <t:FieldURI FieldURI="item:Body"/>
                   <t:CalendarItem>
                       <t:Body BodyType="Text">Wichtiger Termin</t:Body>
                   </t:CalendarItem>
               </t:SetItemField>
               <t:SetItemField>
                   <t:FieldURI FieldURI="calendar:Location"/>
                   <t:CalendarItem>
                       <t:Location>Bubikon</t:Location>
                   </t:CalendarItem>
               </t:SetItemField>
               <t:SetItemField>
                   <t:FieldURI FieldURI="calendar:Start"/>
                   <t:CalendarItem>
                       <t:Start>2014-10-04T12:30:00Z</t:Start>
                   </t:CalendarItem>
               </t:SetItemField>
               <t:SetItemField>
                   <t:FieldURI FieldURI="calendar:End"/>
                   <t:CalendarItem>
                       <t:End>2014-10-04T13:30:00Z</t:End>
                   </t:CalendarItem>
               </t:SetItemField>
           </t:Updates>
       </t:ItemChange>
   </m:ItemChanges>
</m:UpdateItem>
```

Code Snippet 4-17: SOAP, Termin ändern

4.7.3 Erkenntnisse

Folgende Erkenntnisse mit Auswirkungen auf unser Projekt haben sich aus der Evaluation ergeben:

- Zwischen den Exchange Versionen 2007 & 2010 gab es die grössten Änderungen, die eine Rückwärtskompatibilität erschweren. Deswegen wäre es sinnvoll 2010 als Version für die Requests einzusetzen, womit alle höheren Versionen durch ihre Rückwärtskompatibilität abgedeckt werden.
- Werden Anforderungen mit dem ews-java-api nicht erfüllbar, so könnte man eigene SOAP Requests generieren um an die gewünschten Informationen zu kommen.

4.8 Machbarkeit

Durch die Detailanalyse des java-ews-api und SOAP-EWS konnte festgestellt werden, dass alle nötigen Anforderungen erfüllt werden können und die Arbeit somit durchführbar ist. In einem nächsten Schritt wird nun eine Domain Analyse sowie die Architektur erstellt.

5. Domainanalyse

In der nachfolgenden Domainanalyse werden die für die API benötigten Klassen und Attribute aufgeführt. Diese ergeben sich aus den Use Cases.

5.1.1 Domain Model

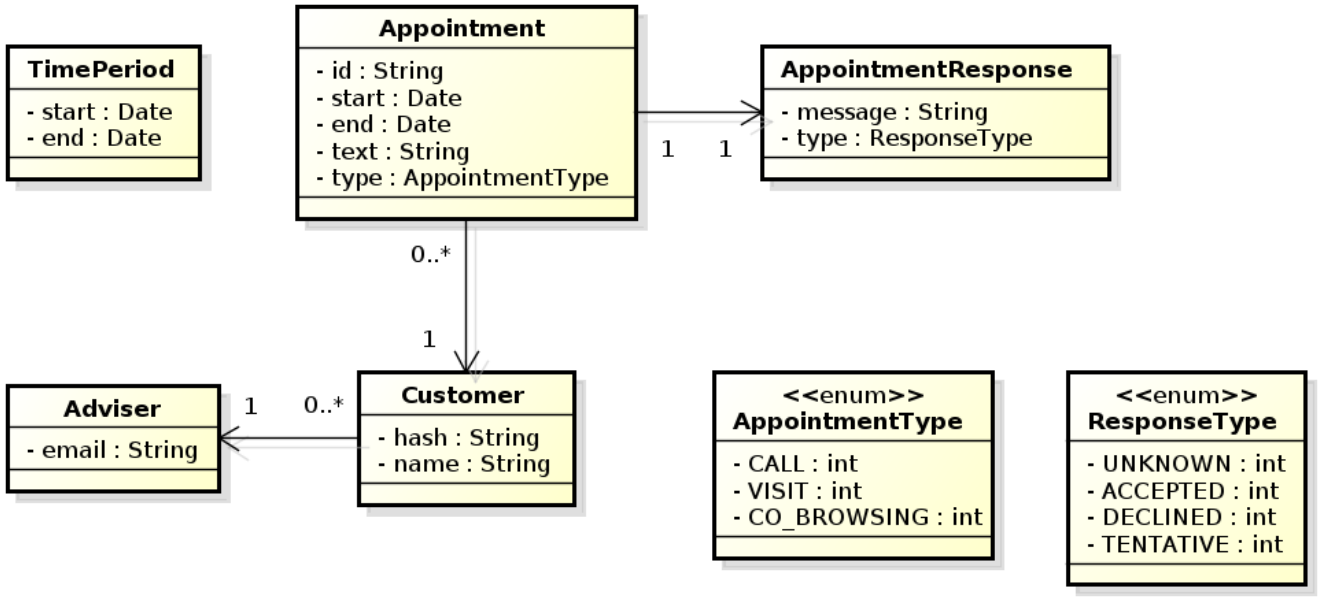


Abbildung 5-1: Domain Model

Nachfolgend werden alle Klassen des Domain Models beschrieben. Es werden nur Attribute und Beziehungen aufgeführt, die einer Beschreibung bedürfen. Selbstverständliche werden nicht behandelt. Ausserdem wird jeweils nur ein Ende einer Beziehung beschrieben.

Klasse Appointment

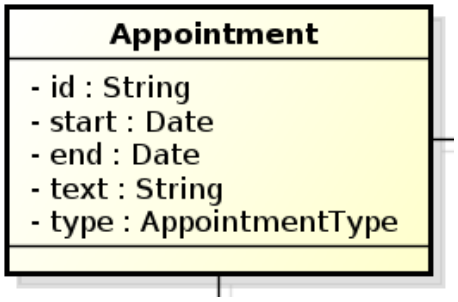


Abbildung 5-2:Klasse Appointment

Die Klasse Appointment entspricht einer akzeptierten, abgelehnten oder noch nicht beantworteten Terminanfrage.

Attribut	Type	Beschreibung
id	String	Die id entspricht einer UniqueID (siehe http://msdn.microsoft.com/en-us/library/office/dn605828(v=exchg.150).aspx) Jedes Item in Microsoft Exchange besitzt eine UniqueID welche universal einzigartig ist. Diese id wird gebraucht um ein Binding zu einem Exchange Objekt herzustellen. *Beispiel einer UniqueID:* *"AAMkAGMzNzI3YjY3LTA2MTktNDAwNi05ZjNiLTMxZWY0M mViYzQxNABGAAAAAADZOtsSrw2uRI3T4ugKHso8BwB0i+2xEpx6Q GIYBQP2MgnAAAAAAEOBBB0i+2xEpx6QZGIYBQP2MgnAAAGAAJjAAA="*
start	Date	
end	Date	
text	String	Nachricht, welche an den Berater geschickt wird.
type	AppointmentType	siehe enum AppointmentType

Tabelle 5-1: Klasse Appointment Beschreibung

Assoziation	Beschreibung
AppointmentResponse	Ist gesetzt, wenn der Berater auf die Terminanfrage geantwortet hat.
Customer	Link zum Kunden, der die Terminanfrage ausgeführt hat.

Tabelle 5-2: Klasse Appointment Assoziationen

Enum AppointmentType

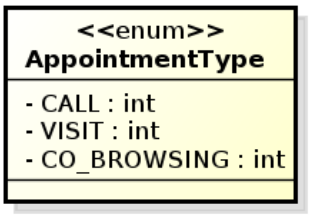


Abbildung 5-3: Enum AppointmentType

Gibt die Art eines Termins an.

Attribut	Type	Beschreibung
CALL	0	Der Kunde wünscht einen Anruf.
VISIT	1	Der Kunde möchte seinen Berater persönlich treffen.
CO_BROWSING	2	Der Kunde möchte mit seinem Berater eine Co-Browsing Sitzung durchführen.
CALL	0	Der Kunde wünscht einen Anruf.
VISIT	1	Der Kunde möchte seinen Berater persönlich treffen.

Tabelle 5-3: Enum AppointmentType Beschreibung

Klasse AppointmentResponse

Abbildung 5-4:Klasse AppointmentResponse

Die Klasse AppointmentResponse beschreibt die Antwort auf eine Terminanfrage des Kunden.

Attribut	Type	Beschreibung
message	String	Entspricht der Nachricht, die der Berater beim Beantworten des Termins eingetippt hat.
type	ResponseType	siehe enum ResponseType

Tabelle 5-4: Klasse AppointmentResponse Beschreibung

Enum ResponseType

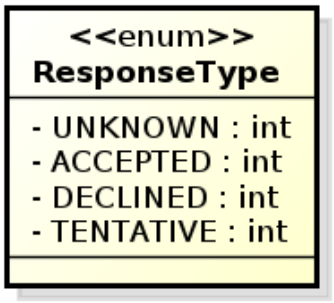


Abbildung 5-5: Enum ResponseType

Gibt die Art einer Antwort auf eine Terminanfrage an.

Attribut	Type	Beschreibung
UNKNOWN	0	Der Berater hat die Anfrage noch nicht beantwortet.
ACCEPTED	1	Der Berater hat die Anfrage akzeptiert.
DECLINED	2	Der Berater hat die Anfrage abgelehnt.
TENTATIVE	3	Der Berater hat die Anfrage mit Vorbehalt akzeptiert.

Tabelle 5-5: Enum ResponseType Beschreibung

Klasse Customer

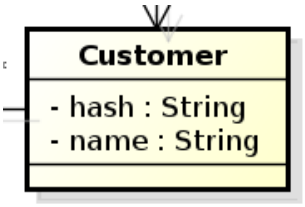


Abbildung 5-6: Klasse Customer

Klasse Adviser

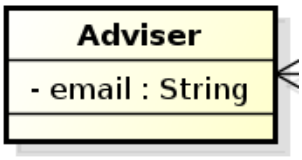


Abbildung 5-7: Klasse Adviser

Ein Berater wird lediglich durch seine E-Mail Adresse identifiziert.

6. Architektur

Nachfolgend wird auf die Architektur aller Komponenten und deren Zusammenspiel eingegangen. Dabei wird zuerst eine grobe Sytemübersicht gezeigt und dann jede Komponente detailliert beschrieben.

6.1 Systemübersicht

6.1.1 Kontextdiagramm

Nachfolgendes Kontextdiagramm zeigt wie die Kommunikation zwischen dem Kunden und dem Berater über die einzelnen Komponenten abläuft.
Der Kunde möchte gerne einen Termin mit seinem Berater. Dafür werden ihm die Termine angezeigt, an denen der Berater Zeit hat. Der Kunde wählt einen Termin, füllt die Details aus und schickt die Anfrage ab. Der Server leitet die Anfrage an das API weiter, welches diese als Termin im Exchange-Server speichert.
Der Berater erhält in seinem gewohnten E-Mail Client eine neue Nachricht mit den Angaben des Kunden. Der Berater nimmt den Termin an oder lehnt ihn ab, in beiden Fällen wird der Kunde die Antwort mit einem optionalen Kommentar in seinem E-Banking sehen.

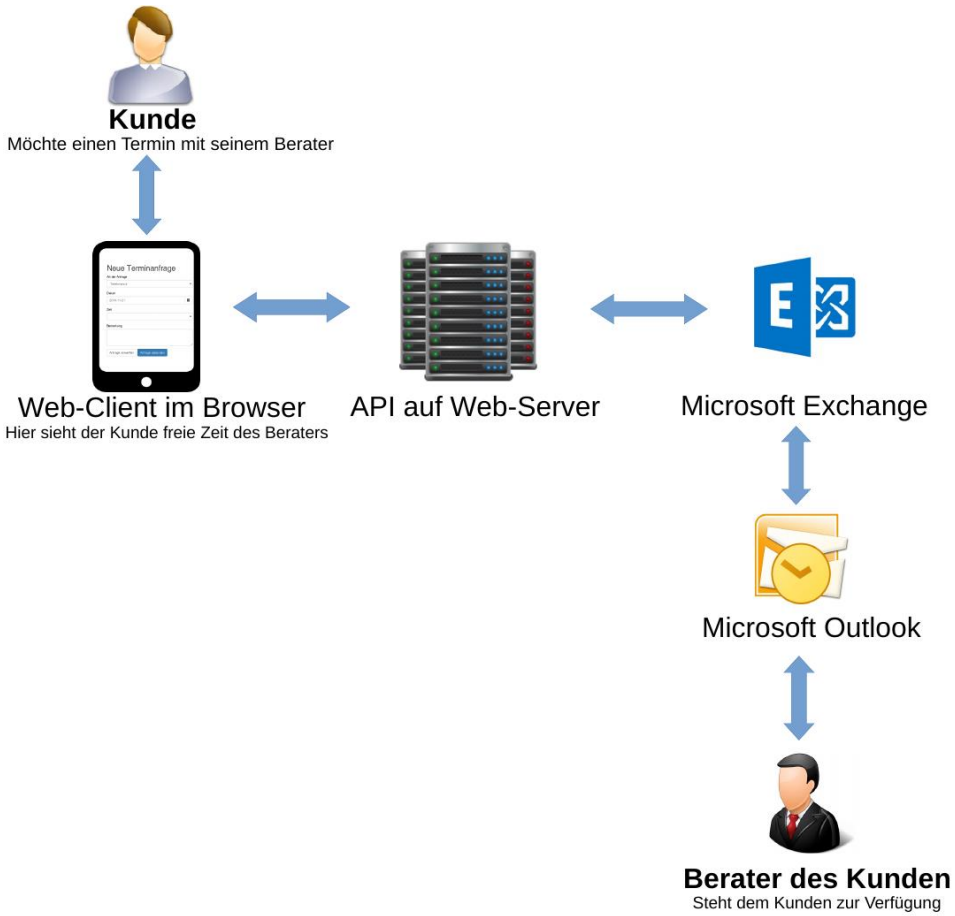


Abbildung 6-1: Kontextdiagramm

6.1.2 Deployment Diagramm

Folgendes Deployment Diagramm zeigt eine Grobübersicht über die Komponenten, die implementiert werden.

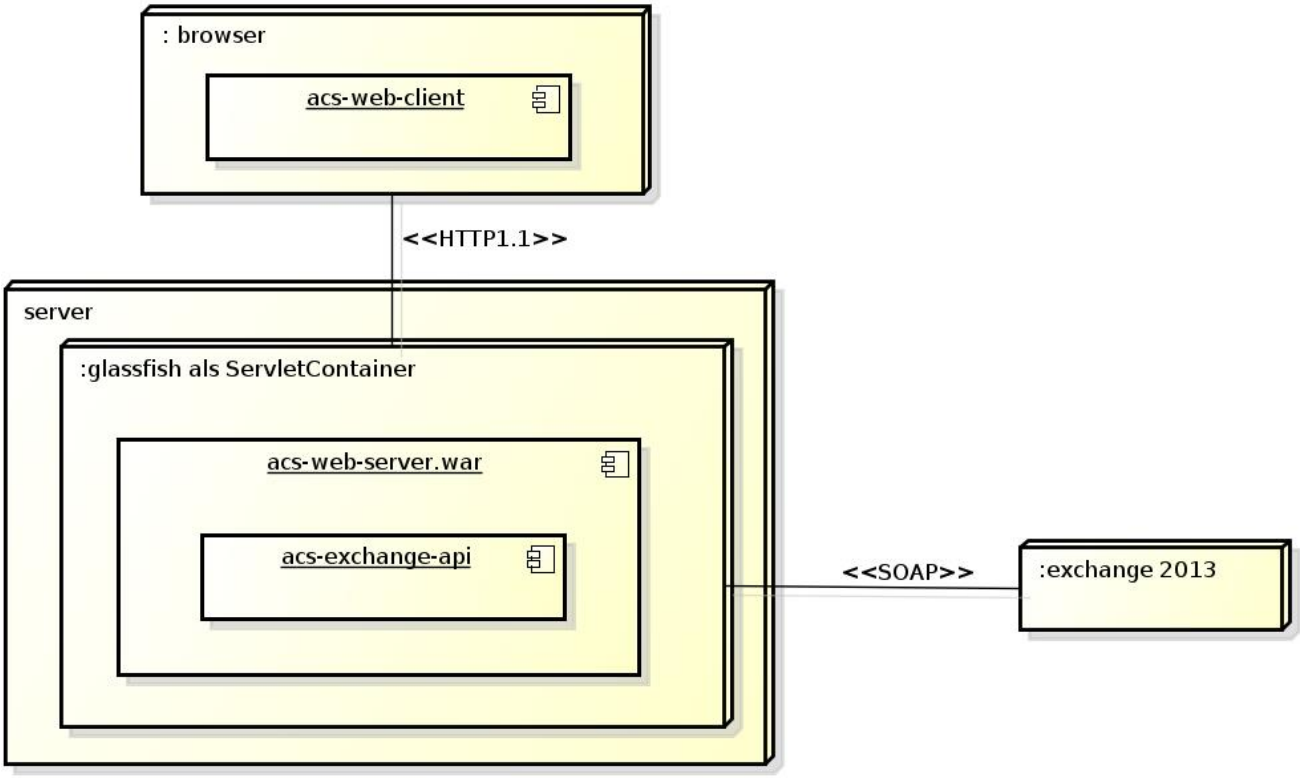


Abbildung 6-2: Deployment Diagramm

Der **acs-web-client** läuft im Browser und spricht über HTTP die REST Schnittstelle des **acs-web-server** an. Dieser beinhaltet das **acs-exchange-api**, welches verwendet wird um per SOAP mit dem Exchange-Server zu kommunizieren.

Nachfolgend wird auf den Aufbau der einzelnen Komponenten eingegangen.

6.2 Komponenten

Das Komponentendiagramm zeigt eine Übersicht über die einzelnen Komponenten. Jede der grün eingefärbten Komponenten wird im Rahmen der Studienarbeit implementiert. Die restlichen sind verwendete Komponenten oder optionale Implementierungen.

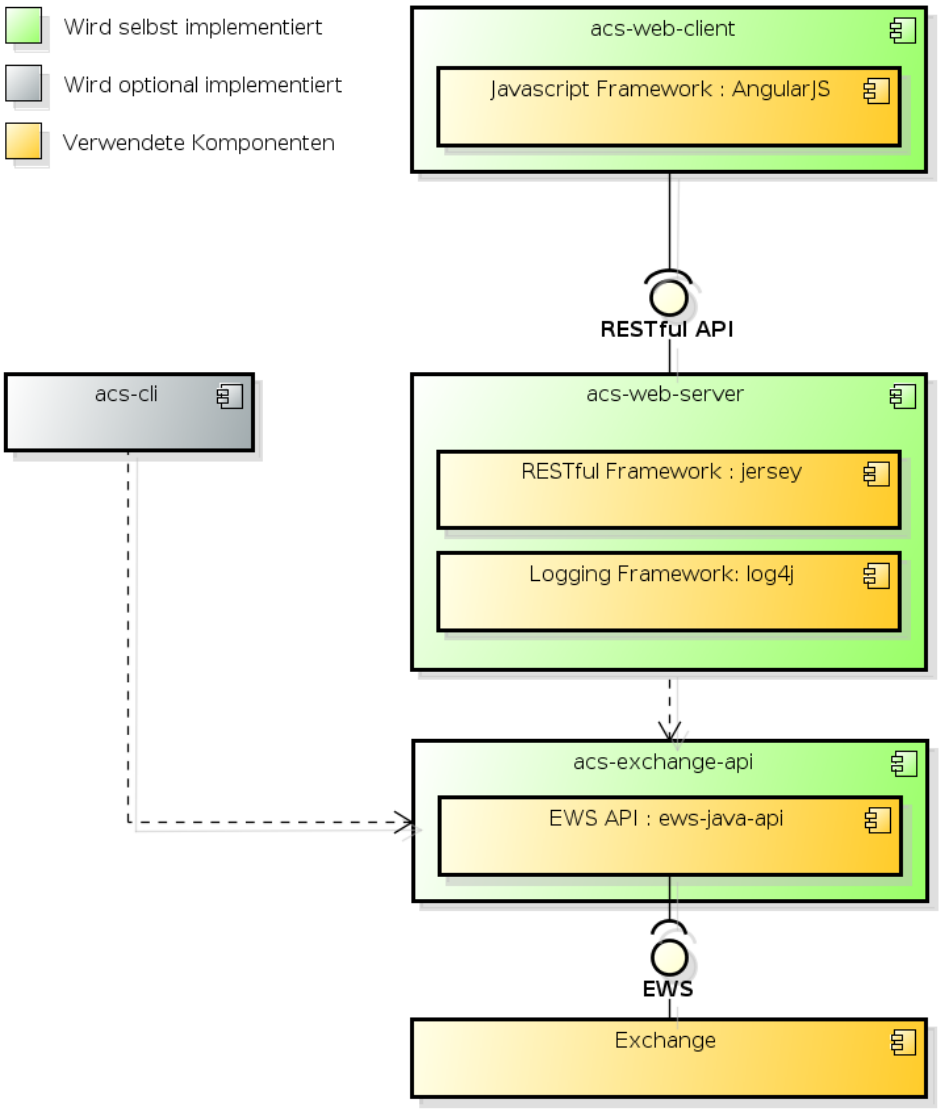


Abbildung 6-3: Komponenten Diagramm

Wie man aus dem Komponentendiagramm entnehmen kann, wird der Exchange-Server über das ews-java-api angesprochen. Als Demo zur Verwendung des acs-exchange-api wird ein Web-Server mit einer REST-Schnittstelle implementiert, welcher mittels Web-Client angesprochen werden kann.

Nachfolgend werden die einzelnen Komponenten beschrieben, die im Rahmen der Semesterarbeit implementiert werden. Dabei wird das **acs-exchange-api** am detailliertesten behandelt, da es sich um das für den Auftraggeber wichtige Endprodukt handelt.

6.2.1 Komponente: acs-exchange-api

6.2.1.1 Package Übersicht

Nachfolgend eine Übersicht über die Packages der Komponente acs-exchange-api:

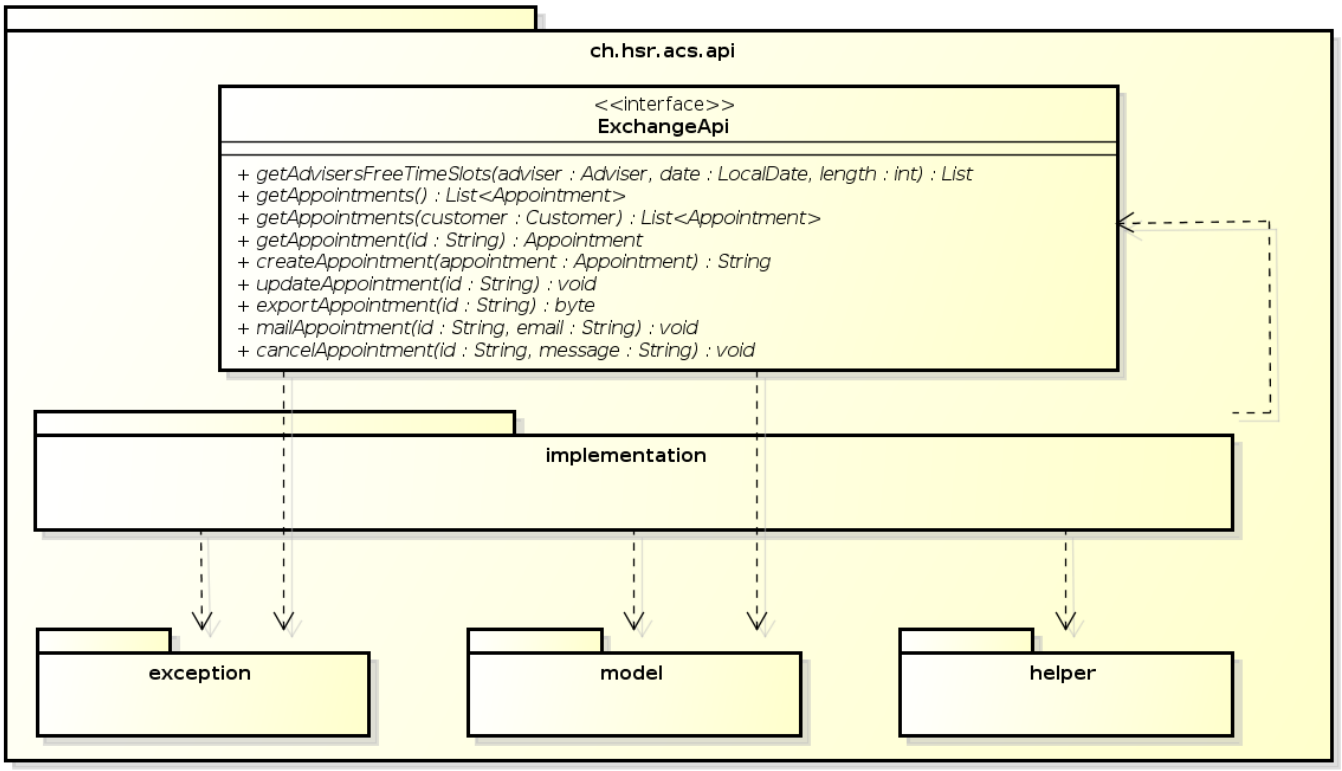


Abbildung 6-4: Package Übersicht

Package	Beschreibung
implementation	Beinhaltet die Implementation des ExchangeApi
exception	Beinhaltet alle Exceptions
model	Beinhaltet die Abbildung des Domain Models
helper	Beinhaltet Hilfsklassen

Tabelle 6-1: Beschreibung Package Übersicht

Nachfolgend werden die Packages mit Erklärungsbedarf genauer beschrieben.

6.2.1.2 Package: Implementation

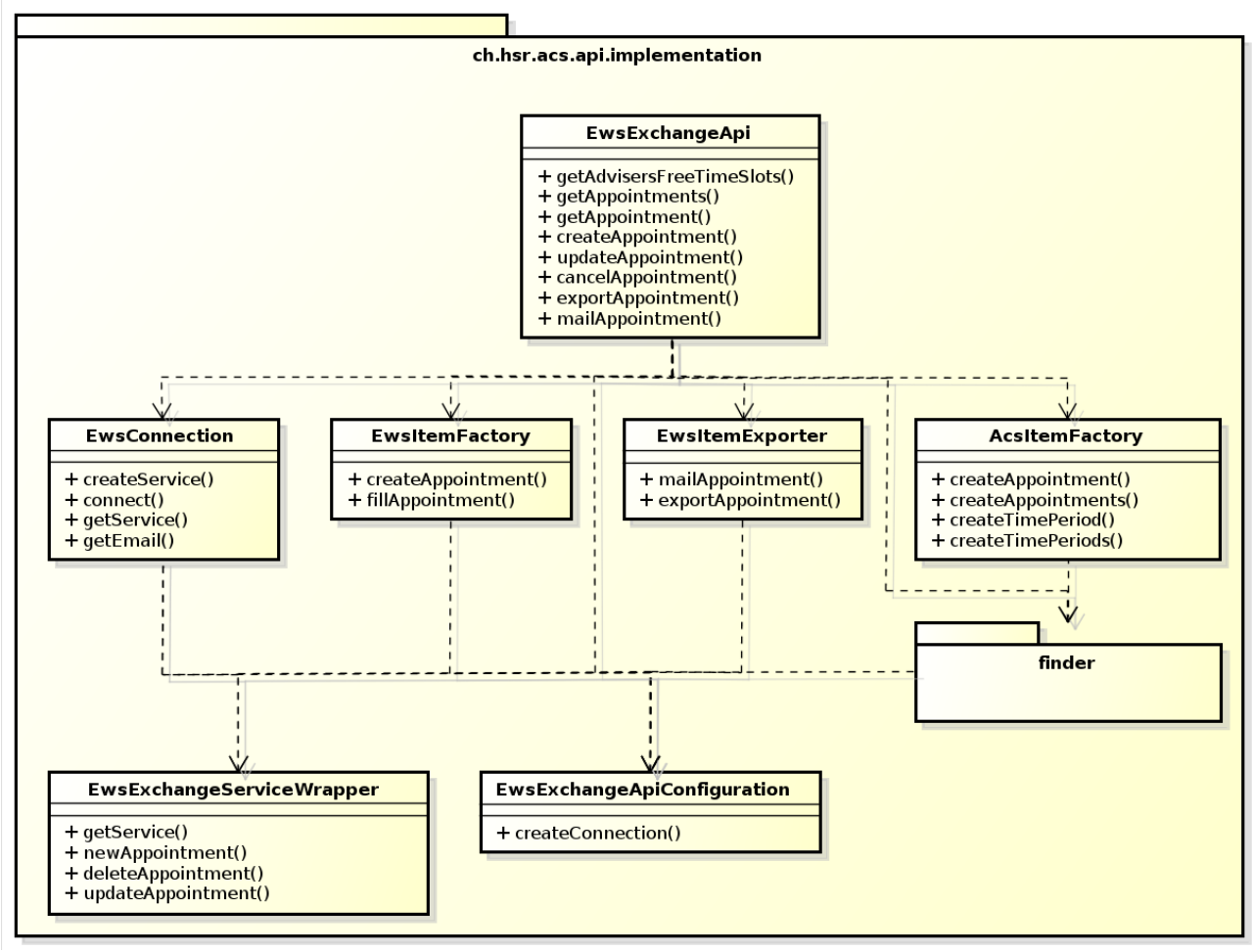


Abbildung 6-5: Package Implementation

Terminologie:
Acs-Objekte: Objekte basierend auf Klassen aus dem Domain Model (z.B. Appointment).
Ews-Objekte: Objekte verwendet von der EWS-Schnittstelle (z.B. EwsAppointment).

Alle Ews-Klassen haben Abhängigkeiten vom EwsExchangeServiceWrapper und der EwsExchangeApiConfiguration. Diese werden zur Kommunikation mit dem ews-java-api verwendet.

Das finder-Package beinhaltet alle Klassen die zum Finden von Ews-Objekten verwendet werden.

Nachfolgend werden die einzelnen Klassen und Packages genauer beschrieben.

6.2.1.2.1 Klasse: EwsExchangeServiceWrapper

Der EwsExchangeServiceWrapper hält eine Instanz der Klasse ExchangeApi, welche vom ews-java-api kommt. Dieser bietet lediglich die Methoden an, die die Klasse ExchangeApi auch anbietet. Benötigt wird er, da das ExchangeApi keine spezifischen und damit unbrauchbare Exceptions wirft. Deswegen werden alle Methodenaufrufe in try-catch Blöcke eingepackt. Falls eine Exception vom ews-java-api auftritt, wird diese in eine EwsServiceException gepackt.

Folgender Ausschnitt aus dem EwsExchangeServiceWrapper zeigt dieses Vorgehen:

