

Dokumentation für das Projekt

**Planung und Erstellung eines Moduls zum Upload von  
Dokumenten einschließlich einer E-Mail-Routine zur  
Information der zuständigen Behörde mit einer  
Downloadoption**

Abschlussprojektarbeit für die Ausbildung  
zum Fachinformatiker Fachrichtung Anwendungsentwicklung

Autor: Frank Burkert

Projektzeitraum: 15.03. bis 31.03.2021

## Zeitmitschrift der Projektarbeit

### Prüfungsteilnehmer/-in

Name: Burkert

Vorname: Frank

Login: 1831535815

Datum	Tätigkeit	Zeit in Stunden
15.03.21	Besprechungen für Planungsphase	1,0
15.03.21	Analyse des Ist-Zustandes	1,0
15.03.21	Softwareauswahl	1,0
16.03.21	Ressourcenplanung	0,5
16.03.21	Kostenplanung	1,0
16.03.21	Besprechungen für Durchführungsphase	0,5
17.03.21	Herstellen der Arbeitsumgebung incl. Softwareinstallation	1,5
	Implementierung der Funktionen:	
17.03.21	- Erstellen der Simulationswebseiten	2,0
18.03.21	- Erstellen einer Methode zur Umwandlung von HTML-Seiten in Strings	5,0
19.03.21	- Erstellen der Routes auf dem Webserver	5,0
22.03.21	- Erstellen der Routes auf dem Webserver	5,0
23.03.21	- Erstellen der Methode zur Datenübergabe an MongoDB	8,0
24.03.21	- Erstellen einer E-Mail-Funktion	8,0
25.03.21	- Erstellen von Webserver Routes zum Download	7,0
26.03.21	- Erstellen einer Methode zur Datenübernahme aus MongoDB	5,0
29.03.21	Testen der o.g. Funktionen	8,0
30.03.21	Refaktorisierung	1,0
30.03.21	Besprechungen für Übergabe-, Auswertungs-, und Dokumentationsphase	0,5
30.03.21	Übergabe des Systems	0,5
30.03.21	Erstellung eines Soll-Ist-Vergleiches	1,5
30.03.21	Erstellung einer persönlichen technischen Auswertung	0,5
31.03.21	Erstellung der Dokumentation	8,0

## **Inhaltsverzeichnis**

Planung und Erstellung eines Moduls zum Upload von Dokumenten einschließlich einer E-Mail-Routine zur Information der zuständigen Behörde mit einer Downloadoption .....	1
<b>1. Einleitung .....</b>	<b>1</b>
1.1 Projektbeschreibung .....	1
1.2 Projektumfeld .....	1
1.3 Projektbegründung .....	1
1.4 Projektschnittstellen.....	1
1.4.1 Technischen Schnittstellen.....	1
1.4.2 Personelle Schnittstellen .....	1
1.5 Projektabgrenzung .....	2
<b>2. Planungsphase.....</b>	<b>2</b>
2.1 Projektziele, und –vorgaben .....	2
2.2 Analyse des Ist-Zustandes .....	2
2.3 Projektbeteiligte Personen.....	3
2.4 SOLL-Zustand .....	3
2.5 Zeitplanung.....	3
2.6 Ressourcenplanung.....	4
2.7 Personalplanung.....	4
2.8 Kostenplanung .....	4
2.9 Abweichungen vom Projektantrag .....	4
<b>3. Durchführungsphase .....</b>	<b>5</b>
3.1 Entscheidungen.....	5
3.2 Herstellen der Arbeitsumgebung .....	5
3.3 Diagramme.....	5
3.4 Implementierung der Funktionen .....	5
3.4.1 Erstellen von Webseiten zur Simulation .....	5
3.4.2 Erstellen der Routes des Webservers zum Upload .....	7
3.4.3 Erstellen der Methode zur Datenübergabe an MongoDB .....	8
3.4.4 Erstellen einer Emailfunktion .....	10
3.4.5 Erstellen von Webserver Routes zum Download.....	11
3.4.6 Erstellen einer Methode zur Datenübernahme aus MongoDB.....	12
3.5 Test .....	12
3.6 Refaktorisierung .....	13
3.7 Einrichtung .....	13
<b>4. Übergabe, Auswertungs-, und Dokumentationsphase .....</b>	<b>13</b>
4.1 Dokumentation .....	13
4.2 Soll-Ist-Vergleich - zeitlich .....	13

