



Abschlussprüfung Sommer 2017

Fachinformatiker Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines Filebrowsers für den Documentservice der SAP Cloud Platform

Abgabetermin: Würzburg, 15.06.2016

Prüfungsbewerber:

Tobias Stelzer
Goethestraße 56
97493 Bergtheim



Praktikumsbetrieb:

FIS Informationssysteme und Consulting GmbH
Röthleiner Weg 1
97506 Grafenrheinfeld

Inhaltsverzeichnis

1	Einleitung	4
1.1	Beschreibung des Praktikumsbetriebs	4
1.2	Vorwort	4
1.3	Projektbeschreibung	4
1.4	Projektziel	5
1.5	Projektumfeld.....	5
1.5.1	Ausführungsort	5
1.5.2	Projektverantwortliche.....	6
1.5.2.1	Projektleitung.....	6
1.5.2.2	Entwicklung	6
1.5.3	Verwendete Programmiersprachen.....	6
1.5.4	Verwendete Entwicklungsumgebungen	6
2	Projektvorbereitung.....	7
2.1	Projektphasen.....	7
2.2	Ressourcenplanung.....	7
2.3	Entwicklungsprozess	8
3	Projektanalyse	8
3.1	Ist-Analyse.....	8
3.2	Wirtschaftlichkeitsanalyse	9
3.3	Anwendungsfälle	10
4	Entwurf	10
4.1	Zielplattform.....	10
4.2	Architekturdesign	10
4.2.1	SAPUI5-Frontend	10
4.2.2	Java-Backend	11
4.3	Entwurf der Benutzeroberfläche.....	11
4.4	Aufbau Documentservice	12
5	Implementierung.....	12
5.1.1	Implementierung des Servers	12
5.1.2	Implementierung der Web-App	14
6	Tests	18
6.1	Unit-Tests	18
6.1.1	Java-Anwendung	18

6.1.2	SAPUI5-Webapp	18
7	Fazit	19
8	Glossar	20
8.1	Apache Software Foundation	20
8.2	Apache Tomcat.....	20
8.3	CMIS-Standard	20
8.4	DocumentService	20
8.5	Eclipse	20
8.6	HTTP	21
8.7	JSON.....	21
8.8	Java EE Spezifikation	21
8.9	MVC	21
8.10	Mockup	21
8.11	OASIS	21
8.12	RFC	22
8.13	SAP SE	22
8.14	SAPUI5.....	22
8.15	SCP	22
8.16	Servlet	23
8.17	Use-Case-Diagramm	23
8.18	WebIDE	23
9	Anhang	24
9.1	Detaillierte Zeitplanung	24
9.2	Use-Case-Diagramm	25
9.3	Mockup der Benutzeroberfläche	26
9.4	Klassendiagramm des Java-Backends	27
9.5	Beispiel eines Ordners als JSON-Repräsentation	28
9.6	Tabelle: Parameter bei Anfragen per GET-Methode	29
9.7	Tabelle: Parameter bei Anfragen per POST-Methode.....	30
9.8	Beispiel eines Request-Bodies beim Dateien hochladen	30
9.9	Codebeispiel: Master.view.xml.....	31
9.10	Codebeispiel: Master.controller.js	31
9.11	Codebeispiel: ServletFilebrowser.java	33
9.12	Testergebnis: Unit-Tests der Java-Anwendung.....	35
9.13	Testergebnis: Unit-Tests der SAPUI5-Webapp	36

1 Einleitung

1.1 Beschreibung des Praktikumsbetriebs

Die FIS Informationssysteme und Consulting GmbH (im Folgenden nur noch FIS GmbH genannt) ist ein unabhängiges Unternehmen mit über 600 fest angestellten Mitarbeiterinnen und Mitarbeitern in der Firmengruppe, das seinen Schwerpunkt in SAP-Projekten hat.

Die FIS GmbH entwickelt eigene Lösungen, welche die SAP-Standard-Softwareprodukte sinnvoll ergänzen und eine wichtige Rolle in der Anwendungsbebauung der Anwenderunternehmen spielen. Mit Ihrer Warenwirtschaftslösung fokussiert Sie in erster Linie Unternehmen aus dem Bereich des Technischen Großhandels. Aber Sie setzt auch immer mehr auf innovative Projekte, die für die SAP Cloud Platform entwickelt werden.

Die Tochtergesellschaft FIS-ASP betreibt und administriert die SAP-Systeme von Kunden in FIS-ASP-Rechenzentren.

1.2 Vorwort

Die SAP Cloud Platform (im Folgenden nur noch SCP genannt) ist eine von SAP SE betriebene Cloud mit der Platform-as-a-Service Infrastruktur. Es lassen sich dort unter anderem Java-Anwendungen und HTML5-Anwendungen einbinden, die dann in der Cloud laufen. Auf der SCP werden auch Datenbanksysteme und andere Services angeboten, die von den Anwendungen konsumiert werden können.

Einer dieser Services ist der DocumentService. Er erlaubt das Verwalten von Dateien in einer Ordnerhierarchie. Er kann von Java-Anwendungen verwendet werden, um Dateien und Ordner zu speichern, zu löschen oder herunterzuladen.

1.3 Projektbeschreibung

In einigen Projekten, die für die SCP entwickelt werden, wird der DocumentService zur Verwaltung von Dateien verwendet. Jedoch bietet der DocumentService keine Oberfläche um die Dateien und Ordnerhierarchie einzusehen oder manuell zu verwalten. Die Verwaltung ist nur programmatisch möglich. Oft werden während der Entwicklung von solchen Projekten kleine Programmteile zum Debugging geschrieben, um zu sehen, ob die

Verwaltungsfunktionen des Programms auch tatsächlich auf dem Documentservice die gewünschten Effekte haben.

Deshalb soll jetzt ein universal einsetzbares Tool entwickelt werden, das eine grafische Oberfläche bietet, mit welcher der Documentservice manuell verwaltet werden kann. Das Tool soll als Webanwendung auf der SCP laufen und in dem modernen SAPUI5-Framework entwickelt werden. Es benötigt auch ein Backend als Vermittler, da es nicht direkt mit dem Documentservice kommunizieren kann. Dieses Backend soll in Java entwickelt werden und muss auf der SCP betrieben werden um mit dem Documentservice kommunizieren zu können.

Die App soll die Möglichkeit bieten die Ordnerhierarchie auf dem Documentservice anzuzeigen. Der Benutzer soll durch die Hierarchie navigieren können um sich die Dateien in den jeweiligen Ordnern anzeigen zu lassen. Es soll auch möglich sein, Dateien hochzuladen und neue Ordner zu erstellen. Es müssen sich Dateien und Ordner auch löschen lassen.

1.4 Projektziel

Ziel des Projekts ist es, die Arbeit der Entwickler zu vereinfachen und besonders Zeit beim Debugging zu sparen, welches speziell für die Funktionalitäten in Verbindung mit dem Documentservice anfällt. Aber auch die einfache Verwaltung des Documentservices zu administrativen Zwecken ist ein Ziel dieses Projekts.

1.5 Projektumfeld

1.5.1 Ausführungsort

Das Projekt wird in einem Büroraum der FIS GmbH in Grafenrheinfeld entwickelt. Im Büroraum befindet sich das Team 4 der Abteilung "Kundenentwicklungsprojekte", welches aus 7 Angestellten besteht.

1.5.2 Projektverantwortliche

1.5.2.1 Projektleitung

Name: Stefan Seufert

E-Mail: s.seufert@fis-gmbh.de

Telefon: 09723 / 9188-797

1.5.2.2 Entwicklung

Name: Tobias Stelzer

E-Mail: t.stelzer@fis-gmbh.de

1.5.3 Verwendete Programmiersprachen

Für das Projekt wird ein Backend in Java entwickelt.

Das Frontend wird als Webanwendung in JavaScript und dem SAPUI5-Framework entwickelt.

1.5.4 Verwendete Entwicklungsumgebungen

Für die Entwicklung des Java-Backends wird Eclipse in der Version Neon.1a verwendet.

Das Frontend wird in der WebIDE entwickelt, welche als Service auf der SCP angeboten wird und über einen Webbrowser bedienbar ist.