```
/**
* Wrapper around ExchangeService from ews-java-api.<br/>
* Necessary to handle Exceptions thrown by ExchangeService.
*/
public class EwsExchangeServiceWrapper {
  private final ExchangeService service;


  public EwsExchangeServiceWrapper(ExchangeService service) {
     this.service = service;
  }


  public ExchangeService getService() {
     return service;
  }


  public FindItemsResults<Item> findItems(WellKnownFolderName calendar, SearchFilter filter,
ItemView view) {
     try {
        return service.findItems(calendar, filter, view);
     } catch (Exception e) {
        throw new EwsServiceException(e);
     }
  }


  public void loadPropertyForItem(Item item, PropertyDefinition propertyDefinition) {
     try {
        item.load(new PropertySet(propertyDefinition));
     } catch (Exception e) {
        throw new EwsServiceException(e);
     }
  }
...
```

Code Snippet 6-1: EwsExchangeServiceWrapper

6.2.1.2.2 Klasse: EwsExchangeApiConfiguration

Die Klasse EwsExchangeApiConfiguration beinhaltet die Konfiguration des APIs und muss diesem zur Instanziierung übergeben werden.

6.2.1.2.3 Klasse: EwsConnection

Mit einer Instanz der EwsConnection wird die Verbindung zum Exchange Service aufgebaut.

Abbildung 6-6: Klasse EwsConnection

Methode	Beschreibung
connect	Stellt die Verbindung mit den Daten, die im Konstruktor übergeben wurden (siehe Kapitel API Initialisierung) her. Wenn keine Verbindung hergestellt werden kann, wird eine ExchangeConnectionException geworfen.

Tabelle 6-2: Klasse EwsConnection Beschreibung

Nachfolgendes Sequenzdiagramm zeigt die Verwendung der EwsConnection im Zusammenspiel mit dem ews-exchange-api. Die erstellte Instanz wird dem Api übergeben, welches die connect Methode aufruft. Mit diesem Aufruf wird der EwsExchangeServiceWrapper instanziiert welche eine Instanz vom ExchangeService beinhaltet. Dieser kommt vom ews-java-api und wird für die Kommunikation mit dem Exchange-Server verwendet.

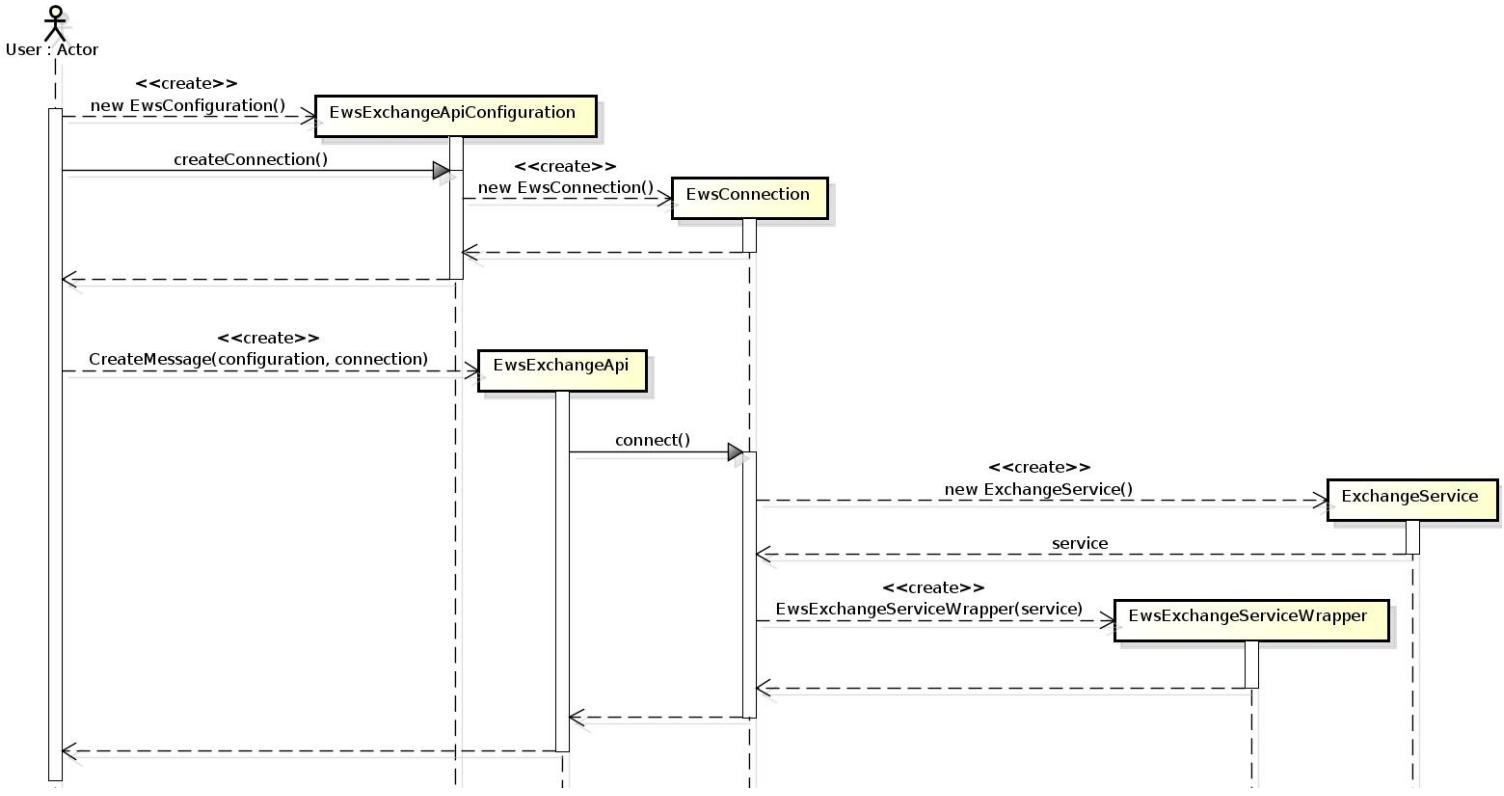


Abbildung 6-7: SSD Connection

6.2.1.2.4 Klasse: EwsItemFactory

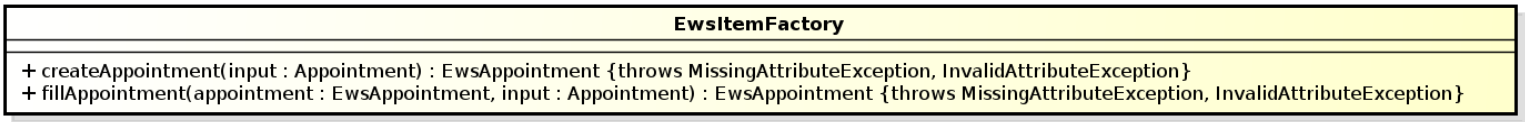


Abbildung 6-8: Klasse EwsItemFactory

Mit der EwsItemFactory werden Ews-Objekte basierend auf Acs-Objekten erstellt.

Methode	Beschreibung
createAppointment	Erstellt ein EwsAppointment mit den Werten des Appointments, das als Parameter übergeben wird. Bei einem fehlerhaften Input werden MissingAttribute- oder InvalidAttribute- Exceptions geworfen. Achtung: Das EwsAppointment welches erstellt wird, ist noch nicht persistiert. Dies erfolgt ausserhalb der EwsItemFactory.
fillAppointment	Füllt das übergebene EwsAppointment mit den Werten des übergebenen Appointments. Bei einem fehlerhaften Input werden MissingAttribute- oder InvalidAttribute- Exceptions geworfen.

Tabelle 6-3: Klasse EwsItemFactory Beschreibung

Nachfolgendes Sequenzdiagramm zeigt die Verwendung der EwsItemFactory, die über das EwsExchangeApi aufgerufen wird.

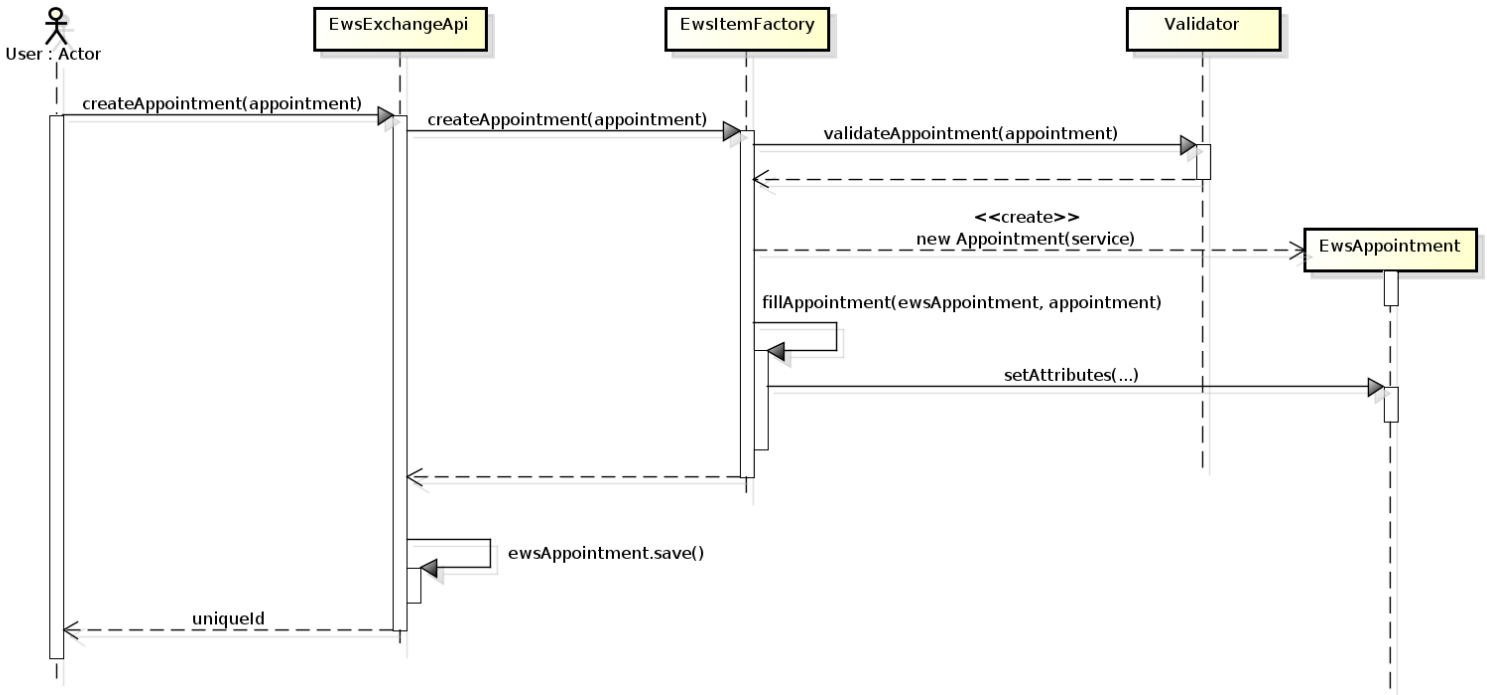


Abbildung 6-9: SSD Appointment erstellen

6.2.1.2.5 Klasse: AcsItemFactory

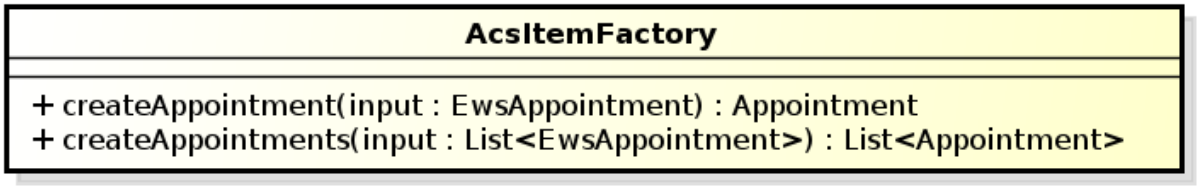


Abbildung 6-10: Klasse AcsItemFactory

Erstellt Acs-Appointments basierend auf den übergebenen EwsAppointments.

Methode	Beschreibung
createAppointment	Gibt ein Appointment mit den Werten des übergebenen EwsAppointments zurück.
createAppointments	Gibt eine Liste von Appointments mit den Werten der übergebenen EwsAppointments zurück.

Tabelle 6-4: Klasse AcsItemFactory Beschreibung

Die Verwendung der AcsItemFactory wird in folgendem Sequenzdiagramm beim Aufruf der getAppointment(String uniqueId) Methode gezeigt.

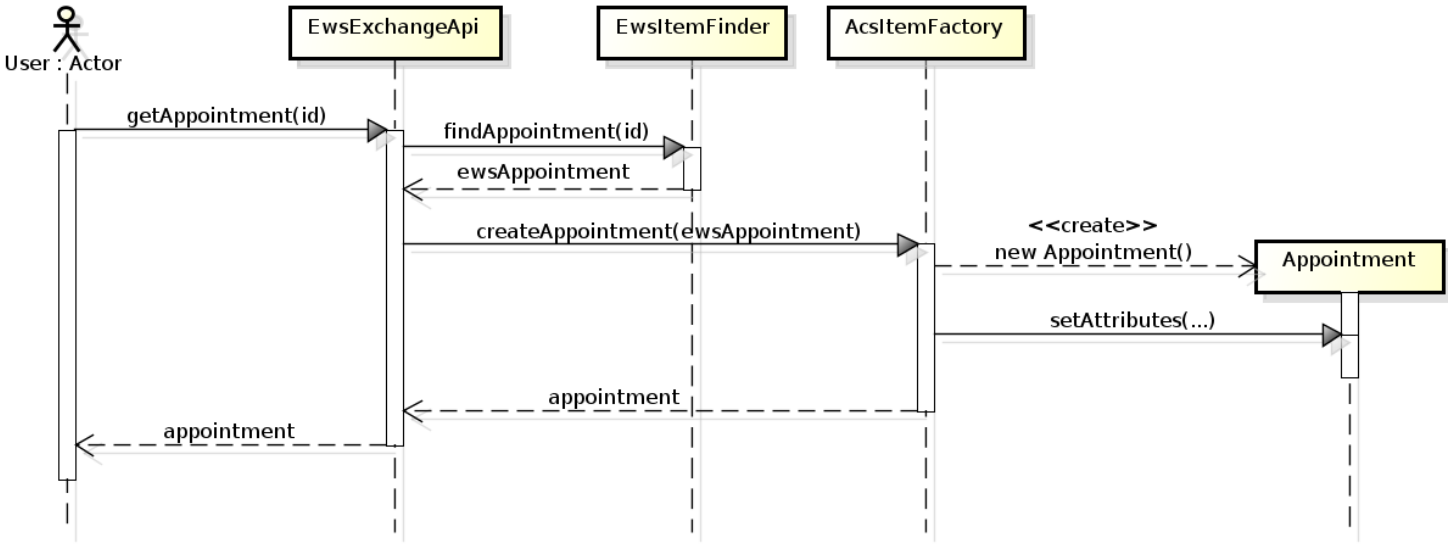


Abbildung 6-11: SSD finde Appointment

6.2.1.3 Package: Implementation.Finder

Mit den Findern werden Objekte vom Exchange-Server geholt.
Die Finder-Klassen werden auf mehrere Unterklassen aufgeteilt und in einem Finder-Package gruppiert, damit nicht zu grosse Klassen entstehen.

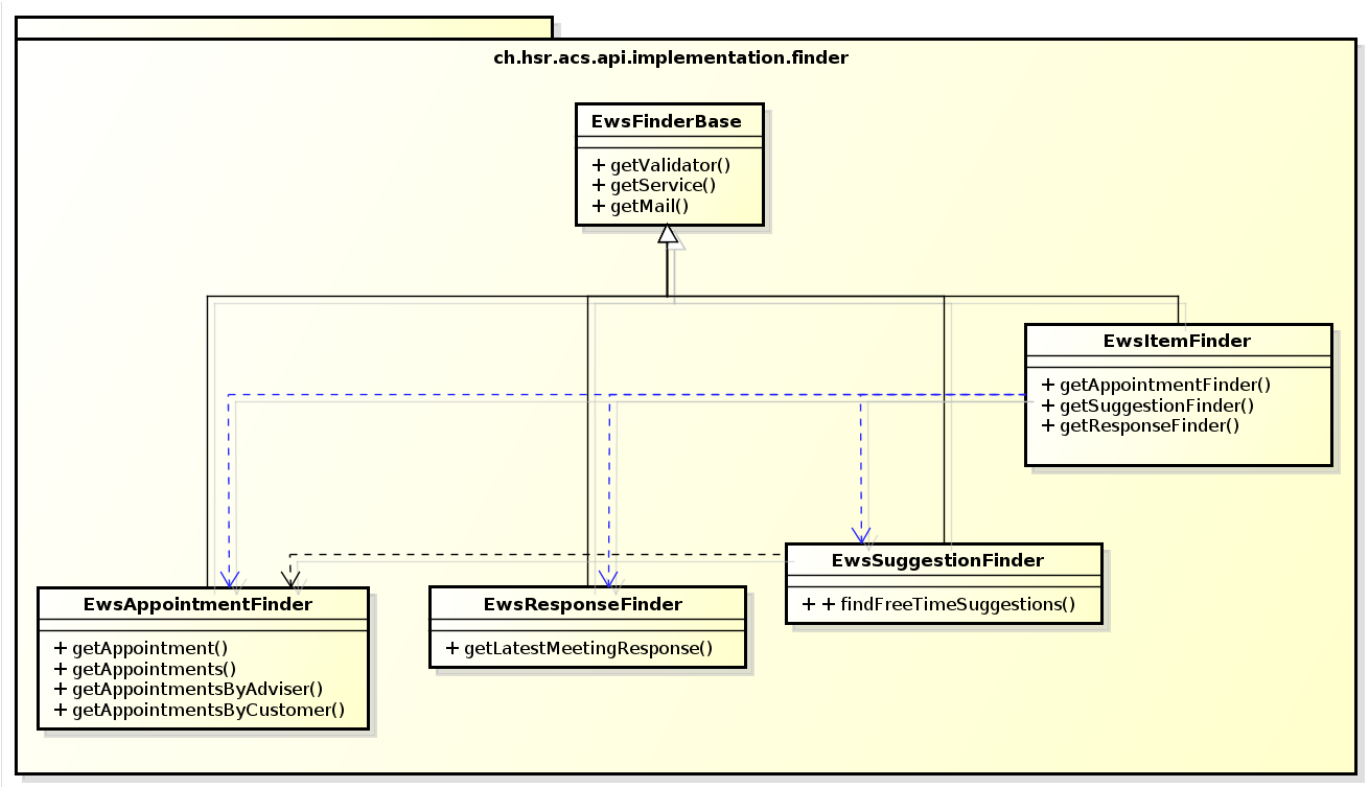


Abbildung 6-12: Package finder

Alle Finder erben von der Klasse EwsFinderBase, in der die Instanzvariablen definiert sind, die von allen Findern benötigt werden.

Nachfolgend werden die alle Finder genauer beschrieben.

6.2.1.3.1 Klasse: EwsItemFinder

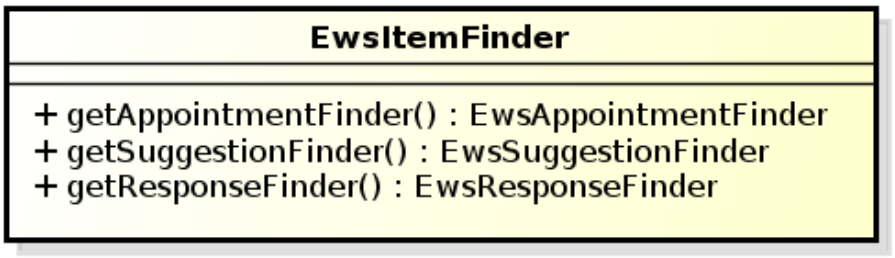


Abbildung 6-13: Klasse EwsItemFinder

Mit dem EwsItemFinder werden Ews-Objekte nach übergebenen Kriterien gefunden. Für jeden Typ von Ews-Objekten gibt es einen eigenen Finder:

- EwsAppointmentFinder
- EwsResponseFinder
- EwsSuggestionFinder

Um einen dieser Finder zu verwenden muss lediglich der EwsItemFinder instanziert werden. Welchen man dann beispielsweise wie folgt verwenden kann:

```
ewsItemfinder.getAppointmentFinder().getAppointment(id)
```

Code Snippet 6-2: Verwendung EwsItemFinder

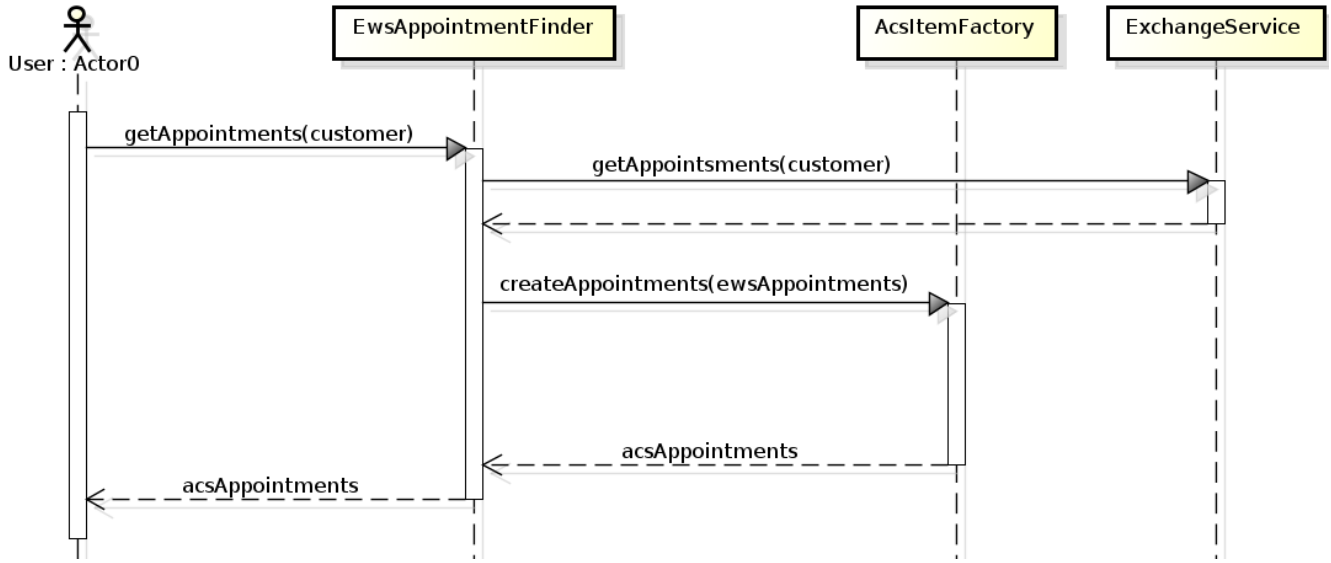
6.2.1.3.2 Klasse: EwsAppointmentFinder

Abbildung 6-14: Klasse EwsAppointmentFinder

Methode	Beschreibung
getAppointment	Gibt das EwsAppointment mit der übergebenen UniqueId zurück. Kann das EwsAppointment nicht gefunden werden, wird eine ItemNotFoundException geworfen.
getAppointments	Gibt eine Liste aller EwsAppointments zurück
getAppointmentByAdivser	Gibt eine Liste aller EwsAppointments des übergebenen Adviser zurück
getAppointmentByCustomer	Gibt eine Liste aller EwsAppointments des übergebenenKunden zurück

Tabelle 6-5: Klasse EwsAppointmentFinder Beschreibung

Folgendes Sequenzdiagramm zeigt das Zusammenspiel zwischen dem EwsAppointmentFinder und der AcsItemFactory. Die EwsAppointments werden vom ExchangeService geholt und mit der AcsItemFactory in AcsObjekte abgefüllt.



6.2.1.3.3 Klasse: EwsResponseFinder

Abbildung 6-15: Klasse EwsResponseFinder

Methode	Beschreibung
getLatestMeetingResponse	Gibt die letzte Antwort des übergebenen EwsAppointments zurück. Falls noch keine Antwort vorhanden ist, wird null zurückgegeben.

Tabelle 6-6: Klasse EwsResponseFinder Beschreibung

6.2.1.3.4 Klasse: EwsSuggestionFinder

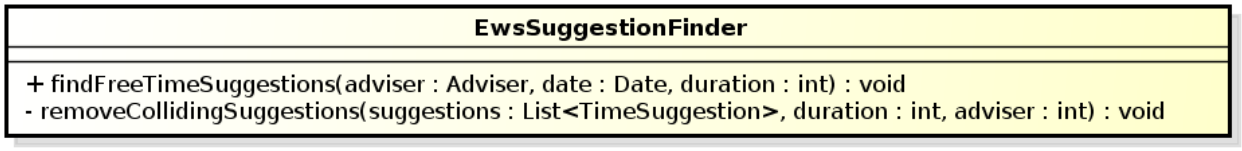


Abbildung 6-16: Klasse EwsSuggestionFinder

Methode	Beschreibung
findFreeTimeSuggestions	Gibt eine Liste von EwsTimeSuggestions zurück. adviser legt den betreffenden Berater fest. Mit date wird der Tag übergeben. Die duration legt die Zeitdauer der Vorschläge in Minuten fest
removeCollidingSuggestions	Entfernt die bereits im acs-exchange-api erfassten Terminen, kollidierenden Vorschläge.

Tabelle 6-7: Klasse EwsSuggestionFinder Beschreibung

6.2.2 Komponente: acs-web-server

6.2.2.1 Beschreibung

Der Web-Server bietet eine REST-Schnittstelle zur Verwendung des APIs. Implementiert wird dieser mit dem Java-Framework jersey, mit welchem komfortabel durch Annotationen definiert werden kann, welche Methoden durch welche URLs aufgerufen werden. Übertragene Objekte werden automatisch von und zu JSON deserialisiert bzw. serialisiert.

Der Server beinhaltet kaum Logik und leitet lediglich die HTTP-Requests an die betreffenden API Methoden weiter. Zusätzlich werden die Exceptions, die das API wirft zu den jeweiligen HTTP Status Codes gemappt.

6.2.2.2 Beispiel Aufruf im Browser

Folgender Aufruf soll alle Meetings vom Exchange abrufen

Request

http://localhost:8080/acs-web-server/appointment

Server Antwort