4.3 Datenschutzrechtliche Betrachtung .....	14
4.4 Ausblick .....	14
4.5 Rückblick (Fazit).....	15
4.6 Persönliche technische Auswertung .....	15
<b>Anhang .....</b>	<b>A</b>
Use-Case-Diagramm.....	A
Aktivitätsdiagramm .....	B
Gantt – Diagramm .....	C
IDE – Projektstruktur .....	D
IDE - pom.xml – dependencys.....	E
Methode zur Datenübergabe an MongoDB .....	F
Methode zur Datenübernahme aus MongoDB.....	G
Testausgaben.....	H
Testausgabe Konsole Webserver .....	H
Testausgabe MongoDB mittels robo3T .....	H
Programmablauf Screenshots .....	I
Programmablauf – Fortsetzung .....	J
Programmablauf – Fortsetzung .....	K
Programmablauf – Fortsetzung .....	L
Glosar.....	M

## **Abbildungsverzeichnis**

Abbildung 1: Kostenplanung .....	4
Abbildung 2: htmlToString – Methode .....	6
Abbildung 3: Ausschnitt aus fileSelectionUpload.html .....	6
Abbildung 4: Beispiel einer post route, hier für die Initialisierung der Websites-Bereitstellung zur Dateiauswahl des Uploads .....	7
Abbildung 5: Übersicht der Datenbank mit robo3T .....	9
Abbildung 6: Anweisungen für die Konsolenausgabe zu Testzwecken .....	12
Abbildung 7: Zeitlicher Soll-Ist-Vergleich .....	13
Abbildung 8: Use-Case-Diagramm.....	A
Abbildung 9: Aktivitätsdiagramm .....	B
Abbildung 10: Gantt -Diagramm .....	C
Abbildung 11: IDE - Projektstruktur .....	D
Abbildung 12: IDE - pom.xml - dependencys .....	E
Abbildung 13: Methode zur Datenübergabe an MongoDB .....	F
Abbildung 14: Methode zur Datenübernahme aus MongoDB .....	G
Abbildung 15: Entsprechende Ausgabe der Konsole mit Session-Id und -Attributen .....	H
Abbildung 16: Ausgabe von robo3T zur Prüfung der Object-Id , filename und metadaten.....	H
Abbildung 17: Programmablauf - Schritt 1 - Der Teilnehmer initialisiert nach Anmeldung den Uploadvorgang.....	I
Abbildung 18: Programmablauf -Schritt 2 - Der Webserver stellt dem Teilnehmer eine angepasste Seite zur Dateiauswahl .....	I
Abbildung 19: Programmablauf - Schritt 3 - Der Absender erhält eine Bestätigungsanzeige für den erfolgten Upload.....	J
Abbildung 20: Programmablauf - Schritt 4 - Der Empfänger erhält eine E-Mail mit Teilnehmerdaten und einem Link zum Download.....	J
Abbildung 21: Programmablauf - Schritt 5 – Der Empfänger hat den Link geöffnet und muss sich nun identifizieren.....	K
Abbildung 22: Programmablauf - Schritt 6 – Der Empfänger hat sich identifiziert und der Download wurde freigegeben.....	K
Abbildung 23: Programmablauf - Schritt 7 – Anzeige des vom Empfänger heruntergeladenen Dokumentes.....	L

## 1. Einleitung

### 1.1 Projektbeschreibung

Die Teilnehmer der Sprachschule müssen ihren jeweiligen Kostenträgern Nachweise sowie Bewertungen über ihre Teilnahme beibringen. Dies geschieht derzeit händisch per Post, Fax oder E-Mail. Der Projektgeber wünscht diesen Prozess durch ein Upload-Tool zu vereinfachen.

### 1.2 Projektumfeld

Die seconos IT & Data Services GmbH wurde 2016 in Berlin gegründet und bietet bundesweit cloudbasierte Software-Lösungen insbesondere für Bildungseinrichtungen an. Schwerpunkt ist das LernManagementSystem „seconos LMS“ welches auf dem KursVerwaltungsSystem „seconos KVS“ basiert.

Die seconos IT & Data Services GmbH ist aus der seconos Qualifications & Services GmbH hervorgegangen, welche selbst ein Bildungsinstitut für Integrations- und Sprachkurse ist.

In der Sprachschule sind ca. 25 Mitarbeiter und Dozenten beschäftigt, diese betreuen ca. 150 Teilnehmer. Die seconos IT besteht derzeit aus 4 Mitarbeitern.