2 Projektvorbereitung

2.1 Projektphasen

Das Projekt wurde im Zeitraum vom 03.04.2017 – 02.05.2017 bearbeitet. Dafür wurden 70 Stunden eingeplant, die nach eigenem Ermessen auf mehrere Phasen aufgeteilt wurden.

Für die groben Projektphasen mit voraussichtlicher Dauer wurde die *Tabelle 1: Grobe Zeitplanung* angelegt. Eine feinere Ausfertigung befindet sich im [Anhang 9.1](#).

Projektphase	Geplante Zeit
Analysephase	3 h
Entwurfsphase	15 h
Realisierungsphase	40 h
Testphase	4 h
Dokumentation	8 h
Gesamt	70 h

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Personal

- Praktikant Fachinformatiker für Anwendungsentwicklung – Projektumsetzung
- Entwickler – Projektleitung

Hardware

- Laptop, Bildschirm, Tastatur, Maus
- Internetzugang

Räumlichkeiten

- Büroarbeitsplatz

2.3 Entwicklungsprozess

Während der Entwicklung des Projekts wurde nach dem Wasserfallmodell vorgegangen. Dabei wurden linear die folgenden Schritte durchlaufen:

1. Anforderungen festlegen
2. Softwarearchitektur planen
3. Implementieren
4. Testen
5. Abnahme

Dieser Entwicklungsprozess wurde gewählt, da die Anforderungen sehr klar waren und deshalb keine iterarische Vorgehensweise nötig war.

3 Projektanalyse

3.1 Ist-Analyse

In einigen Projekten, die für die SCP entwickelt werden, wird der Documentservice zur Verwaltung von Dateien verwendet. Jedoch bietet der Documentservice keine Oberfläche um die Dateien und Ordnerhierarchie einzusehen oder manuell zu verwalten. Die Verwaltung ist nur programmatisch möglich. Oft werden während der Entwicklung von Projekten, die den Documentservice verwenden, kleine Programmteile zum Debugging geschrieben, um zu sehen, ob die Verwaltungsfunktionen des Programms auch wie erwartet auf dem Documentservice funktionieren. Dies kostet Zeit von Programmierern, die für das eigentliche Projekt verwendet werden könnte.

3.2 Wirtschaftlichkeitsanalyse

Zuerst wird der Stundenlohn des Entwicklers berechnet, welcher ein Praktikant ist. Es wird von einem Monatsgehalt von 300,00 € ausgegangen. Aufgrund der betrieblichen Geheimhaltung werden für die Rechnung fiktive Werte verwendet.

Arbeitstage im Monat: ~20

*Stunden im Monat = 20 * 8 h = 160 h*

$$\text{Stundenlohn} = \frac{300 \text{ €}}{160 \text{ h}} = 1,875 \frac{\text{€}}{\text{h}}$$

Für die verwendeten Ressourcen wird ein Stundensatz von 10 € angenommen und für die Angestellten des Betriebes wird ein Stundenlohn von 30 € angenommen. Im Folgenden werden die Vorgänge mit deren Kosten gelistet.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklung	70 h	1,875 € + 10 € = 11,875 €	831,25 €
Hilfestellung von Kollegen	1 h	30 € + 10 € = 40 €	40,00 €
Abnahme	1 h	30 € + 10 € = 40 €	40,00 €
Gesamt			911,25 €

Um einschätzen zu können, ob sich die Umsetzung des Projekts lohnt, wird nun die Amortisierungsdauer berechnet. Es wird davon ausgegangen, dass der Filebrowser jährlich in 6 Projekten eingebunden wird. Für die Erstellung einer eigenen Documentservice-Verwaltung wird eine Dauer von 11 Stunden angenommen, während für die Einbindung des Filebrowsers 15 Minuten ausreichen.

$$\text{Einsparung pro Projekt} = 11 \text{ h} - 0,25 \text{ h} = 10,75 \text{ h}$$

$$\text{Einsparung pro Jahr} = 10,75 \text{ h} * 6 = 64,5 \text{ h}$$

$$\text{Jährliche Einsparung in €} = 64,5 \text{ h} * (30 \text{ €} + 10 \text{ €}) = 2.580,00 \text{ €}$$

$$\text{Amortisationsdauer} = \frac{911,25 \text{ €}}{2.580,00 \text{ €}} = 0,35 \text{ Jahre} \approx 18 \text{ Wochen}$$

3.3 Anwendungsfälle

Zur Darstellung der Anwendungsfälle wurde ein Use Case-Diagramm angefertigt, das im [Anhang 9.2 Use-Case-Diagramm](#) zu finden ist.

4 Entwurf

4.1 Zielplattform

Das Projekt wird als Web-App auf der SAP Cloud Platform realisiert. Dadurch ist die Anwendung von überall erreichbar, wo der Benutzer einen Internetzugang hat. Da das SAPUI5-Framework von Haus aus nicht nur PCs, sondern auch Tablets und Smartphones unterstützt, hat der Benutzer eine große Auswahl an verwendbaren Geräten und ist in seiner Mobilität kaum eingeschränkt.

4.2 Architekturdesign

4.2.1 SAPUI5-Frontend

Da SAPUI5 grundsätzlich die Model-View-Controller (MVC) Architektur verwendet, kommt sie auch bei diesem Projekt zum Einsatz. Durch die Verwendung von MVC ist die Anwendung leichter erweiterbar und anpassbar, da die Programmlogik von der Anzeige getrennt wird. Beispielsweise können so Oberflächen ausgetauscht werden, ohne die Logik oder das Model anzupassen.

Model: Als Model kann der Entwickler JSON-Models oder OData-Models verwenden. Das sind clientseitige Implementierungen von SAP, um einen entsprechenden Service zu konsumieren. Die Models können an die Views gebunden werden um die Daten anzuzeigen.

View: In SAPUI5 können Views im XML-Format, JSON-Format oder in JavaScript geschrieben werden. Der Autor verwendet XML-Views, weil dies die bevorzugte Wahl der FIS GmbH ist und auch in den Entwicklungsrichtlinien so festgelegt wurde.

Controller: Controller werden in SAPUI5 ausschließlich in JavaScript geschrieben. Sie steuern die Anwendung, indem einzelne Methoden an Elemente in Views gebunden werden (z.B. Buttons).

4.2.2 Java-Backend

Für die Kommunikation mit dem SAPUI5-Frontend implementiert das Java-Backend die Servlet-Spezifikation. Zur Verwaltung des Documentservices wird die CMIS-API verwendet. Im Folgenden werden die beiden Technologien kurz vorgestellt.

Servlet-Spezifikation:

Das Java-Backend verwendet die Servlet-Technologie, die Teil der Java EE Spezifikation ist, um eine Schnittstelle für die Kommunikation mit dem Frontend zu bieten. Servlets sind Java-Klassen, die die Schnittstelle `javax.servlet.Servlet` implementieren. Sie können Anfragen bearbeiten und dynamisch Antworten zurückliefern, beispielsweise in Form von HTTP-Requests und HTTP-Responses. Servlets sind Komponenten im Web Container eines Java EE Servers. Der Web Container kümmert sich um den Lebenszyklus der Servlets und leitet Anfragen, die an dem Server ankommen an das richtige Servlet weiter.

CMIS-API:

Der Documentservice der SCP ist eine Implementierung des CMIS-Standards, der von OASIS gepflegt wird. Um auf den Documentservice zuzugreifen und ihn zu verwalten, wird von SAP SE und der Apache Software Foundation eine API in Form von Java-Klassen bereitgestellt. Der vom Autor implementierte Java-Service verwendet diese API um bei Anfragen, die an das Servlet gestellt werden, die jeweiligen Aktionen auf dem Documentservice auszulösen.

4.3 Entwurf der Benutzeroberfläche

Das SAPUI5-Framework bietet eine Vielzahl von Controls, die verwendet werden können, um grafische Oberfläche einer Web-App aufzubauen. Vor der Entwicklung der App wurde ein Mockup der Benutzeroberfläche auf Papier angefertigt. Dabei wurde darauf geachtet, dass die Vorstellungen mit den SAPUI5-Controls umsetzbar sind und sich an gewisse Richtlinien – sogenannte Floorplans, die von SAP SE gepflegt werden – halten. Dadurch soll eine gute User Experience gewährleistet werden. Das Mockup ist im [Anhang 9.3](#) zu finden.