```
[{
	"customer": {
		"hash":
		"1fe73c15f029d5ffaa9b84d0b09b0d9c56cd739e48d25aabbc3f4e1dc9d586eb2a848e83c
		0223a12f52bb562edd674fe8cdc9c93b11534684440e58d1d5ef17d"
	},
	"end": "2014-12-10T11:30:00+01:00",
	"id":
	"AAMkAGNiN2VjZGMzLTIxM2QtNGZkOC1hZDdiLWNjZTU0ODIwYjRiMwBGAAAAAADD2qTd4zLcQo/KOPCz1
	MSsBwDVBo64uz6AS51M8cz3r9HMAAAAAAEOAADVBo64uz6AS51M8cz3r9HMAAAUfYqeAAA=",
	"response": {
		"message": "",
		"type": "UNKNOWN"
	},
	"start": "2014-12-10T11:00:00+01:00",
	"text": "Co-Browsing",
	"type": "CO_BROWSING"
}
]
```


Code Snippet 6-3: JSON

6.2.2.3 Exception Mapping

Folgendes Exception Mapping bietet sich für die Exceptions an, die das **acs-exchange-api** wirft.

Exception	HTTP Status Code
ExchangeConnectionException	503 Service Unavailable
InvalidAttributeException	422 Unprocessable Entity
ItemNotFoundException	404 Not Found
MissingAttributeException	400 Bad Request
AuthenticationFailedException	401 Unauthorized
ForbiddenActionException	400 Bad Request
NotAuthenticatedException	401 Unauthorized

Tabelle 6-8: Exception Mapping

Die restlichen Exceptions können auf “500 Internal Server Error” gemappt werden.

6.2.2.4 Package-Übersicht

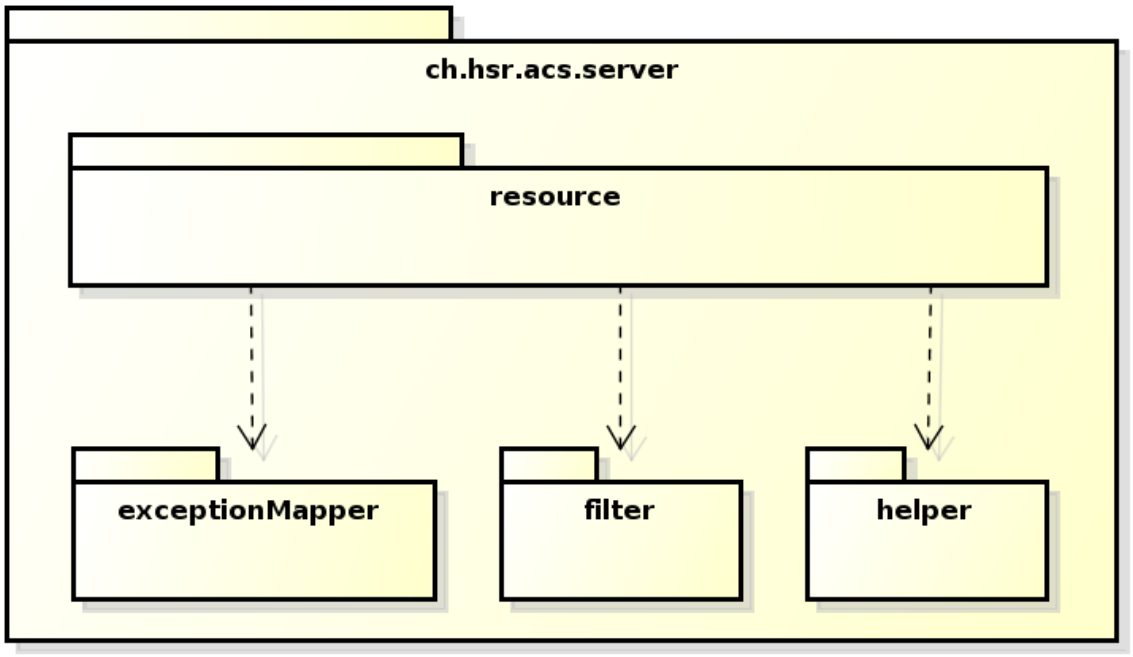


Abbildung 6-17: Package Webserver

Die Packages sind nachfolgend beschrieben.

6.2.2.5 Package: Helper

Das “helper”-Package enthält Hilfsklassen die von allen Resourcen verwendet werden können (Beispielsweise DateHelper).

6.2.2.6 Package: ExceptionMapper

Das “exceptionMapper”-Package enthält das jersey-Mapping zwischen den Exceptions und den HTTP Status Codes.

Beispielsweise wird die API-Exception ItemNotFoundException wie folgt auf den HTTP Status Code 404 gemappt.

```
@Provider
public class ItemNotFoundExceptionMapper implements ExceptionMapper<ItemNotFoundException> {
  public Response toResponse(final ItemNotFoundException ex) {
     return Response.status(Status.NOT_FOUND).build();
  }
}
```

Code Snippet 6-4: ExceptionMapper

Durch die Annotation @Provider erkennt jersey das Mapping ohne weitere Konfiguration.

6.2.2.7 Resource-Package (Schnittstelle)

Der Server bietet eine REST-Schnittstelle, über die er angesprochen werden kann. Nach der Jersey-Konvention befinden sich die aufrufbaren Methoden in Klassen im Package "resource". Für jede Resource gibt es eine eigene Resource Klasse (z.B. AppointmentResource für die Operationen betreffend der Termine). Daraus ergeben sich folgende Resource-Klassen:

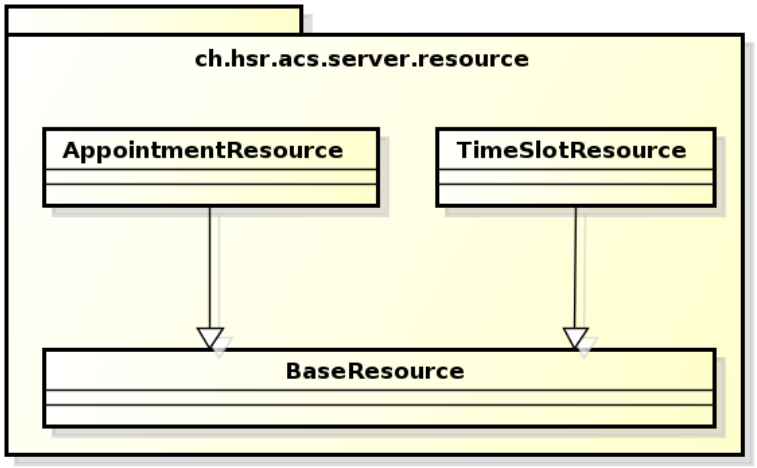


Abbildung 6-18: Package Webserver Resource

Das Mapping der URLs auf die passenden Methoden wird vom jersey-Framework übernommen. Dafür werden die Methoden mit den Standard Java Annotationen für RESTful Services aus dem Package "javax.ws.rs" annotiert.

Beispielsweise wird die Klasse AppointmentResource wie folgt annotiert:

```
@Path("/appointment")
public class AppointmentRessource extends BaseRessource {


  @GET
  @Produces(MediaType.APPLICATION_JSON)
  public List<Appointment> get() {
     return getApi().getAppointments();
  }


  @POST
  @Consumes(MediaType.APPLICATION_JSON)
  public void create(final JAXBElement<Appointment> appointment) {
     getApi().createAppointment((Appointment) appointment.getValue());
  }
…
}
```

Code Snippet 6-5: jersey routing

Damit ist die Methode 'get' mit einem GET-Request, sowie die Methode 'create' mit einem POST-Request auf folgende URL aufrufbar:
https://{serverurl}:8080/acs-server/appointment/

6.2.3 Komponente: acs-web-client

Mit dem web-client wird die Verwendung des ews-exchange-api demonstriert. Hierbei handelt es sich um einen Web-Client welcher mit Java Script implementiert ist.

Angular JS

Angular JS ist ein Open-source Javascript-Framework von Google. Angular JS vereinfacht die Java Script Programmierung deutlich.

Abbildung 6-19: AngularJS

Quelle: https://angularjs.org/

Model View Controller

Angular JS basiert auf einem Model View Controller Prinzip.

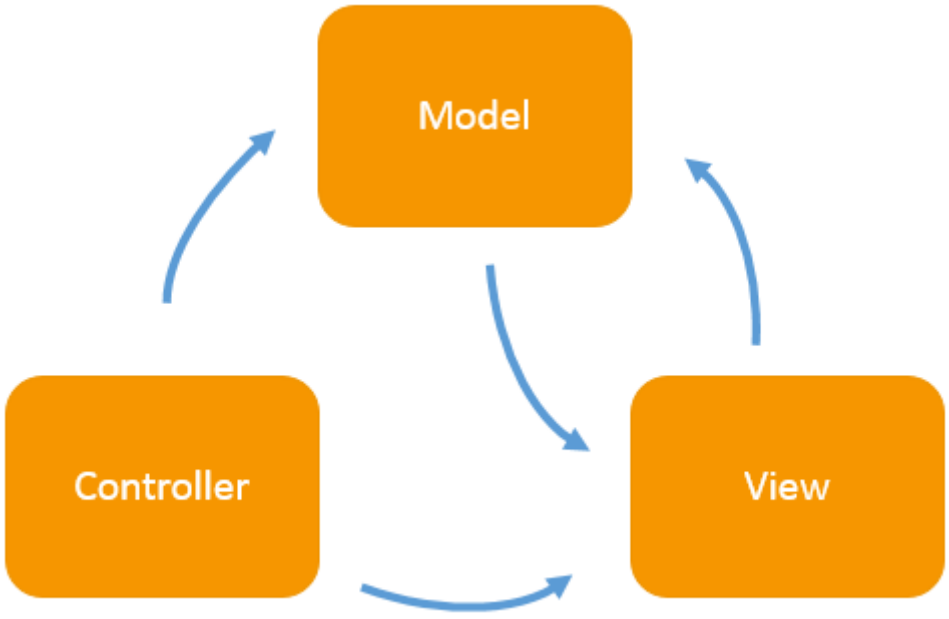


Abbildung 6-20: Model View Controller

Der Client interagiert ausschliesslich mit der View. Klickt der Benutzer einen Button, so reagiert der Controller mit einer entsprechenden Aktion.
Beispiel:
User klickt den Login-Button, so prüft der Controller die Eingaben. Ist der Benutzer erfolgreich autorisiert worden, so leitet der Controller den Benutzer auf die nächste View bzw. Seite.

Datei Struktur

Angular arbeitet nicht nach ‘configuration over convention’ Prinzip. Das heisst, die Projektstruktur ist nicht vorgegeben.
Wir haben uns für eine „Sort by Type“ Struktur entschieden. Somit liegen innerhalb des „app“ Verzeichnisses jeweils Controller, Views, Fonts, Styles etc. in einem separaten Ordner. Der Vorteil liegt darin, dass man schnell einen Überblick über das Projekt erhält. Wird das Projekt jedoch grösser und komplexer, so empfiehlt es sich die Guidelines von Google anzuschauen.

Abbildung 6-21: Client Datei Struktur

Module

Eine starke Vereinfachung bieten die Module in Angular. Ein Modul ist eine Sammlung von services, directives, controller und Filter.

Services

Nachfolgende Services sind wichtige Bestandteile des acs-web-client

$ressource

Das Modul $ressource bieten eine bequeme Art REST Schnittstellen anzusprechen.

Mit $ressource bietet ein einfaches Handling um eine REST Schnittstelle anzusprechen.

Folgendes Beispiel zeigt, wie man eine Liste von Appointments entgegen nimmt.

1. Factory mit Namen und Pfad zur Schnittstelle.

```
 .factory("Appointment", function($resource) {
       return $resource(base_url + "acs-web-server/appointment/:id",{id:'@id'});
 })
```

Code Snippet 6-6: AngularJS Factory Definition

2. In jedem verwendeten Controller muss die Factory übergeben werden

```
angular.module('acsWebClientApp')
 .controller('ListAppointmentsCtrl', function ($rootScope, $scope, $location,
GetAppointmentByCustomerHash)
```

Code Snippet 6-7: AngularJS Factory Controller

3. Verwendung im Controller

```
$scope.appointments = Appointment.query();
```

Code Snippet 6-8: AngularJS Factory Verwendung

Neben query() können auch die folgenden REST Verben verwendet werden

```
{
 'get':   {method:'GET'},
 'save':   {method:'POST'},
 'query':  {method:'GET', isArray:true},
 'remove': {method:'DELETE'},
 'delete': {method:'DELETE'}
}
```

Code Snippet 6-9: AngularJS Factory Verben

$location

Location bietet eine einfache Möglichkeit eine URL zu parsen oder Redirects auszuführen.

```
// aktuelles Verzeichnis auslesen
$location.path();

// nach /appointment wechseln
$location.path('/appointment)
```

Code Snippet 6-10: AngularJS Redirect mit $location

Controller

Eine View besitzt ein Controller, welcher die Bussiness Logic implementiert. Damit der Code im Controller nicht zu gross wird, besitzt jede View einen eigenen Controller.

Filter

Filter bringen die Möglichkeit, Daten die dem User dargestellt werden zu formatieren
Mit folgenden Beispielfilter wird aus “2014-11-21T19:30:00+01:00” →2014-11-21

```
$filter('date')($scope.appointment.date, "yyyy-MM-dd")
```

Code Snippet 6-11: AngularJS Filter

Direktiven

Direktiven sind vordefinierte oder auch selbst implementierte Tags, welche sich bequem in den HTML-Code einbinden lassen. Vordefinierte direktiven sind mit “ng-name” gekennzeichnet.

Nachfolgend wird die Verwendung eines ng-repeat und ng-switch Tags demonstriert.
In diesem Beispiel wird mit ng-repeat über eine Liste von Appointments iteriert und je nach Status wird mit ng-switch der zutreffende Status in der View angezeigt.