Projektgeber ist die seconos IT & Data Services GmbH.

Die Umsetzung des Projektes erfolgt in den Räumen der seconos IT & Data Services GmbH am Kaiserdamm 97, 14057 Berlin.

### 1.3 Projektbegründung

Im bisherigen Prozess ist ein hoher Zeitaufwand von Nöten und er ist fehleranfällig.

Es ist keinerlei Nachweis über die Bereitstellung vorhanden.

Nachfragen seitens der Behörden, zusätzlicher Abgleich der Informationen sind unnötige Störquellen anderer Aufgaben im Personalbereich.

Beim erneuten Senden von Dokumenten kann es zu unterschiedlichen Informationen kommen, da das Dokument inzwischen eventuell verändert wurde.

Verzögerungen anhängender Prozesse, wie zum Beispiel Leistungszahlungen sind die Folge.

### 1.4 Projektschnittstellen

#### 1.4.1 Technischen Schnittstellen

Vom Projektgeber wurden Vorgaben bezüglich Programmiersprache, Framework und Datenbankanwendung getätigt, um eine spätere Implementierung in das bestehende System zu erleichtern.

Als Programmiersprache soll JAVA angewendet werden, insbesondere beim zu nutzenden Spark JAVA Web Framework, sowie MongoDB als Datenbankanwendung.

#### 1.4.2 Personelle Schnittstellen

Herr Kazim Karadag (GF) als Projektgeber mit folgenden Aufgaben:

Projektdefinition, Projekttragende Entscheidungen, Projektabnahme.

Die genannten Personen sind mit der Nennung ihrer Namen einverstanden.

## 1.5 Projektabgrenzung

Meine Aufgabe wird es sein das Projekt selbstständig in einem Modul außerhalb des bestehenden Systems umzusetzen. Eine Integration in das bestehende System wird nach Abschluss geprüft. Um dies zu erleichtern hat der Projektgeber daher die Vorgaben bezüglich Programmiersprache, Framework und Datenbankanwendung getätigt.

## 2. Planungsphase

### 2.1 Projektziele, und –vorgaben

#### Projektziele:

- Analyse des Ist-Zustandes
- Finale Besprechung des Projektauftrags
- Herstellen der Arbeitsumgebung
- Planung des Projektes
- Auswahl der Software, zum Bsp. Framework, DB-Auswahl
- Modellierung der Datenbank
- Erstellen einer Upload-Möglichkeit für Teilnehmer
- Aufsetzen eines Webservers
- Planung und Implementierung der grafischen Benutzeroberfläche
- Implementierung der im Soll-Zustand beschriebenen Funktionalität
- Testen der Funktionen
- Erstellung eines Soll-Ist-Vergleiches
- Erstellung der Dokumentation

#### Projektvorgaben:

- JAVA als serverseitige Programmiersprache durch Anwendung des Spark JAVA Web Frameworks
- MongoDB als Datenbank

### 2.2 Analyse des Ist-Zustandes

Momentan müssen sich die Teilnehmer der Sprachschule Ihre Nachweise ausdrucken, diese unterschreiben lassen, wieder einscannen oder kopieren, um sie dann per Post oder E-Mail an die jeweilige Behörde zu senden. Dieser Prozess ist Zeit- und Personalaufwändig sowie fehleranfällig.

## 2.3 Projektbeteiligte Personen

Ausschließlich der Projektnehmer war an der Umsetzung des Projektes beteiligt.

## 2.4 SOLL-Zustand

Ziel ist es den Teilnehmern der Sprachschule die Möglichkeit anzubieten die entsprechenden Dokumente auf dem Portal der Schule an entsprechender Stelle (Zeitraum, Qualifikation) hochzuladen und in einer Datenbank abzuspeichern.

Der entsprechende Kostenträger des Teilnehmers soll daraufhin automatisch eine E-Mail als Information des Uploads erhalten. Diese E-Mail soll einen Link enthalten, über welchen der Kostenträger das entsprechende Dokument für seine eigene Verwendung herunterladen kann.

Eine eventuelle elektronische Signatur der Dokumente wird hierbei nicht betrachtet. Diese ist Gegenstand eines anderen Projektes.