4.4 Aufbau Documentservice

Der Documentservice ist in Repositories aufgeteilt, die als einzelne Arbeitsbereiche angesehen werden können. Jedes Repository hat einen Namen und ein Passwort, anhand derer sich eine Verbindung zu dem Repository aufbauen lässt. Außerdem hat jedes Repository seine eigene Ordnerhierarchie mit Root-Ordner. Zur logischen Strukturierung des Documentservices könnte man also für jede Anwendung, die den Documentservice verwendet, ein eigenes Repository erstellen, so dass jede Anwendung ihre eigene Ordnerhierarchie hat.

Dateien und Ordner werden also in den einzelnen Repositories in einer Hierarchie mit einem Root-Ordner verwaltet. Die CMIS-Spezifikation bezeichnet Dateien als Documents und Ordner als Folders. In der API sind die Klassen *Document* und *Folder* Unterklassen von der gemeinsamen Oberklasse *CmisObject*. Jedes *CmisObject* besitzt unter anderem eine eindeutige ID und Informationen über den Ordner, der das *CmisObject* enthält. Ein *Folder*-Objekt hat Informationen über alle *CmisObjects*, die der Ordner enthält. Ein *Document*-Objekt enthält Attribute der zugehörigen Datei, beispielsweise Dateiname, Dateigröße, Dateityp und Inhalt der Datei via *InputStream*.

Es ist anzumerken, dass nach der CMIS-Spezifikation ein *Document* in mehreren Ordnern gleichzeitig enthalten sein kann oder auch existieren kann, ohne in einem Ordner enthalten zu sein. Diese Features werden in der CMIS-Spezifikation entsprechend *Multi-Filing* und *Un-Filing* genannt. Der Filebrowser soll diese Features nicht unterstützen.

5 Implementierung

5.1.1 Implementierung des Servers

Wie schon in Punkt 4.2.2 genannt wurde, implementiert das Java-Backend ein Servlet. Das Servlet bearbeitet HTTP-Anfragen, die über die GET-Methode oder die POST-Methode an den Server gestellt werden, indem es die Methoden

```
void doGet(HttpServletRequest request, HttpServletResponse response) und  
void doPost(HttpServletRequest request, HttpServletResponse response)
```

der Klasse `javax.servlet.http.HttpServlet` überschreibt. Der Web Container des Java EE Servers benötigt einige Metadaten über das Servlet, um in der Lage zu sein es zu instanziiieren und ihm eine gewisse URL (relativ zur URL der Java-Anwendung) zuzuweisen.

Dies geschieht über einen Eintrag in der Deployment Descriptor-Datei namens `web.xml`, mit welcher der Java EE Server mit Metadaten versorgt wird. Der Eintrag in der `web.xml` lautet:

```
<servlet>
  <servlet-name>ServletFilebrowser</servlet-name>
  <servlet-class>de.fisgmbh.tgh.filebrowser.ServletFilebrowser</servlet-class>
  <multipart-config>
    <location>/tmp</location>
    <max-file-size>5242880</max-file-size>
    <max-request-size>20971520</max-request-size>
    <file-size-threshold>0</file-size-threshold>
  </multipart-config>
</servlet>
<servlet-mapping>
  <servlet-name>ServletFilebrowser</servlet-name>
  <url-pattern>/filebrowser/*</url-pattern>
</servlet-mapping>
```

Abbildung 1: Eintrag in Deployment Descriptor

Hierdurch wird dem Servlet das url-pattern `/filebrowser/*` zugeordnet. Das bedeutet, dass alle Anfragen, die an die URL `{JavaApplicationURL}/filebrowser` gerichtet sind, an das Servlet `ServletFilebrowser` weitergeleitet werden. Der Code für das Servlet befindet sich als Codebeispiel in [Anhang 9.11](#).

Um leichter mit dem DocumentService arbeiten zu können, wurden die Klassen `DocumentAdapter`, `FolderAdapter` und deren übergeordnete Klasse `ObjectAdapter` erstellt. Ein Klassendiagramm befindet sich in [Anhang 9.4](#). Diese Klassen repräsentieren ihr jeweiliges Gegenstück der CMIS-API und bilden eine Abstraktionsstufe zwischen dem Servlet und dem DocumentService. Sie enthalten unter anderem Methoden, die JSON-Repräsentationen als String zurückgeben, welcher direkt in ein Model der Webapp geladen werden kann. Eine beispielhafte JSON-Repräsentation eines Ordners befindet sich im [Anhang 9.5](#). Das Servlet verwendet diese Adapter-Klassen um die nötigen Aktionen, gemäß der eintreffenden HTTP-Anfrage, durchzuführen. In der Tabelle im [Anhang 9.6](#) werden alle möglichen Aktionen, die das Servlet als Reaktion auf eine Anfrage mit der GET-Methode durchführen kann, aufgelistet und durch welche Anfrage-URL sie ausgelöst werden.

Sonderfall: Das Hochladen von Dateien funktioniert mit einer Anfrage per POST-Methode und wird im Folgenden erklärt. Um eine Datei hochzuladen wird eine Anfrage per POST-Methode an die URL `"/filebrowser"` gesendet. Die genannte URL ist relativ zur URL der Anwendung. Außerdem muss der HTTP-Header "Content-Type" der Anfrage auf `multipart/form-data` gesetzt sein und der Request-Body dementsprechend formatiert sein. Die Spezifikation für diese Formatierung findet sich im RFC1341. Der Request-Body muss

einen Part mit dem Namen "fileUploader" enthalten, der einen Datenstrom der hochzuladenden Datei enthält. Außerdem muss ein weiterer Part mit dem Namen "fileUploader-data" existieren, der einen String mit den Parametern enthält. Der String muss folgendermaßen aufgebaut sein:

`parameter1:wert1;parameter2:wert2;.....;parameterN:wertN`

Eine Tabelle mit einer Übersicht der Parameter bei Anfragen per POST-Methode befindet sich in [Anhang 9.7](#) und ein Beispiel, wie der Request-Body aufgebaut sein muss, befindet sich in [Anhang 9.8](#).

Durch die Verwendung des FileUploader-Controls im SAPUI5-Framework wird es einfacher, solche Anfragen zu senden.

5.1.2 Implementierung der Web-App

index.html: Der Einstiegspunkt ist die *index.html*-Datei, in der das SAPUI5-Framework geladen wird und ein Container-Control aus der SAPUI5-Library erstellt wird, welches in der Datei *Component.js* definiert ist.

manifest.json: Die *manifest.json*-Datei enthält Daten und Metadaten, die für die Initialisierung der App wichtig sind. Beispielsweise der Name und die Beschreibung der App, die nötige Version des SAPUI5-Frameworks, Ressourcen und auch Informationen zum Routing. Die Routing-Informationen weisen Views bestimmte URL-Muster zu, so dass die richtige View angezeigt wird, wenn eine bestimmte URL eingegeben wird.

Component.js: Diese Datei definiert ein Container-Control, welches die App initialisiert und als Container für die Views dient.

Views: Die Views werden im Ordner *views* abgelegt und gemäß Namenskonvention *{Viewname}.view.xml* benannt. Sie beschreiben die Struktur und Formatierung der grafischen Oberfläche durch die Auszeichnungssprache XML. Hierfür verwendet man Tags um SAPUI5-Controls zu kennzeichnen. Mithilfe der Attribute der Tags kann man die Eigenschaften der SAPUI5-Controls anpassen und Daten (aus einem Model) oder Eventhandler (aus dem Controller) an das Control binden. Ein Codebeispiel für eine View befindet sich im [Anhang 9.9](#).

Controller: Die Controller werden im Ordner *controller* abgelegt und gemäß Namenskonvention *{Viewname}.controller.js* benannt. Controller definieren die Eventhandler, die an die zugehörigen Views gebunden werden können und interne Funktionen, die von den Eventhandlern verwendet werden. Sie reagieren also auf Events (beispielsweise Benutzereingaben) und steuern das Verhalten der App. Ein Codebeispiel für einen Controller befindet sich im [Anhang 9.10](#).

Im Folgenden wird die Oberfläche gezeigt und erklärt. In Abbildung 3 sieht man einen Screenshot der Oberfläche, in dem einige Controls rot markiert und nummeriert wurden.