```
<tr ng-repeat="appointment in appointments">
<td>
  <div ng-switch on="appointment.response.type">
        <div ng-switch-default>
        <span class="label label-warning">Offen</span>
        </div>
        <div ng-switch-when='ACCEPTED'>
        <span class="label label-success">Angenommen</span>
        </div>
        <div ng-switch-when='DECLINED'>
        <span class="label label-danger">Abgelehnt</span>
        </div>
  </div>
</td>
```

Code Snippet 6-12: AngularJS Direktiven

Mockups und Resultate

Die Mockups wurden schon im ersten Meilenstein “Elaboration-Spezifikation” erstellt. Der Vorteil liegt darin, dass sich der Auftraggeber schon sehr früh ein Bild des Clients machen kann. Die Mockups wurden in Balsamiq erstellt, was uns die Möglichkeit gab die Mockups “clickable” zu gestalten, welche somit sehr realitätsgetreu waren. Die Crealogix war mit dem Resultat sehr zufrieden und da der Client nur für Demonstrations-Zwecke verwendet wurde, haben wir auf weitere Usability-Tests verzichtet. Zudem konnten wir uns in der Implementierung an den vorhandenen Mockups richten, was die Arbeit erleichterte.

Das erstellte GUI und die Mockups unterscheiden sich nur wenig, doch stellt man folgende Unterschiede fest:

- Das GUI hat gegenüber den Mockups kein Menü im Header
- Beim GUI kam zusätzlich die optionale Anforderung für “Termin exportieren dazu”.

Nachfolgend sind die Mockups mit dem finalen GUI gegenübergestellt

Termine auflisten

Der Benutzer soll alle Termine, deren Status, Inhalte und weitere Aktionen einfach auf einer Seite sehen können. Dafür wurde eine simple Tabelle erstellt.

Mockup:

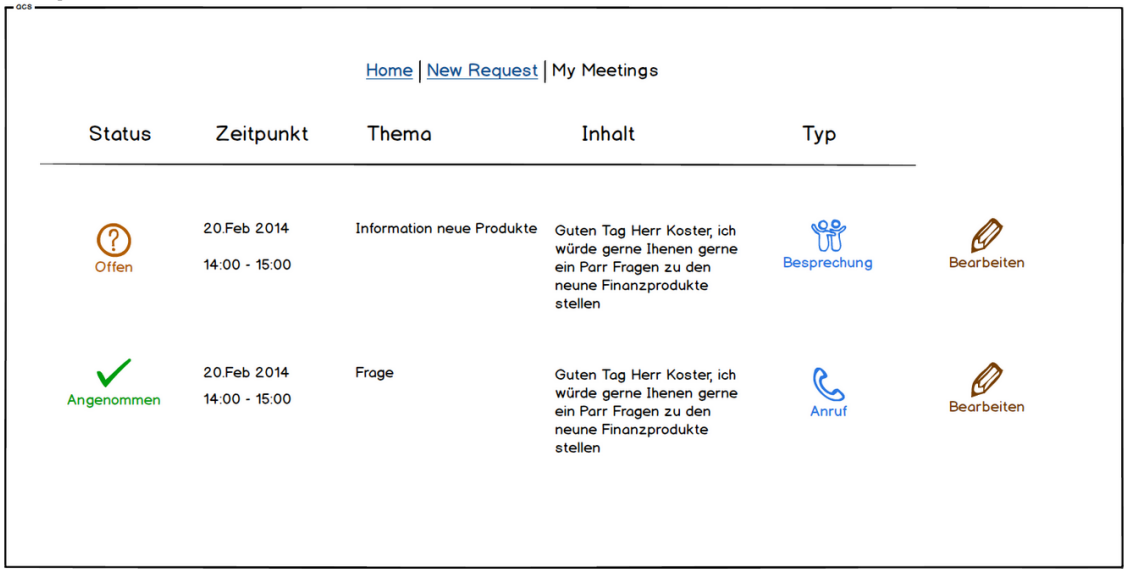


Abbildung 6-22:Client Mockup Termine auflisten

Finales GUI:

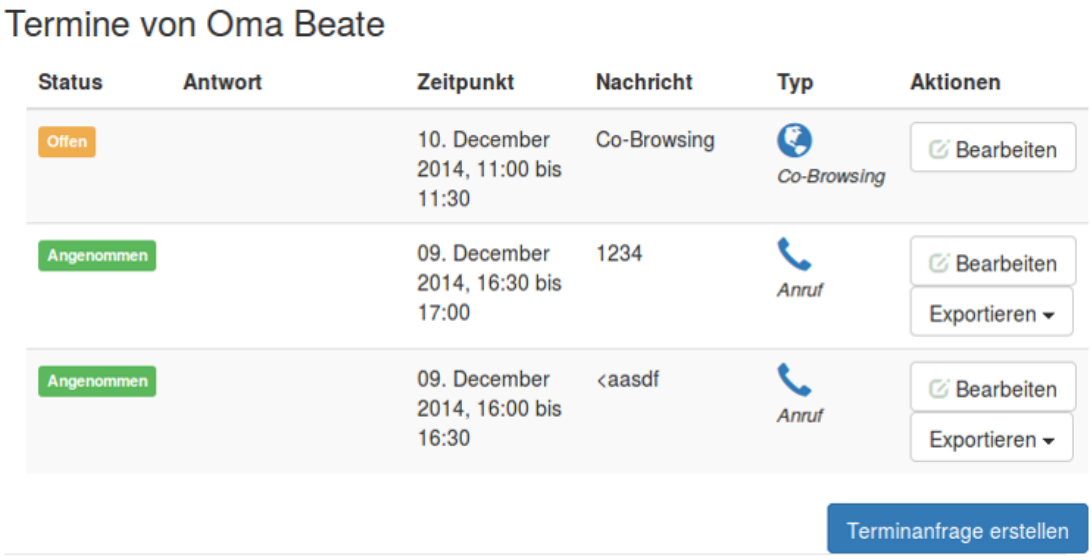


Abbildung 6-23: Client GUI Termine auflisten

Neuer Termin erstellen

Für eine Termin-Erstellung wird dem Benutzer ein einfaches Formular verwendet.

Mockup:

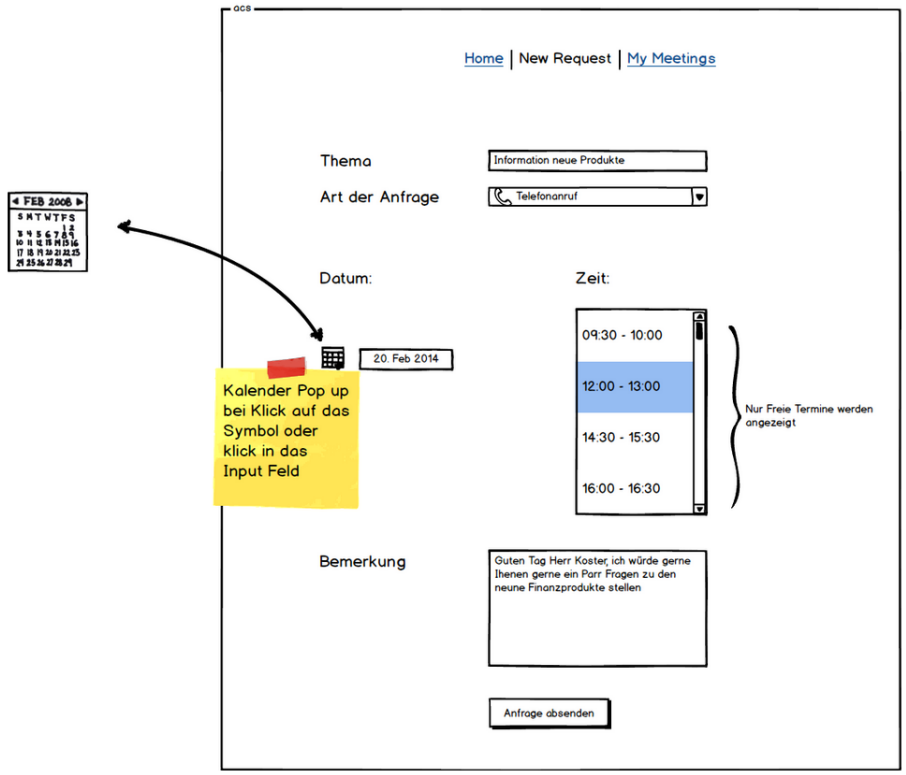


Abbildung 6-24: Client Mockup neuer Termin

Finales GUI:

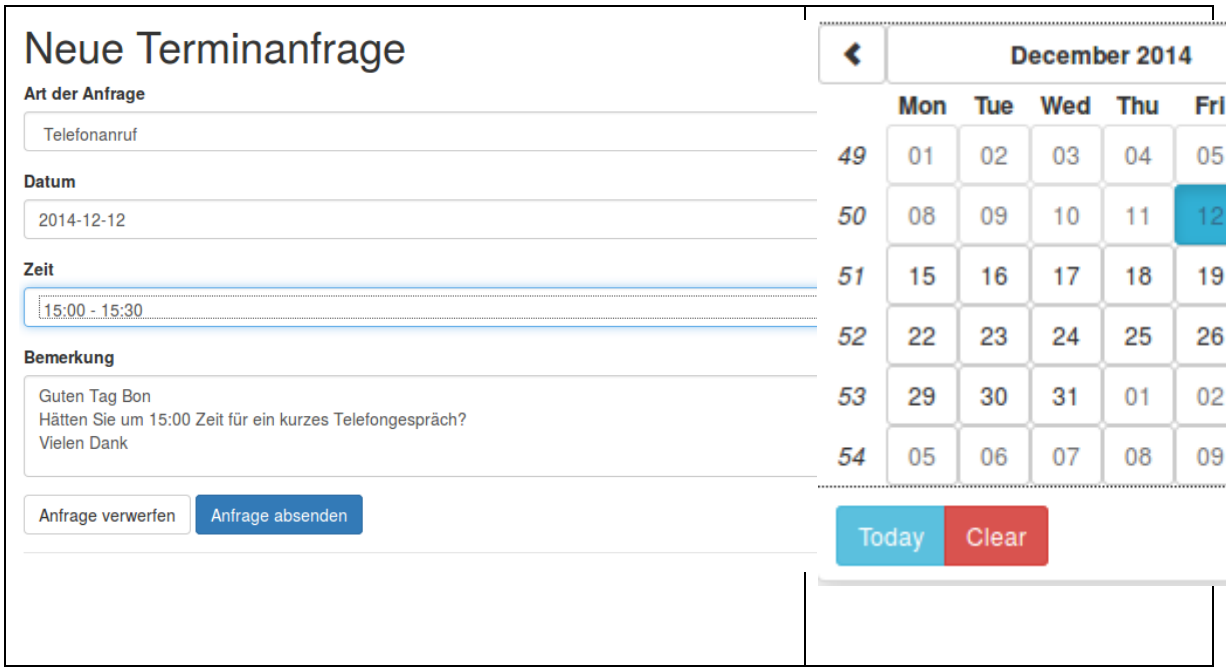


Abbildung 6-25: Client GUI neuer Termin

7. Qualitätssicherung

7.1 Unit Tests

Da „Test Driven“ entwickelt wurde, sind 35 Tests vorhanden. Diese schreiben direkt auf den Exchange Server, weswegen sie eine lange Laufzeit von 80 Sekunden haben.

Nachfolgend eine Übersicht über alle Tests mit den jeweiligen Laufzeiten:

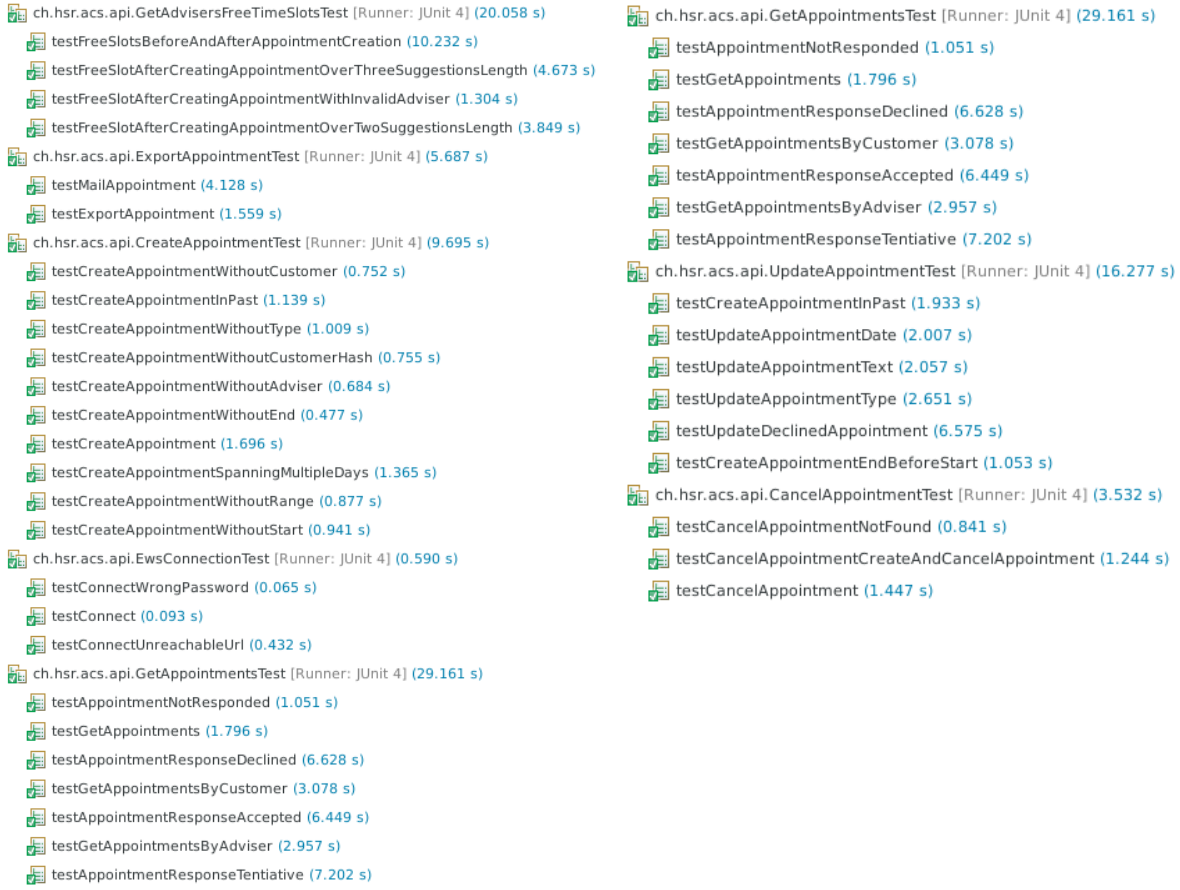


Abbildung 7-1: Unit Test

7.2 Test-Converage

Aus der hohen Anzahl von Tests ergibt sich eine Code Coverage von 85 %. Leider zählen bei aktuellen Coverage Tools für Eclipse die Testfälle, bei denen Exceptions **erwartet** werden, **nicht**. Deswegen ist der tatsächliche Code Coverage Wert noch höher.

7.3 Development Workflow

Nachfolgend wird der festgelegte Entwicklungsablauf beschrieben, welcher eine hohe Code-Qualität gewährleistet.
Beispiel für die Entwicklung eines neuen Features:

1. Redmine Ticket wählen / erstellen
2. Aktuellen Master pullen
3. Branch mit Redmine Ticket Id + frei wählbaren Namen erstellen
4. Entwickeln
5. Änderungen pushen
6. Pull Request auf Github erstellen
7. Pull Request für Code Review an anderen Entwickler geben
8. Allfälliges Feedback verbessern
9. Branch mergen & löschen

Dabei wird grosser Wert darauf gelegt, Test Driven zu entwickeln. Das heisst, vor der Implementierung einer neuen Funktion wird ein Test für diese geschrieben. Sobald der Test erfolgreich durchläuft, kann die Methode refactored werden, wobei immer gewährleistet ist, dass der Output der Funktion richtig ist.

7.3.1 Code Review & Refactoring

In Github hat man mit einem Pull Request eine gute Übersicht der Änderungen. Damit kann ein anderer Entwickler perfekt nachvollziehen was angepasst wurde. Zudem kann er die Änderungen kommentieren und damit Feedback geben und allfällige Änderungen vorschlagen.

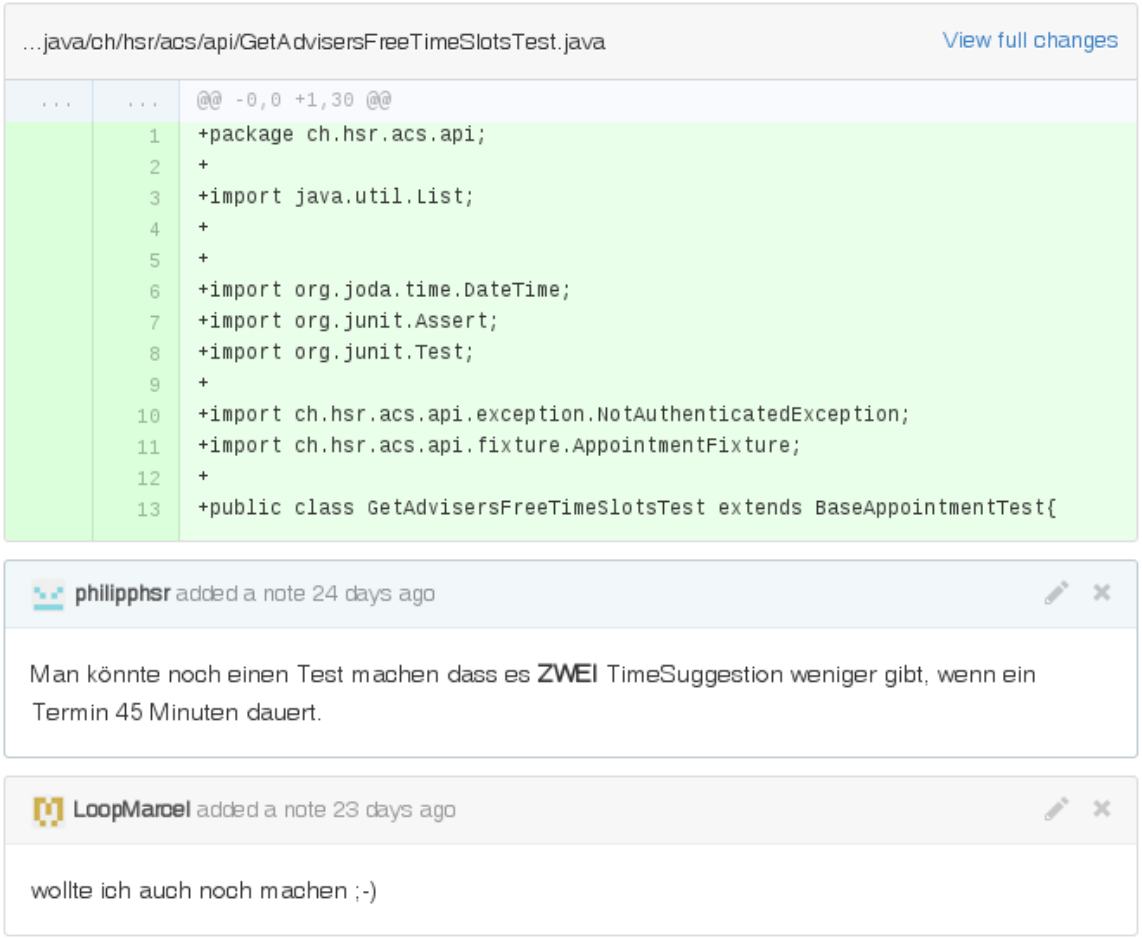


Abbildung 7-2: Code Review

7.4 Continuous integration

Zur Continuous Integration wurde jenkins aufgesetzt. Für jede entwickelte Komponente wurde ein Projekt in jenkins konfiguriert. Bei einem Git Commit in eines der Repository löst ein Webhook den Build des jeweiligen Projektes aus. Damit erfährt man sofort, wenn ein Commit fehlerhaft ist und die Tests nicht sauber durchlaufen. Zusätzlich hat man auf dem Server immer die aktuellste Version des Clients und Servers mit API gebuildet und kann die Änderungen dort ausprobieren.

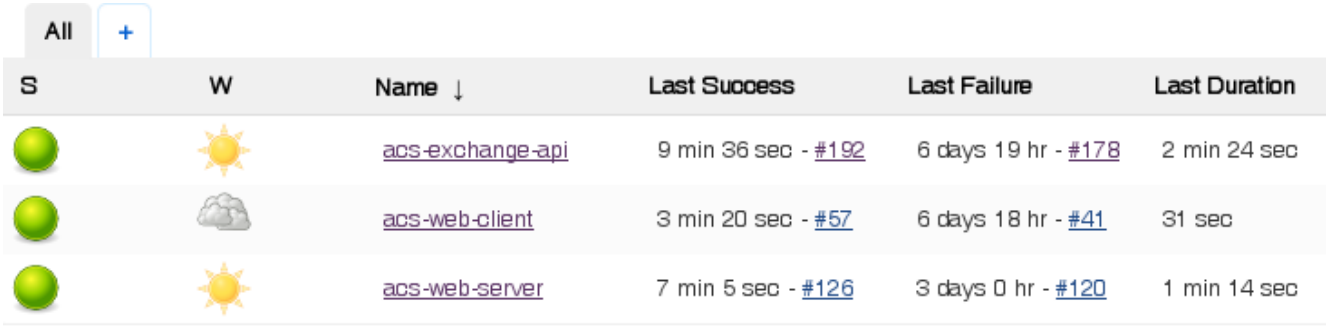


Abbildung 7-3: Jenkis

7.5 Abhängigkeiten Packages

Führt man eine Package Analyse mit Analyse-Tool STAN für Eclipse durch, erhält man folgendes Resultat:

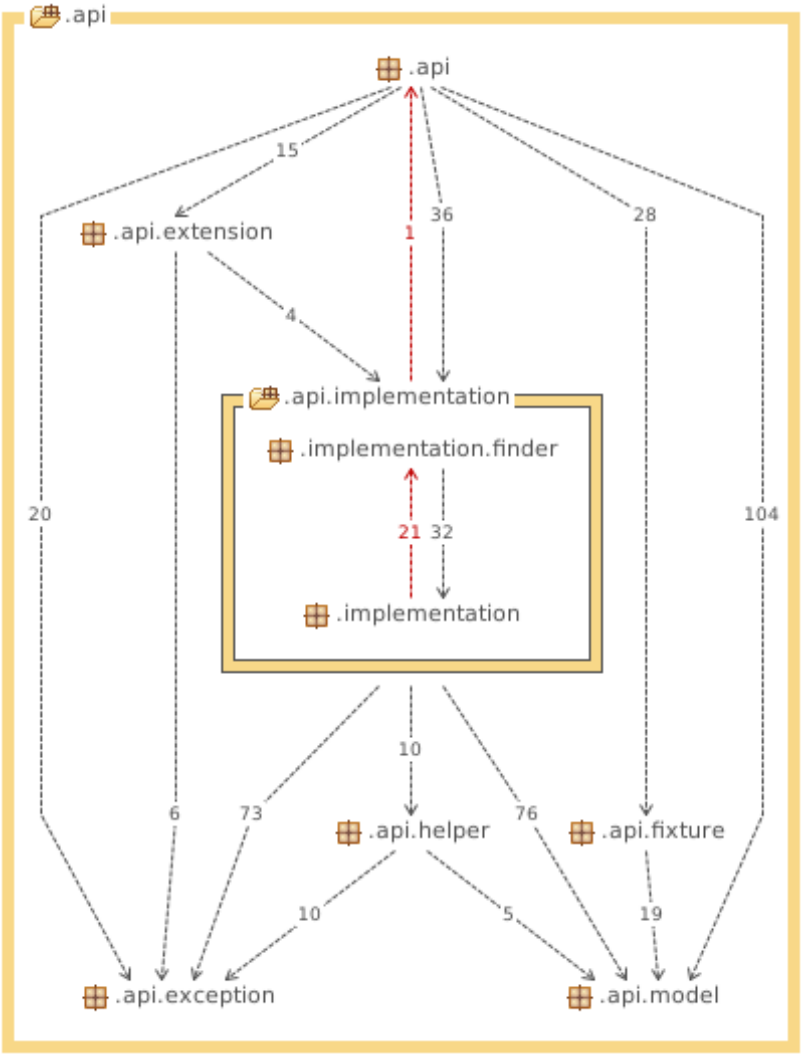


Abbildung 7-4: STAN

Die Abhängigkeit zwischen dem API-Package und dem Implementation-Package existiert nur, da die Klasse EwsExchangeApi das Interface ExchangeApi implementiert.
Ansonsten existieren nur zwischen dem finder- und implementation-Package bidirektionale Abhängigkeiten. Diese Abhängigkeiten existieren weil die Klasse AcsItemFactory eine Instanz des EwsItemFinder hält. Diese Instanz wird benötigt, damit beim Kreieren eines AcsAppointments die aktuelle AppointmentResponse vom Server gelesen werden kann.
Da das finder-Package lediglich die Finder-Klassen gruppiert, um eine bessere Übersicht zu gewährleisten, sind die Abhängigkeiten vom finder- zum implementation-Package vernachlässigbar.
Ansonsten zeigt die Auswertung von STAN keine Mängel auf.

7.6 Metrics

Die folgenden Werte wurden mit dem Analyse Tool „Metrics“ generiert. Die nachstehende Tabelle soll einen zusätzlichen Überblick zum Umfang des Projektes geben.

Folgende Werte ergeben sich aus der Analyse der Komponente acs-exchange-api.

Metric	Wert
Anzahl Klassen	54
Anzahl Methoden	201
Anzahl Zeilen Code	2034
Anzahl Packages	9

Der entwickelte Client in JavaScript hat ca. 630 Zeilen Code.

Der entwickelte Server besitzt. 243 Zeilen Code.

Somit besitzen alle Komponenten zusammen ca. 2900 Zeilen Code.

8. Erreichte Ziele / Offene Punkte

In diesem Kapitel sollen die erfüllten und nicht erfüllten Use Cases sowie nicht Funktionalen Anforderungen dargestellt werden.

8.1 Legende

Symbol	Bedeutung
	Überprüft und funktioniert
	Nicht erfüllt

Tabelle 8-1: Legende Symoble

Quelle Icons: https://www.iconfinder.com/icons

8.2 Use Cases

Eine detailierte Beschreibung bzw. Brief und Fully Dressed findet sich im Kapitel 3.3 Use Caes Fully Dressed

8.2.1 Soll-Cases

Alle der Folgenden Klassen wurden im Kapitel 6.2.1 beschrieben.

Nr	Use Case	Bemerkung	Erfüllt?
01	**Termin erstellen (Create)**	Wurde in der Klasse EwsItemFactory implementiert (Siehe Klasse: EwsItemFactory)	
01	**Termin auslesen (Read)**	Wurde in der Klasse EwsAppointmentFinder implementiert (Siehe Klasse: EwsAppointmentFinder)	
01	**Termin ändern (Update)**	Wurde mit den Klassen AcsItemFactory, EwsItemFinder, EwsItemFacotry) implementiert. Mit der Klasse EwsItemFinder wird das Objekt mittels der unique ID ausgelesen und neu erstellt. Die AcsItemFactory konvertiert das Objekt, damit es beim Client angezeigt werden kann. (Siehe: Package: Implementation)	
01	**Termin absagen (Delete)**	Wurde innerhalb der Klasse EwsExchangeApi implementiert. (Siehe Kapitel Package Übersicht)	
04	**Termin bestätigen**	Wurde innerhalb der Klasse EwsResponseFinder imlementiert. (Siehe: Klasse: EwsResponseFinder	

Tabelle 8-2: Ergebnis soll Use Cases

8.2.2 Optionale Use Cases

Nr	Use Case	Bemerkung	Erfüllt?
02	**Termin exportieren**	Termin kann via Mail oder als File exportiert werden. (Siehe: Kapitel 10.2.2 Template - Terminexport).	✓