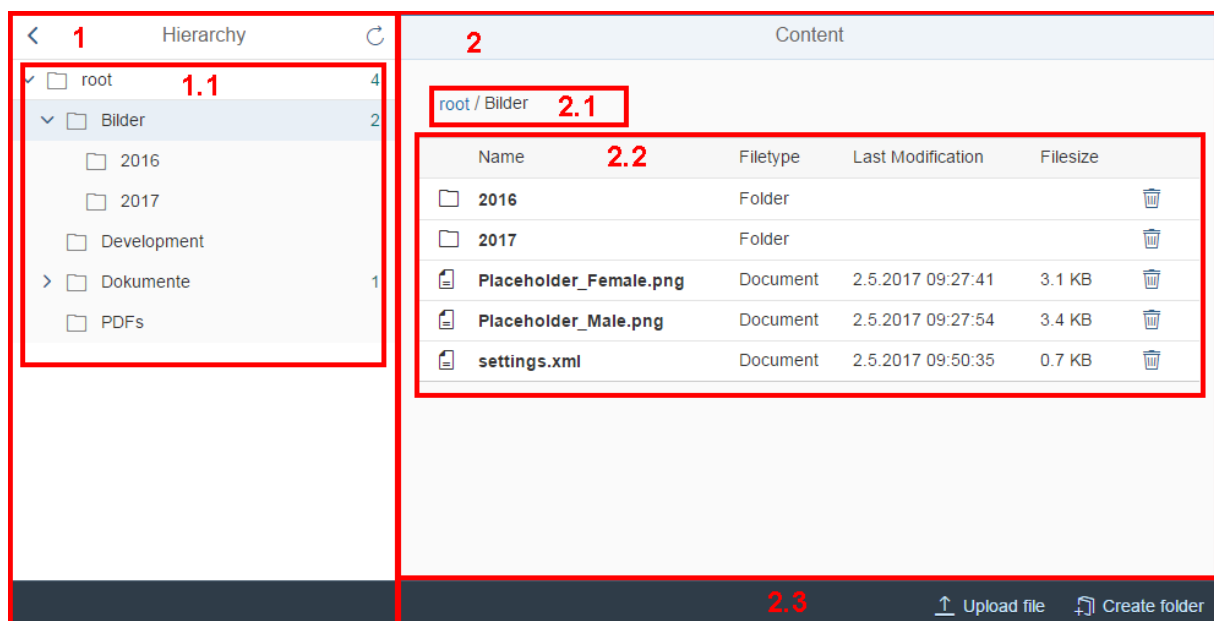


Abbildung 2: Oberfläche der Web-App

Beschreibung der Controls:

1 Hierarchieansicht

Die Hierarchieansicht ist eine View, die in der Datei *Master.view.xml* beschrieben ist. Sie enthält ein MasterPage-Control (sap.m.semantic.MasterPage) mit einem Titel, einem Navigationsbutton und einem Refreshbutton.

Navigationsbutton: Dieser Button navigiert in der Browserhistorie einen Schritt zurück. Er erfüllt also den gleichen Zweck wie der "Zurück"-Button im Browser.

Refreshbutton: Dieser Button lädt die Daten der Hierarchie neu.

1.1 Hierarchiebaum

Der Hierarchiebaum ist ein Tree-Control (`sap.m.Tree`). Er ist an ein Model gebunden, dass die Ordnerhierarchie auf dem Documentservice enthält und zeigt deshalb die Hierarchie an. Außerdem ist das Click-Event seiner Einträge an einen Eventhandler gebunden, so dass bei einem Klick auf einen Eintrag der Inhalt des entsprechenden Ordners auf der rechten Seite (Abschnitt 2) angezeigt wird.

2 Inhaltsansicht

Die Inhaltsansicht ist eine View, die in der Datei *Detail.view.xml* beschrieben ist. Sie enthält ein `DetailPage-Control` (`sap.m.semantic.DetailPage`)

2.1 Ordnerpfad

Der Ordnerpfad besteht aus dem Breadcrumbs-Control (`sap.m.Breadcrumbs`) und zeigt den aktuellen Ordner und alle seine übergeordneten Ordner an, so dass der Benutzer weiß, an welcher Stelle er sich in der Hierarchie befindet. Außerdem kann er auf einen übergeordneten Ordner klicken, um dorthin zu navigieren und dessen Inhalt anzuzeigen.

2.2 Inhaltstabelle

Die Inhaltstabelle ist ein Table-Control (`sap.m.Table`), das in mehrere Spalten aufgeteilt ist. Die Tabelle ist an ein Model gebunden, das mit Daten über den Inhalt des aktuellen Ordners gefüllt ist. Dadurch werden alle beinhalteten Ordner und Dateien dieses Ordners angezeigt. Für Dateien wird außerdem das letzte Änderungsdatum und die Dateigröße angezeigt. Weiterhin befindet sich am Ende jeder Zeile ein Button-Control (`sap.m.Button`), das durch einen Klick einen Löschvorgang der entsprechenden Datei oder des entsprechenden Ordners an das Backend sendet.

2.3 Aktionenleiste

Die Aktionenleiste ist eine `customFooter-Aggregation`, welche ein Teil des `DetailPage-Controls` ist. Sie enthält zwei Button-Controls:

Upload file: Dieser Button öffnet einen Dialog um Dateien in den aktuellen Ordner auf dem Documentservice hochzuladen. Der Dialog ist in Abbildung 4 dargestellt.

Create folder: Dieser Button öffnet einen Dialog um Ordner in dem aktuellen Ordner auf dem DocumentService anzulegen. Der Dialog ist in Abbildung 5 dargestellt.

Im Folgenden sieht man die beiden Dialoge, welche durch die gerade beschriebenen Buttons geöffnet werden können. Der Upload-Dialog ist in Abbildung 4 dargestellt und der Download-Dialog ist in Abbildung 5 dargestellt.

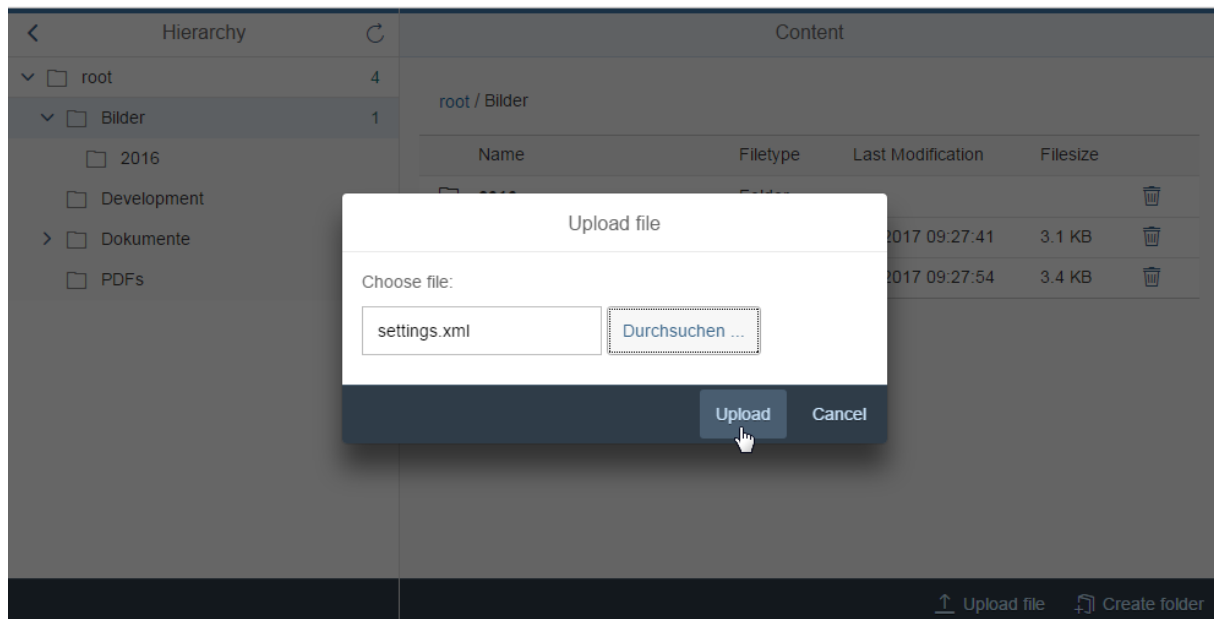


Abbildung 3: Dialog um Dateien hochzuladen

In diesem Dialog wird ein FileUploader-Control (sap.ui.unified.FileUploader) verwendet, um eine Datei von einem Laufwerk auszuwählen. Durch einen Klick auf den Upload-Button werden die benötigten Parameter dem FileUploader mitgeteilt und daraufhin der Upload durch den FileUploader gestartet und der Dialog geschlossen. Durch einen Klick auf den Cancel-Button kann der Dialog jederzeit geschlossen werden, ohne den Upload auszulösen.

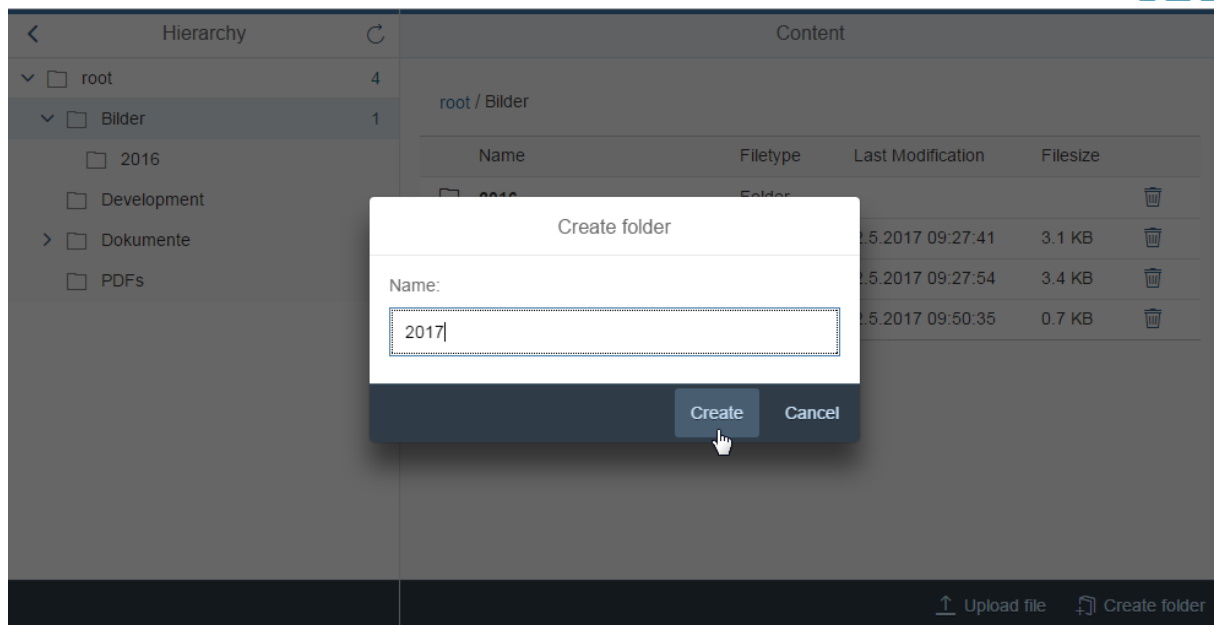


Abbildung 4: Dialog um Order anzulegen

In diesem Dialog wird ein Input-Control (sap.m.Input) verwendet, um einen Namen durch den Benutzer eingeben zu lassen. Durch einen Klick auf den Create-Button wird eine Anfrage an das Backend gesendet, die das Erstellen eines Ordners mit dem eingegebenen Namen auslöst. Durch einen Klick auf den Cancel-Button kann der Dialog jederzeit geschlossen werden, ohne den Erstellvorgang auszulösen.

6 Tests

6.1 Unit-Tests

6.1.1 Java-Anwendung

Mithilfe des Unit-Test Frameworks JUnit wurden Unit-Tests auf der Java-Seite erstellt. Das Ergebnis der Unit-Tests ist in [Anhang 9.12](#) dargestellt.

6.1.2 SAPUI5-Webapp

Es wurden auch Unit-Tests für die SAPUI5-Webapp mithilfe des Frameworks QUnit angelegt. Das Ergebnis eines Testdurchlaufs sieht man in [Anhang 9.13](#).

7 Fazit

Im Rahmen des Projektes hat der Autor zahlreiche neue Erfahrungen gesammelt, unter anderem mit:

- Der Versionsverwaltung "Git"
- Java Servlet Technologie
- Arbeiten mit dem SAPUI5-Framework
- Hypertext Transfer Protocol (HTTP)

Ganz besonders wurde deutlich, wie wichtig eine gute Planung der Ressourcen und des Softwaredesigns ist, um eine wartungsfreundliche Software in einem bestimmten Zeitrahmen zu entwickeln.

Der Filebrowser steht nun für interne Projekte zur Verfügung und es wird darüber nachgedacht, ihn auch externen Kunden anzubieten.

8 Glossar

8.1 Apache Software Foundation

Eine Organisation, die ehrenamtlich arbeitet und Open-Source-Projekte leitet und entwickelt.

8.2 Apache Tomcat

Apache Tomcat ist ein Open-Source-Webserver, der einen Teil der Java EE Spezifikation implementiert. Er ist also kein vollständiger Java EE Server, sondern implementiert nur einen Webcontainer, in welchem Servlets ausgeführt werden können.

8.3 CMIS-Standard

Der Content Management Interoperability Services (CMIS) Standard wird von OASIS spezifiziert und ermöglicht das Speichern und Verwalten von Inhalten auf einem Server. Durch die Standardisierung kann man auf alle CMIS-Services auf die gleiche Weise zugreifen.

8.4 Documentservice

Eine Implementierung des CMIS-Standard von SAP SE, die auf der der SAP Cloud Platform läuft und von Anwendungen auf der Cloud zugreifbar ist. Dieser Service kann verwendet werden, um Inhalte in der Cloud innerhalb einer Ordnerstruktur zu speichern und zu verwalten.

8.5 Eclipse

Eine quelloffene Entwicklungsumgebung, die hauptsächlich für die Entwicklung in Java genutzt wird. Sie beinhaltet auch viele Entwicklungswerkzeuge und ist über Plugins erweiterbar.

8.6 HTTP

Das Hypertext Transfer Protocol (HTTP) ist ein Protokoll zur Datenübertragung. Es wird vor allem von Browsern und Webservern verwendet, um Webseiten über das Internet zu übertragen und Anfragen an Webserver zu tätigen.

8.7 JSON

Das Javascript Object Notation (JSON) Format ist ein kompaktes Datenformat, das strukturierte Daten darstellt. Es wird sehr oft in Verbindung mit Javascript eingesetzt, um Daten zwischen einem Client und einem Webservice zu übertragen.

8.8 Java EE Spezifikation

Java EE ist eine Spezifikation der Oracle Corporation, die den Aufbau von Java EE Servern beschreibt. Diese Spezifikation kann von unabhängigen Anbietern implementiert und angeboten werden.

8.9 MVC

Das Model-View-Controller (MVC) Entwurfsmuster ist eine Architektur für Software, bei der die Ansicht, das Datenmodell und die Programmlogik unabhängig voneinander sind. D

8.10 Mockup

Mockups werden für die grundlegende Planung einer Benutzeroberfläche erstellt. Hierfür wird beispielsweise mit Stift und Papier der Aufbau der Benutzeroberfläche skizziert. So kann man schnell Vorstellungen der Entwickler mit den Anforderungen der Kunden vergleichen.

8.11 OASIS

Die Organization for the Advancement of Structured Information Standards (OASIS) ist eine internationale Organisation, in der ehrenamtliche Mitglieder verschiedene Standards weiterentwickeln.

8.12 RFC

Die Requests for Comments (RFC) sind technische Dokumente mit Vorschlägen zur Standardisierung für alle Bereiche, die mit dem Internet zu tun haben. Die RFCs werden von der Organisation *RFC Editor* editiert, katalogisiert und veröffentlicht. Es gibt keine Vorschriften, wer solche RFCs verfassen und einreichen kann. Viele der heutigen Standards, die mit dem Internet zu tun haben, sind in RFCs verankert.