03	**Terminänderung beantworten**	Aus zeitlichen Gründen nicht mehr implementiert.	✗
06	**Terminänderung vorschlagen**	Aus zeitlichen Gründen nicht mehr implementiert.	✗
07	**Arbeitszeit festlegen**	Wurde innerhalb der Klasse EwsSuggestionFinder implementiert (Siehe: Klasse: EwsSuggestionFinder)	✓

Tabelle 8-3: Ergebnis optionale Use Cases

8.3 Nicht funktionale Anforderungen

Die funktionalen Anforderungen wurde gemäss Kapitel 3.4 festgehalten.

8.3.1 Funktionalität

Was	Beschreibung	Erfüllt?
Richtigkeit	Momentan noch durch ein pull Intervall im Client gelöst	✓
Interoperabilität	Server bietet mit dem Jersey-Framework eine REST-Schnittstelle und kann somit durch diverse Clients angesprochen werden	✓
Sicherheit	Benutzer müssen sich im Active Directory identifizieren können.	✓

Tabelle 8-4: Ergebnis NFR Funktionalität

8.3.2 Zuverlässigkeit

Was	Beschreibung	Erfüllt?
Fehlertoleranz	Im API wurden Exceptions definiert (Siehe: Kapitel 10.4 Exceptions) Im acs-web-server werden Exceptions vom API auf Standard HTTP Meldungen gemapt.	✓

Tabelle 8-5: Ergebnis NFR Zuverlässigkeit

8.3.3 Benutzbarkeit

Was	Beschreibung	Erfüllt?
Verständlichkeit	Da das GUI nur sehr klein ist, bleibt es übersichtlich und bedarf keiner grossen Erklärung.	✓

Tabelle 8-6: Ergebnis NFR Benutzbarkeit

8.3.4 Übertragbarkeit

Was	Beschreibung	Erfüllt?
Anpassbarkeit	Der Serverteil kann als war File exportiert werden und auf einem anderen Applications Server gestartet werden.	✓
Installierbarkeit	Server, Client und API können per git heruntergeladen werden. Server und API können über maven gebuildet werden. Um das ews-java-api einzubinden, steht ein README im git Repository zur Verfügung	✓

Tabelle 8-7: Ergebnis NFR Übertragbarkeit

8.4 Verbesserungsvorschläge

In Zukunft können die zwei offenen Use Cases implementiert werden.
Zudem wären folgende Verbesserungen denkbar:

Einsatz von Sockets:

Um die NRF „Richtigkeit“ eleganter zu lösen, wäre der Einsatz von Sockets zu empfehlen. Dies würde einerseits Performance einsparen, da der Exchange weniger abgefragt wird und andererseits wäre der NRF im acs-exchange-api abgedeckt.

Explizite Angabe wann Berater Zeit hat:

Die Applikation wurde so implementiert, dass der Berater immer als frei markiert ist, wenn dieser keinen Termin hat. Ein anderer Ansatz wäre denkbar, dass der Berater explizit einen Termin erstellt, wann er Zeit für den Kunden hat. Daraufhin werden dem Kunden nur Termine in diesem Zeitrahmen vorgeschlagen.

Caching im API:

Um den Exchange Server zu schonen, wäre es sinnvoll Requests zwischen zu speichern und nur bei Änderungen einen Request auszulösen. Damit kann die Performance verbessert und der Exchange geschont werden.

9. Persönliche Berichte

9.1 Marcel Loop

Schon bei der Auswahl der Studienarbeit war für mich klar, dass ich eine Applikation in Java schreiben möchte. Die Aufgabenstellung dieses Projekts entsprach dann voll und ganz meinen Vorstellungen einer Studienarbeit.

Der Gedanke, zusammen mit einem Industriepartner ein Produkt zu entwickeln, welches in einem späteren Zeitpunkt eventuell eingesetzt wird, motivierte mich dann umso mehr.
Die Zusammenarbeit mit der Crealogix war stets sehr angenehm, besonders während der „MS1-Elaboration-Spezifikation“ waren wir froh, dass wir Vorort arbeiten durften und somit sehr schnell die Anforderungen aufnehmen konnten.

Eine neue Herausforderung war ausserdem die Verfolgung des Test-Driven-Development Ansatzes. Es war ungewohnt und bedeutete zu Beginn einen deutlichen Mehraufwand. Dieser zahlte sich dann aber aus, als das Projekt grösser und komplexer wurde. Man konnte neue Features schreiben und wenn alle Test durchliefen konnte man sich ziemlich sicher sein dass man keine Seiteneffekte oder dergleichen hatte.

Da wir nicht nur ein API sondern auch noch eine Serverapplikation und Client entwickelten, war die Arbeit sehr Abwechslungsreich.
Philipp und ich haben schon andere Projekte zusammen durchgeführt und sind mittlerweile ein gut eingespieltes Team. Dies spiegelt sich auch in unserem Qualitätsprozess wieder. Jede Änderung wurde vom Anderen gereviewt. Dies half sehr oft Fehlern vorzubeugen.

In einem nächsten Projekt werden wir sicher Latex für die Dokumentation verwenden. Da wir ab und zu Schwierigkeiten mit Microsoft Word hatten, als wir zusammen am Dokument arbeiteten.

Abschliessend kann ich sagen, dass das Projekt ein voller Erfolg war und ich so einiges mitnehmen konnte. Zudem möchte ich der Crealogix und Herrn Prof. Hans Rudin für die gute Zusammenarbeit danken.

9.2 Philipp Koster

Ich bin sehr zufrieden mit der Durchführung der Arbeit und dem Endprodukt.
Mit Crealogix hatten wir einen in der IT-Branche erfahrenen und etablierten Industriepartner. Während der Elaboration waren wir wöchentlich in ihrem Hauptsitz in Altstetten und konnten auf speditive Art und Weise Besprechungen durchführen, Anforderungen aufnehmen und an der Durchführungsplanung arbeiten.

Der Umfang der Arbeit erwies sich als optimal, da es nicht zu viel, aber auch nicht zu wenig war. Deswegen konnten wir alle Prozesse sauber planen wie auch durchführen und hatten nur einen geringen Zeitdruck. Gezeigt hat sich dies im Entwicklungsablauf, da wir jede Codeänderung vom anderen gegenlesen lassen konnten, was zu einer sehr hohen Code Qualität führte, da kleine wie auch grosse Fehler sofort auffielen. Ausserdem haben wir nach dem „Test-Driven-Ansatz“ entwickelt, wodurch wir eine sehr hohe Code Coverage erreichten und jede Funktion durch Tests abdecken konnten. Es hat Spass gemacht so systematisch vorzugehen und damit Redundanzen und Falschentwicklungen zu vermeiden.

Auch der technologische Ansatz war sehr interessant. Über ein Java API, sowie einem REST Web-Server entwickelten wir auch einen Web-Client mit JavaScript. Daraus ergab sich Abwechslung während der Entwicklung und die saubere Umsetzung der Architektur und Kommunikation forderte uns heraus.

Im Verlauf der Arbeit bereuten wir, dass wir uns nicht für Latex zur Dokumentation entschieden haben. Die gemeinsame Arbeit im Google Docs verlief zwar problemlos, allerdings war das Layouten am Ende in Microsoft Word umso mühsamer.

Insgesamt hat die Arbeit aber Spass gemacht, war sehr Interessant und ich habe viel gelernt. Zu alle dem haben Marcel und ich uns als gutes Termin erwiesen und werden auch gemeinsam die Bachelorarbeit bestreiten.

10. Verwendung API

10.1 Initialisieren

Um das API zu initialisieren braucht man eine Instanz der Klasse **EwsExchangeApiConfiguration**. Auf dieser müssen mit Settern folgende Werte gesetzt werden:

Wert	Beschreibung	Beispielwert
url	URL zum EWS Endpunkt.	https://acs.hsr.ch/EWS/Exchange.asmx
version	Exchange Version die angesprochen werden soll (enum EXCHANGE_VERSION).	EXCHANGE_VERSION.Exchange2010
domain	Domäne der Firma.	"acs.local"
username	Benutzername des Benutzers, der vom API verwendet wird um Termine zu erstellen.	“acs”
password	Passwort des Benutzers.	“supersecure”
templatesPath	Der Pfad zu den Templates.	/etc/acs/templates/
emailTemplateFileName	Der Name des Templates für das E-Mail einer Terminanfrage.	Terminanfrage.html
exportTemplateFileName	Der Name des Templates für das E-Mail eines Terminexports.	Terminexport.html
exportFileName	Der Name des exportieren Files.	Termin.ics

Tabelle 10-1: Verwendung API

Das EwsExchangeApi kann daraufhin mit der Instanz der EwsExchangeApiConfiguration, sowie einer Instanz der EwsConnection erstellt werden. Die EwsConnection kann einfach aus der EwsExchangeApiConfiguration erstellt werden.

Folgendes Beispiel zeigt diese Aufrufe:

```
EwsExchangeApiConfiguration configuration = Configuration.buildConfiguration();
connection = configuration.createConnection();
api = new EwsExchangeApi(connection, configuration);
```

Code Snippet 10-1: EwsExchangeApi Instanzierung

10.2 Templates

Mit der Konfiguration werden im API-Pfäde gesetzt, an denen sich benötigte Templates befinden. Bei den Templates handelt es sich lediglich um HTML Dateien die Platzhalter beinhaltet. Diese Platzhalter werden beim Generieren der Nachricht mit den jeweiligen Werten ersetzt.
Wichtig ist, dass CSS-Styles nicht aus CSS-Dateien eingebunden werden, sondern Inline im HTML-File vorhanden sind. Das ist notwendig, da Outlook den Head-Tag von der Nachricht entfernt. Für das Inlinen von CSS Styles kann man allerdings Webtools wie folgendes verwenden: http://templates.mailchimp.com/resources/inline-css/

Nachfolgend werden die benötigten Templates und die jeweiligen Platzhalter beschrieben:

10.2.1 Template - Terminanfrage

Aus den Konfigurationparametern ergibt sich der Pfad: *templatesPath* + emailTemplateFileName. Also Beispielsweise: '/etc/acs/templates/Terminanfrage.html'.

Das Template wird für Terminanfragen verwendet. Folgende Platzhalter müssen vorhanden sein:

Platzhalter	Wird ersetzt durch	Beispielwert
${customer}	… den Namen des Kunden	Max Muster
${type}	… die Art der Anfrage	Anruf
${datetime}	… den Zeitraum der Anfrage	20.12.2014 von 15:00 - 15:30 Uhr
${text}	… den Text der Anfrage	Hallo, hätten Sie Zeit? Gruss

Tabelle 10-2: Template Parameter

10.2.2 Template - Terminexport

Aus den Konfigurationparametern ergibt sich der Pfad: *templatesPath* + *exportTemplateFileName*. Also Beispielsweise: '/etc/acs/templates/Terminexport.html'.
Das Template enthält keine Platzhalter.

10.3 Methoden

Das Exchange-Api umfasst folgende Methoden:

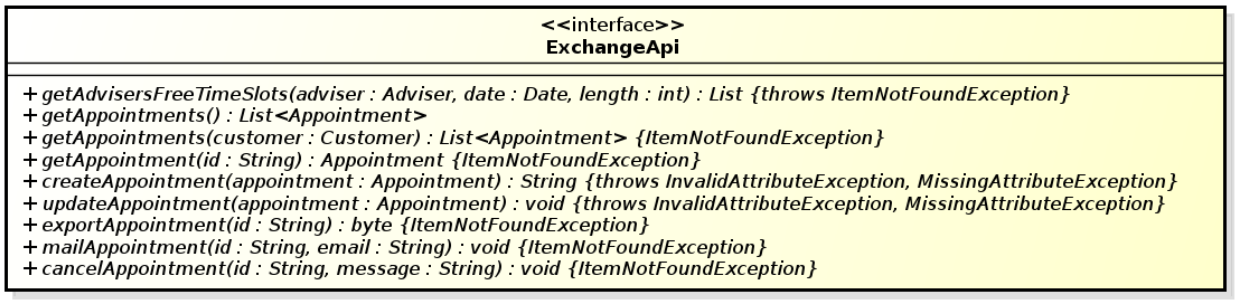


Abbildung 10-1: ExchangeApi

Nachfolgend wird auf die Methoden mit Erklärungsbedarf genauer eingegangen:

Methoden	Beschreibung
getAdvisersFreeTimeSlots(Adviser adviser, Date date, int length)	Die Methode gibt die TimeSlots für den übergebenen Tag zurück in denen der übergebene Adviser Zeit für einen Termin hat. Die Länge der Vorschläge wird dabei durch den Integer Parameter length bestimmt. Beim Auslesen der TimeSlots wird die Arbeitszeit, sowie hängige Terminanfragen von anderen Kunden berücksichtigt.
createAppointment(Appointment appointment)	Erstellt das übergebene Appointment. Dadurch wird dem Berater ein E-Mail mit der Terminanfrage geschickt.
updateAppointment(Appointment appointment)	Updated das übergebene Appointment mit den übergebenen Werten. Der Berater erhält die Änderungen ebenfalls und falls der Termin bereits bestätigt war, wird dieser wieder auf den Status Offen gesetzt.
exportAppointment(String id)	Gibt einen byte-Array des übergebenen Appointments im ICAL-Format zurück.
mailAppointment(String id, String email)	Schickt der übergebenen E-Mail Adresse das Appointment mit der übergebenen Id im ICAL-Format.
cancelAppointment(String id, String message)	Sagt das Appointment mit der übergebenen Id ab. Der Berater wird darüber benachrichtigt und erhält die Nachricht, die übergeben wurde.

Tabelle 10-3: Beschreibung ExchangeApi

10.4 Exceptions

Folgende Exception gibt es im acs-exchange-api:

```
exception/
├── AuthenticationFailedException.java
├── EwsPropertiesException.java
├── EwsServiceException.java
├── ExchangeConnectionException.java
├── ExportException.java
├── ForbiddenActionException.java
├── InvalidAttributeException.java
├── ItemNotFoundException.java
├── MissingAttributeException.java
├── NotAuthenticatedException.java
├── NotConnectedException.java
├── TemplateException.java
└── UnsupportedExchangeVersionException.java
```

Abbildung 10-2: Exceptions im acs-exchange-api

Nachfolgend wird auf die Exceptions mit Erklärungsbedarf genauer eingegangen:

Exception	Beschreibung
EwsPropertiesException	Wird geworfen, wenn ein Fehler beim Auslesen von Extended Properties auftritt.
EwsServiceException	Wird bei Fehlern bei der Kommunikation mit dem ExchangeService geworfen.
ForbiddenActionException	Wird geworfen, wenn Operationen ausgeführt werden, die eigentlich erlaubt sind, aber vom API nicht erlaubt werden. Beispielsweise beim Bearbeiten von bereits abgelehnten Appointments.

Tabelle 10-4: Beschreibung Exceptions

11. Abbildungsverzeichnis

12. Code Snippets

13. Tabellenverzeichnis

14. Glossar

Begriff	Beschreibung
RUP	Rational Unified Process ist ein agiles Vorgehensmodell in der Softwareentwicklung.
REST	Vorgehensweise zur Kommunikation über das HTTP Protokoll. Es beinhaltet die vier Verben. • Post • Put • Get • Delete
API	Application Programming Interface. Programmierschnittstelle
Active Directory	Verzeichnisdienst von Microsoft welches LDAP verwendet
Test Driven Development	Vorgehensweise in der Softwareentwicklung. Dabei wird vor der Implementierung ein Test dazu geschrieben.
UC	Abkürzung für Use Case
CRUD	Abkürzung für Create, Read, Update und Delete
SOAP	SOAP ist ein Transportprotokoll, welches in der Anwendungsschicht angliedert ist. SOAP stützt sich auf xml für die Repräsentation der Daten.
EWS	Exchange Web Services ist die Server-Schnittstelle von Exchange
JSON	Java Script Object Notation ist ein Datenformat für den Austausch von Daten
uniqueID	String welcher Micosoft für die eindeutige Identifikation eines Items im Exchange verwendet
CDO	API älterer Microsoft Exchange Server Produkte (Veraltet). Abgelöst von EWS in neueren Exchange Versionen