8.13 SAP SE

Die SAP SE ist ein deutsches Unternehmen, welches im Jahr 1972 gegründet wurde und Unternehmenssoftware entwickelt. In diesem Bereich ist sie internationaler Marktführer mit über 335.000 Kunden in mehr als 180 Ländern.

8.14 SAPUI5

SAPUI5 ist ein modernes HTML5-Framework von SAP SE und die Standard Oberfläche der SAP für Web Applikationen. Das Framework setzt die Verwendung des MVC-Entwurfsmusters voraus und ist gut geeignet um dynamische und übersichtliche Webapplikationen zu entwickeln. Es stellt eine Vielzahl von Steuerungselementen zur Verfügung, mit denen die Benutzeroberfläche gebaut werden kann und die für die Benutzerinteraktion dienen.

8.15 SCP

Die SAP Cloud Platform (SCP) ist ein Cloud-Service mit der PaaS-Infrastruktur. Auf die Cloud können unter anderem Java- und HTML5-Anwendungen hochgeladen werden, die dann auf den Cloud-Servern laufen. Außerdem bietet die Cloud Datenbanksysteme an, die von den Java-Anwendungen verwendet werden können. Weiterhin gibt es mehrere Services, die separat aktiviert und deaktiviert werden können, wie zum Beispiel die WebIDE oder der DocumentService.

8.16 Servlet

Ein Servlet ist eine Java-Klasse, welche die Schnittstelle `javax.servlet.Servlet` implementiert und in einem Webcontainer, der nach der Java EE Spezifikation realisiert ist, ausgeführt wird. Der Webcontainer verwaltet die Servlets auf einem Webserver und leitet Anfragen, die an den Server kommen, an das richtige Servlet weiter. Die Servlets sind dafür da, um Anfragen anzunehmen, zu bearbeiten und eine Antwort zurückzusenden.

8.17 Use-Case-Diagramm

Ein Use-Case-Diagramm stellt Akteure eines Systems dar und Anwendungsfälle, die die Akteure durchführen können. Die Anwendungsfälle geben einen guten Überblick über den Funktionsumfang, der von Benutzern gefordert ist und beugen so Missverständnissen zwischen Entwicklern und Benutzern vor.

8.18 WebIDE

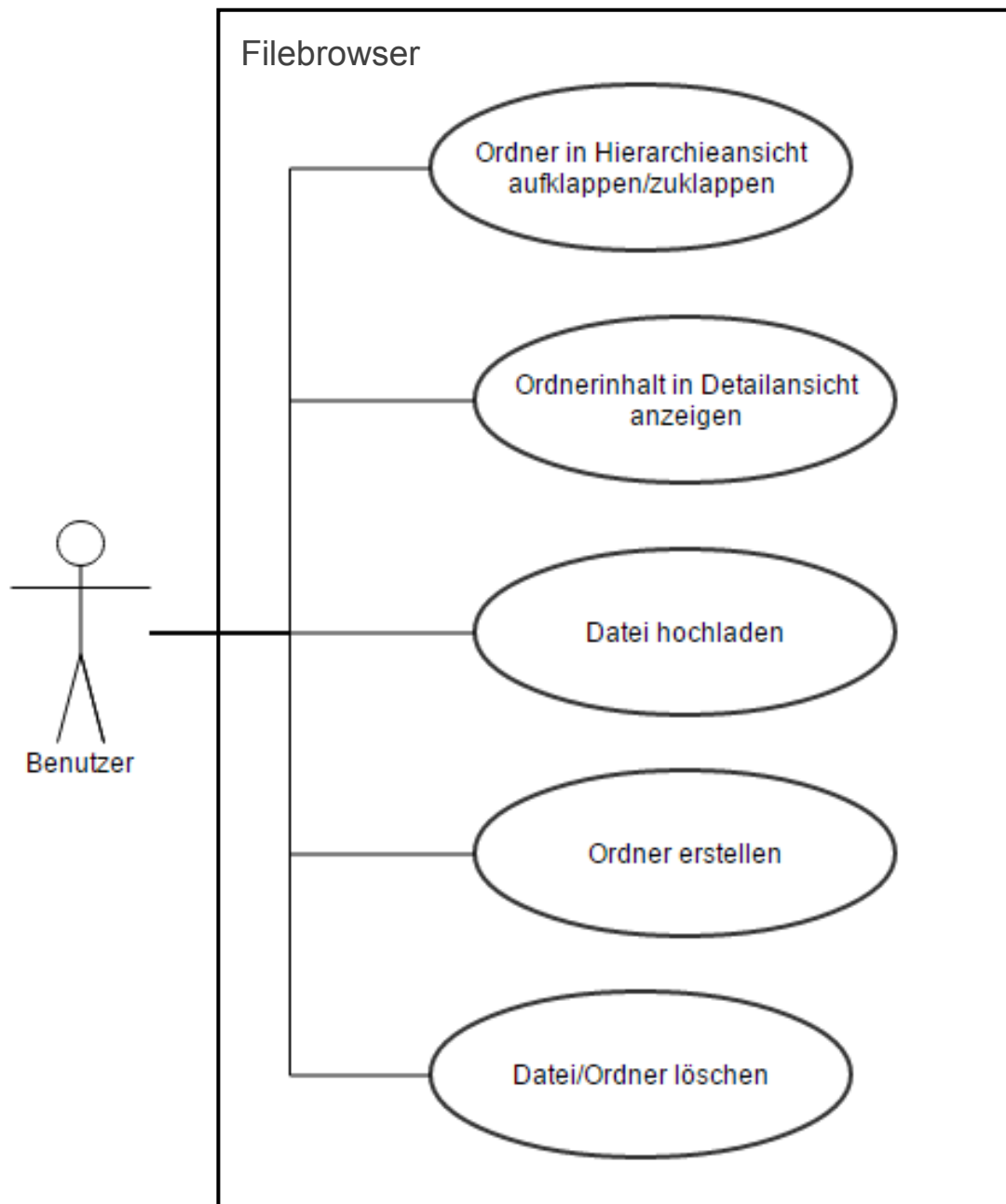
Die WebIDE ist eine Entwicklungsumgebung auf der SAP Cloud Platform für HTML5-Anwendungen. Sie wird im Webbrowser verwendet und bietet die Möglichkeit, entwickelte Projekte schnell und einfach auf die Cloud zu laden.

9 Anhang



























9.1 Detaillierte Zeitplanung

Projektphase	Geplante Zeit
Analysephase	3 h
Ist-Analyse	1 h
Wirtschaftlichkeitsprüfung	1 h
Soll-Konzept erstellen	1 h
Entwurfsphase	15 h
Java-Backend	6 h
SAPUI5-Frontend	6 h
Kommunikation zwischen Backend und Frontend	3 h
Realisierungsphase	40 h
Einrichten der Entwicklungsumgebung	2 h
Java-Backend	16 h
Kommunikation mit Documentservice	9 h
Implementierung der API für das Frontend	7 h
SAPUI5-Frontend	22 h
XML-Views erstellen	6 h
JavaScript-Controller implementieren	16 h
Testphase	4 h
Dokumentation	8 h
Gesamt	70 h

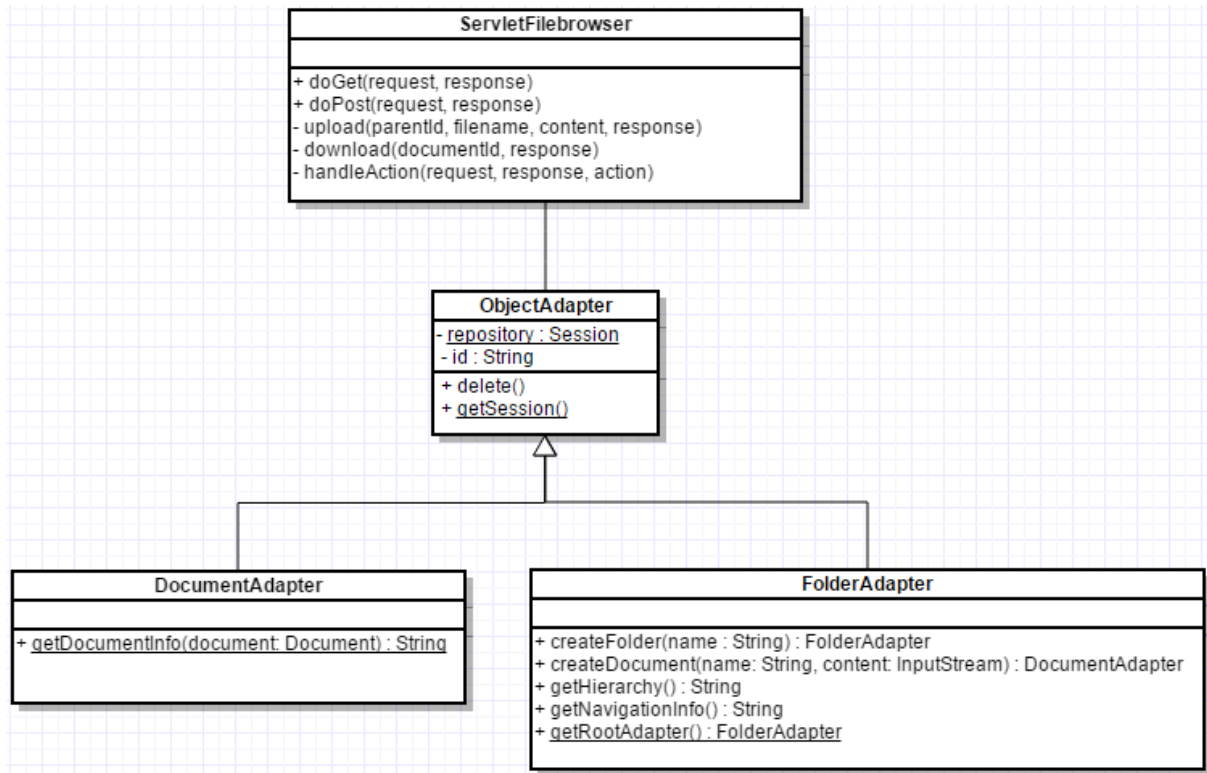
9.2 Use-Case-Diagramm



9.3 Mockup der Benutzeroberfläche

Hierarchie	Inhalt																
<ul style="list-style-type: none">✓  Root✓  Ordner1<ul style="list-style-type: none">>  Ordner1.1<ul style="list-style-type: none"> Ordner1.2>  Ordner 2 Ordner 3	<p>Root/Ordner1</p> <table><tr><th>Name</th><th>Typ</th><th>Größe</th><th>Aktionen</th></tr><tr><td>Ordner 1.1</td><td>Ordner</td><td></td><td> </td></tr><tr><td>Ordner 1.2</td><td>Ordner</td><td></td><td> </td></tr><tr><td>bild.png</td><td>Datei</td><td>50KB</td><td> </td></tr></table>	Name	Typ	Größe	Aktionen	Ordner 1.1	Ordner		 	Ordner 1.2	Ordner		 	bild.png	Datei	50KB	 
Name	Typ	Größe	Aktionen														
Ordner 1.1	Ordner		 														
Ordner 1.2	Ordner		 														
bild.png	Datei	50KB	 														
	<div> Ordner erstellen</div> <div> Upload</div>																

9.4 Klassendiagramm des Java-Backends



9.5 Beispiel eines Ordners als JSON-Repräsentation

So könnte die Antwort des Java-Servers lauten, wenn der Client zu einem bestimmten Ordner navigieren möchte:

```
{
  "SelectedFolder": {
    "Name": "Folder1Layer3",
    "Id": "3SFQRvvAyRz01S2BihoP1cfUeCF1htZsGQ5DhNwLYRE",
    "Type": "Folder",
    "Parents": [{
      "Name": "Folder1Layer2",
      "Id": "if_yqpSDEbjJoL66Zd2NzyTnM1CNEv8bqG9EsdvKf2w",
      "Type": "Folder"
    }, {
      "Name": "14",
      "Id": "HAhBkZaWC9H004CYCSZ8L1P0m2Uf11da3vDKRehIyq0",
      "Type": "Folder"
    }, {
      "Name": "root",
      "Id": "060aea7971862fc06ca4a585",
      "Type": "Folder"
    }
  ],
  "Children": [{
    "Name": "Folder2Layer3",
    "Id": "IhZ6XtRZW7Sx3RukyLZyZG6gdHf10R2TgeIPoJjw4mA",
    "Type": "Folder"
  }
]
}
```

9.6 Tabelle: Parameter bei Anfragen per GET-Methode

Funktion	URL (relativ zu Service-URL)
Hierarchie anfordern	/filebrowser?action=hierarchy
Daten und Inhalt eines Ordners anfordern	/filebrowser?action=navigate&id={FolderId} Parameter: <ul style="list-style-type: none"> FolderId: Die Id des Ordners, dessen Daten angefordert werden
Datei oder Ordner löschen	/filebrowser?action=delete&id={ObjectId} Parameter: <ul style="list-style-type: none"> ObjectId: Die Id des Ordners oder der Datei, die gelöscht werden soll
Ordner erstellen	/filebrowser?action=createfolder&parentid={ParentId}&name={Name} Parameter: <ul style="list-style-type: none"> ParentId: Die Id des Ordners, in dem der neue Ordner angelegt werden soll Name: Name des Ordners, der neu angelegt werden soll
Datei herunterladen	/filebrowser/{DocumentId} Parameter: <ul style="list-style-type: none"> DocumentId: Die Id der Datei, die heruntergeladen werden soll

9.7 Tabelle: Parameter bei Anfragen per POST-Methode

Parameter	Beschreibung
action	Muss "upload" sein, um Datei hochzuladen. Andere Werte sind bei POST-Anfragen nicht möglich.
name	Der Name, unter dem die Datei angelegt werden soll.
parentid	Die ID des Ordners, in dem die Datei gespeichert werden soll.

9.8 Beispiel eines Request-Bodies beim Dateien hochladen

```
--WebKitFormBoundaryQHA7HM7d6h4RvFJO
Content-Disposition: form-data; name="fileUploader"; filename="Test.png"
Content-Type: image/png

--WebKitFormBoundaryQHA7HM7d6h4RvFJO
Content-Disposition: form-data; name="_charset_"
UTF-8

--WebKitFormBoundaryQHA7HM7d6h4RvFJO
Content-Disposition: form-data; name="fileUploader-data"
action:upload;name:Test.png;parentid:060aea7971862fc06ca4a585
```

9.9 Codebeispiel: Master.view.xml

```
<mvc:View controllerName="de.fis.filebrowser.controller.Master"
    xmlns:mvc="sap.ui.core.mvc"
    xmlns:core="sap.ui.core" xmlns="sap.m"
    xmlns:semantic="sap.m.semantic">
    <semantic:MasterPage id="page" title="{i18n>masterTitle}"
        navButtonPress="onNavBack" showNavButton="true">
        <semantic:customHeaderContent>
            <Button icon="sap-icon://refresh" press="onRefresh"
                visible="{= !$device>/support/touch}"
                tooltip="{i18n>refreshButtonTooltip}" />
        </semantic:customHeaderContent>
        <semantic:content>
            <PullToRefresh id="pullToRefresh"
                visible="{device>/support/touch}"
                refresh="onRefresh" />
            <Tree id="Tree" items="{path: 'hierarchyModel>/Hierarchy'}"
                mode="SingleSelectMaster" includeItemInSelection="true"
                busyIndicatorDelay="{masterView>/delay}"
                busy="{masterView>/busy}"
                itemPress="onItemPressed"
                updateFinished="onUpdateFinished">

                <StandardTreeItem title="{hierarchyModel>Name}"
                    icon="sap-icon://folder-blank"
                    type="Active"
                    counter="{=$hierarchyModel>Children.Length}" />

            </Tree>
        </semantic:content>
    </semantic:MasterPage>
</mvc:View>
```

9.10 Codebeispiel: Master.controller.js

```
sap.ui.define([
    "de/fis/filebrowser/controller/BaseController",
    "sap/ui/model/json/JSONModel",
    "sap/ui/model/Filter",
    "sap/ui/model/FilterOperator",
    "sap/m/GroupHeaderListItem",
    "de/fis/filebrowser/model/formatter"
], function (BaseController, JSONModel, Filter, FilterOperator,
    GroupHeaderListItem, formatter) {
    "use strict";

    return BaseController.extend("de.fis.filebrowser.controller.Master", {
        formatter: formatter,

        /* lifecycle methods */

        /**
         * Called when the master list controller is instantiated.
         */
    });
});
```

```

onInit : function () {
    var oTree = this.byId("Tree"),
        hierarchyModel = this.getModel("hierarchyModel"),
        oViewModel = this.getModel("masterView");

    this.getOwnerComponent().oListSelector.setBoundMasterList(oTree);
    this._initViewModel();

    oViewModel.setProperty("/busy", true);

    hierarchyModel.attachEventOnce("requestCompleted", function () {
        oTree.expandToLevel(1);
        oViewModel.setProperty("/busy", false);
        var oItem = oTree.getItems()[0];
        oTree.setSelectedItem(oItem, true, true);
        this._showDetail(oItem);
        var oBinding = oTree.getBinding("items");
        oBinding.attachEvent("change", this.onUpdateFinished, this);
    }, this);

    hierarchyModel.loadData("filebrowser?action=hierarchy");

    this.getRouter().getRoute("master").attachPatternMatched(
        this._onMasterMatched, this
    );
    this.getRouter().attachBypassed(this.onBypassed, this);
},

/* begin: event handlers */
/**
 * Fired after the list data is available
 */
onUpdateFinished : function (oEvent) {
    this.byId("pullToRefresh").hide();
    this.getModel("masterView").setProperty("/busy", false);
    var oTree = this.getView().byId("Tree");
    oTree.rerender();
},

/**
 * Event handler for refresh event. Reloads the data into the model.
 */
onRefresh : function () {
    this.getModel("hierarchyModel").loadData(
        "filebrowser?action=hierarchy"
    );
    this.getModel("masterView").setProperty("/busy", true);
},

/**
 * Event handler for the tree item pressed event
 */
onItemPressed : function (oEvent) {
    var oItem = oEvent.getParameter("listItem") || oEvent.getSource();
    this._showDetail(oItem);
},

/**

```



```

    * Fired when no routing pattern matched.
    * Removes any selection from the master list.
    */
    onBypassed : function () {
        this.getView().byId("Tree").removeSelections(true);
    },

    onNavBack : function() {
        history.go(-1);
    },

    /* begin: internal methods */
    _initViewModel: function () {
        var oModel = this.getModel("masterView");

        oModel.setProperty("title",
            this.getResourceBundle().getText("masterTitleCount", [0])
        );
        oModel.setProperty("noDataText",
            this.getResourceBundle().getText("masterListNoDataText")
        );
    },

    /**
     * Shows the selected item on the detail page
     */
    _showDetail : function (oItem) {
        this.getRouter().navTo("object", {
            objectId : oItem.getBindingContext("hierarchyModel")
                        .getProperty("Id")
        }, bReplace);
    }
    });
}
);

```

9.11 Codebeispiel: ServletFilebrowser.java

```

private void handleAction(HttpServletRequest request, HttpServletResponse
response, String action) throws ServletException {

    switch (action) {
    case "navigate":
        String id = request.getParameter("id");

        FolderAdapter fa = null;
        if (id == null || id.equals("")) {
            fa = FolderAdapter.getRootAdapter();
        } else {
            fa = new FolderAdapter(id);
        }

        String jsonResponse = fa.getNavigationInfo();
        try {

```

```
        response.getOutputStream().print(jsonResponse);
        response.setStatus(HttpServletResponse.SC_OK);
    } catch (IOException e) {
        throw new ServletException("Error when trying to send the
response");
    }
    break;

    case "delete":
        id = request.getParameter("id");
        ObjectAdapter oa = new ObjectAdapter(id);
        oa.delete();
        response.setStatus(HttpServletResponse.SC_OK);
        break;

    case "rename":
        id = request.getParameter("id");
        String newName = request.getParameter("name");
        oa = new ObjectAdapter(id);
        oa.rename(newName);
        response.setStatus(HttpServletResponse.SC_OK);
        break;

    case "createfolder":
        String parentid = request.getParameter("parentid");
        String name = request.getParameter("name");
        fa = new FolderAdapter(parentid);
        FolderAdapter newFolder = fa.createfolder(name);
        if (newFolder != null) {
            response.setStatus(HttpServletResponse.SC_OK);
        } else {
            throw new ServletException("Error. A folder with this name already
exists");
        }
        break;

    case "hierarchy":
        try {
            fa = FolderAdapter.getRootAdapter();
            jsonResponse = fa.getHierarchy();
            response.getOutputStream().print(jsonResponse);
            response.setStatus(HttpServletResponse.SC_OK);
        } catch (IOException e) {
            throw new ServletException("Error when trying to send the
response");
        }
        break;

    default:
        throw new ServletException("Error. Invalid value for 'action'-
Parameter");
    }
}
```

9.12 Testergebnis: Unit-Tests der Java-Anwendung

JUnit Test Cases

Class: DocumentAdapterTest

Methods:

testGetDocumentInfo1:	Success!
testGetDocumentInfo2:	Success!
testContentStream1:	Success!
testContentStream2:	Success!
testGetDocumentInfoAsJson:	Success!

Class: FolderAdapterTest

Methods:

testGetHierarchy:	Success!
testGetFolderInfo:	Success!
testGetRootAdapter:	Success!
testCreatedocument1:	Success!
testCreatedocument2:	Success!
testCreatefolder1:	Success!
testCreatefolder2:	Success!
testGetNavigationInfo:	Success!

Class: ObjectAdapterTest

Methods:

testGetSession:	Success!
testDelete:	Success!
testRename:	Success!

Final stats:

Status:	SUCCESS!
Tests run:	16
Failures:	0
Ignored:	0

9.13 Testergebnis: Unit-Tests der SAPUI5-Webapp

Unit tests for Filebrowser

☐ Hide passed tests
 ☐ Check for Globals
 ☐ No try-catch
 ☐ Enable coverage

Module: < All Modules >
 Filter:
Go

QUnit 1.18.0; Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36

Tests completed in 21 milliseconds.
17 assertions of 17 passed, 0 failed.

1. createDeviceModel: Should initialize a device model for desktop (1)	Rerun	6 ms
2. createDeviceModel: Should initialize a device model for phone (1)	Rerun	1 ms
3. createDeviceModel: Should initialize a device model for non touch devices (1)	Rerun	0 ms
4. createDeviceModel: Should initialize a device model for touch devices (1)	Rerun	0 ms
5. createDeviceModel: The binding mode of the device model should be one way (1)	Rerun	0 ms
6. formatter - formatDate: Should return the 3rd March 2017 (1)	Rerun	3 ms
7. formatter - formatDate: Should return the 29th February 2000 (1)	Rerun	0 ms
8. formatter - formatDate: Should not return the 29th February 2001 (1)	Rerun	0 ms
9. formatter - formatFileSize: Should return 0 B (1)	Rerun	0 ms
10. formatter - formatFileSize: Should not return <0.1 KB (1)	Rerun	1 ms
11. formatter - formatFileSize: Should return <0.1 KB (1)	Rerun	0 ms
12. formatter - formatFileSize: Should return 10.0 MB (1)	Rerun	0 ms
13. formatter - formatFileSize: Should round down to 9.9 MB (1)	Rerun	0 ms
14. formatter - formatFileSize: Should round up to 10.0 MB (1)	Rerun	0 ms
15. formatter - formatFileSize: Should return 999.9 KB (1)	Rerun	0 ms
16. formatter - formatFileSize: Should return 1.0 MB (1)	Rerun	1 ms
17. formatter - formatFileSize: Should return 0 B (1)	Rerun	0 ms

9.14 Quellenverzeichnis

RFC:

<https://www.rfc-editor.org/>

<https://tools.ietf.org/html/rfc1341>

Java EE Spezifikation:

<https://docs.oracle.com/javaee/7/firstcup/index.html>

OASIS und CMIS-Spezifikation:

<https://www.oasis-open.org/org>

<http://docs.oasis-open.org/cmisis/CMIS/v1.1/CMIS-v1.1.html>

<http://chemistry.apache.org/java/0.13.0/maven/apidocs/>

SAP SE:

<https://www.sap.com/corporate/de.html>

SAP Cloud Platform:

<https://cloudplatform.sap.com/index.html>

SAPUI5 API:

<https://sapui5.hana.ondemand.com/#docs/api/symbols/sap.ui.html>