## 2.5 Zeitplanung

### Planungsphase - 5,0 Stunden

- Planung Zeitpuffer (1,0 Stunden)
- Besprechungen für Planungsphase (1,0 Stunden)
- Analyse des Ist-Zustandes (1,0 Stunden)
- Softwareauswahl (1,0 Stunden)
- Ressourcenplanung (0,5 Stunden)
- Kostenplanung (0,5 Stunden)

### Durchführungsphase - 45,0 Stunden

- Planung Zeitpuffer (1,0 Stunden)
- Besprechungen für Durchführungsphase (3,0 Stunden)
- Herstellen der Arbeitsumgebung (1,0 Stunden)
- Implementierung der o.g. Funktionen (30,0 Stunden)
- Testen der o.g. Funktionen (10,0 Stunden)
- Refaktorisierung

### Übergabe-, Auswertungs-, und Dokumentationsphase - 20 Stunden

- Planung Zeitpuffer (1,0 Stunden)
- Besprechungen für Übergabe-, Auswertungs-, und Dokumentationsphase (2,0 Stunden)
- Übergabe des Systems (3,0 Stunden)
- Erstellung eines Soll-Ist-Vergleiches (4,0 Stunden)

- Erstellung einer persönlichen technischen Auswertung (3,0 Stunden)
- Erstellung der Dokumentation (8 Stunden)

Siehe Anhang: Gantt-Diagramm

## 2.6 Ressourcenplanung

Arbeitsplatz:	1 Schreibtisch mit Stromversorgung und Internetzugang, 1 Bürostuhl
Hardware:	1 PC, 2 Bildschirme min.24“, Maus, Tastatur,
Software:	Entwicklungsumgebung, E-Mail-Account, Browser, Office Paket, Framework

## 2.7 Personalplanung

Andere beteiligte Personen stehen außerhalb des Projektes und werden unter „1.4.2 Personelle Schnittstellen“ aufgeführt.

## 2.8 Kostenplanung

Abbildung 1: Kostenplanung

Position	Monatliches Gehalt	Stündliches Gehalt	Anzahl Stunden	Kosten
Geschäftsführer	6.000,00 €	37,50 €	2,00	75,00 €
Projektnehmer	1.185,00 €	7,40 €	71,50	529,10 €
<b>Personalkosten Gesamt</b>				<b>604,10 €</b>
Materialkosten pauschal				30,00 €
Mietkosten anteilig				120,00 €
Nebenkosten pauschal				40,00 €
<b>Gesamtkosten</b>				<b>794,10 €</b>

Das Gehalt des Geschäftsführers ist mir nicht bekannt und wurde daher von mir geschätzt.  
Da das Projekt im Homeoffice realisiert wurde habe ich meine eigenen Betriebskosten zu Grunde gelegt.

Somit belaufen sich die Gesamtkosten des Projektes auf **794,10 €**.

## 2.9 Abweichungen vom Projektantrag

In der Projektplanung gab es keine Abweichungen vom Projektantrag.

## 3. Durchführungsphase

### 3.1 Entscheidungen

Es wurde vorab vom Projektgeber entschieden, dass das Projekt außerhalb des bestehenden Systems selbstständig laufen soll.

### 3.2 Herstellen der Arbeitsumgebung

Einrichten des Arbeitsplatzes:

Stuhl, Bildschirme, Maus und Tastatur ergonomisch ausrichten

Einrichten des Rechners:

Laptop mit Windows 10 Pro vorinstalliert,

Installation der Entwicklungsumgebung IntelliJ IDEA Community Edition 2020.3,

Installation von MongoDB Server Version 4.4.2,

Installation Thunderbird Mail-Client

Installation Firefox und Chrome Browser

### 3.3 Diagramme

- Aktivitätsdiagramm: [Siehe Anhang](#)
- Use-Case-Diagramm: [Siehe Anhang](#)
- Gantt-Diagramm: [Siehe Anhang](#)

### 3.4 Implementierung der Funktionen

IntelliJ:

- Anlegen eines neuen JAVA-Projektes mit dem Namen “fileupload” mittels **Maven**
- Einfügen der Spark dependency in die **pom.xml**, ein aktiverter Autoimport erleichtert die Installation
- Um die Protokollierung des Webservers zu aktivieren, füge ich die org.slf4j - Abhängigkeit zum Projekt hinzu

<http://www.slf4j.org/codes.html>

- Anlegen der Main-Klasse

#### 3.4.1 Erstellen von Webseiten zur Simulation

Im Projektverzeichnis habe ich einen “html”-Ordner angelegt. In diesem speichere ich verschiedene Webseiten, welche bei Aufruf der entsprechenden Methode vom Webserver zur Verfügung gestellt werden. Hier können zum Beispiel für verschiedene Behörden angepasste Seiten hinterlegt werden. Diese Seiten simulieren den Informationsfluss ähnlich der künftigen Anwendung:

- *startUpload.html* – simuliert die Anmeldung des Teilnehmers am System

Siehe Anhang: Programmablauf - Screenshots Schritt 1

- *fileSelectionUpload.html* – vom Webserver bereitgestellt, gibt dem Teilnehmer die Vorgaben für die Dateiauswahl

Siehe Anhang: Programmablauf - Screenshots Schritt 2

- *startDownload.html* – simuliert das Anmelden der Behörde vor dem Download

Diese Seiten werden bei Aufruf in einen String umgewandelt und zurückgegeben. Hierfür habe ich die *htmlToString* - Methode geschrieben, welche ein HTML-Dokument in einen String umwandelt.

Dies vereinfacht es bei einem bestimmten post-Aufruf (teilnehmerabhängig) eine vordefinierte Webseite zurückzugeben. So können dem Teilnehmer spezifische Informationen zum Datei-Upload angezeigt werden wie z.Bsp. akzeptierte Dateiformate oder maximale Dateigröße.

Abbildung 2: *htmlToString* – Methode

```
// Methode zur Umwandlung einer Html-Seite in einen String
// append()-Methode: hiermit können Inhalte an das Ende einer bestehenden Datei anhängt werden
private static Object htmlToString(String s) {

    StringBuilder contentBuilder = new StringBuilder();
    try {
        BufferedReader in = new BufferedReader(new FileReader(s));
        String str;
        while ((str = in.readLine()) != null) {
            contentBuilder.append(str);
        }
        in.close();
    } catch (IOException e) {
    }
    String content = contentBuilder.toString();
    return content;
}
```

Um eine Datei hochzuladen, benötige ich entsprechende **form- und post-Handler** im HTML Body. Der form-Handler benötigt den richtigen **Enctype - multipart/form-data** sowie ein Eingabefeld mit dem Typ "file" und einem Namen meiner Wahl (im Projekt "uploaded\_file")

Abbildung 3: Ausschnitt aus *fileSelectionUpload.html*

```
<form class="" action="http://127.0.0.1:4567/UploadMongoDB" method="post" enctype="multipart/form-data">
    <input type="file" id="myfile" name="uploaded_file"><br><br>
    <input type="submit" value="Hochladen">
</form>
```

### 3.4.2 Erstellen der Routes des Webservers zum Upload

- Auf dem Webserver erstelle ich spezifische **Routes** welche dann auf den entsprechenden post-Aufruf des HTML-Dokumentes reagieren.

- Damit Spark in der Lage ist, eine hochgeladene Datei zu extrahieren, muss ich vorab ein spezielles Anfrageattribut (request) setzen: "org.eclipse.jetty.multipartConfig".

Dies wird vom implementierten Jetty–Webserver bereitgestellt.

Es gibt an, dass Instanzen des **Servlets** Anfragen erwarten, die dem MIME-Typ **multipart/form-data** entsprechen.

Servlets, die mit MultipartConfig annotiert sind, können die Part-Komponenten einer gegebenen multipart/form-data-Anfrage durch die Verwendung der getPart()-Methode auf die Rohanfrage (raw request) abrufen.

Abbildung 4: Beispiel einer post route, hier für die Initialisierung der Websites-Bereitstellung zur Dateiauswahl des Uploads

```
post( path: "/StartUpload", (req, res) -> {

    // Multipart-Konfiguration für die Formulardaten:
    req.attribute( attribute: "org.eclipse.jetty.multipartConfig", new MultipartConfigElement( location: "/temp"));

    // Kontrollausgabe für die Konsole, mit Session-ID für die Verlaufskontrolle
    System.out.println("Post /StartUpload - die Session mit der ID : "+ req.session().id() + " wurde gestartet");

    // Festlegen der SessionAttribute aus den form-values
    req.session().attribute( name: "nachname",req.queryParams("nachname"));
    req.session().attribute( name: "vorname",req.queryParams("vorname"));
    req.session().attribute( name: "id",req.queryParams("id"));

    // Bereitstellen der aktualisierten UploadWebSeite mit Aufruf der Methode
    return htmlToString( s: "html/fileSelectionUpload.html");
}
);
```

In diesem Projekt habe ich folgende post-routes erstellt:

- /startUpload - initialisiert den Uploadvorgang
- /UploadMongoDB - startet den eigentlichen Upload
- /DownloadMongoDB/:id - initialisiert den Downloadvorgang aus einem E-Mail-Link
- /startDownload - startet den eigentlichen Download

#### /startUpload

reagiert auf die erste Initialisierung des Uploadvorgangs durch den Teilnehmer und stellt diesem eine Webseite zur Verfügung (fileSelectionUpload.html) mittels der die Dateiauswahl zum Upload erfolgen kann. Weiterhin wird die Session Id abgefragt und mit den Session-Attributen wie „Nachname“, „Vorname“, „Teilnehmer-Id“ an der ServerKonsole zu Protokollzwecken ausgegeben. Bei einer Integration in das bestehende System kann hier auch das gehashte Passwort des Teilnehmers übernommen und geprüft werden.

## /UploadMongoDB

startet den eigentlichen Uploadvorgang. Die Sessionattribute werden wieder zu Protokollzwecken an die Konsole übergeben (Ist dies noch die gleiche Session?). Mit der vorab genannten getPart()Methode und Regex Befehlen werden die einzelnen Partkomponenten erfasst und zu einer neuen Dateipräfix und -suffix verarbeitet. Die Dateipräfix wird somit nicht dem Teilnehmer überlassen, sondern kann eindeutiger benannt werden. Zum Beispiel können hier Name, Vorname und Id mit eingearbeitet werden. (Durch Verschlüsselung müssen diese personenbezogenen Informationen dann bei der Emailübertragung geschützt werden). Mit

```
„String date Time = new SimpleDateFormat("yyyy-MM-dd-HH-mm").format(new Date());“
```

ist es auch möglich das Präfix einen Zeitstempel hinzuzufügen.

Die empfangenen Daten werden gestreamt und der Stream in einer neuen Datei lokal auf dem Webserver zwischengespeichert.

```
try (InputStream input = req.raw().getPart("uploaded_file").getInputStream())
      { Files.copy(input, tempFile,
      StandardCopyOption.REPLACE_EXISTING);}
```

Den Speicherort habe ich vorab durch Pfad, Präfix und Suffix definiert:

```
Path tempFile = Files.createTempFile (uploadDir.toPath(),pref,suf);
```

Nun wird die zwischengespeicherte Datei mit angepasster Bezeichnung an eine Funktion zum Speichern auf der MongoDB Datenbank übergeben:

```
fileObjekId = MongoDBInstance.init(tempFile,fileNameUpload);
```

Gleichzeitig erhalte ich die fileObjekId von der Datenbank zurück, welche ich später für die Erstellung des Download-Links in der E-Mail verwenden kann.

### 3.4.3 Erstellen der Methode zur Datenübergabe an MongoDB

```
public static ObjectId init(Path filepath, String filename)
```

Da es sich um eine Simulation handelt wird hier auf MongoDB mittels

```
MongoClient mongoClient = new MongoClient("localhost", 27017);
```

zugegriffen.

Mit `private static String DATABASE = "uploadserver";`

```
MongoDatabase database = mongoClient.getDatabase(DATABASE);
```

stelle ich die Verbindung zur gewünschten Datenbank “uploadserver” her.

Zur Visualisierung der Datenbank habe ich mich für “**robo3T**” entschieden. Damit kann ich schnell prüfen, ob die Daten an der gewünschten Stelle der Datenbank gespeichert werden.

Sollten die Datei die **BSON**-Dokumentengröße von 16 MB überschreiten, ist es nötig die Daten mittels **GridFS** zu übertragen.

```
GridFSBucket gridFSFilesBucket = GridFSBuckets.create(database, "uploadfiles");
```

Hiermit wird ein gridFSBucket(Sammelbehälter) mit dem Namen "uploadfiles" erstellt.

Abbildung 5: Übersicht der Datenbank mit robo3T

The screenshot shows the Robo 3T application window. On the left, there's a sidebar with a tree view of connections and collections. A connection named 'uploadserver' is selected, showing its collections: 'uploadfiles.chunks' and 'uploadfiles.files'. The 'uploadfiles.files' collection is currently selected. On the right, a main panel displays the results of a MongoDB query: 'db.getCollection('uploadfiles.files').find({})'. The results are presented in a table with columns 'Key', 'Value', and 'Type'. There are 14 rows, each representing a document in the collection, with IDs ranging from '602e64abedbb985f8cd8b298' to '605c5155084dfa3c04b3253e'.

Key	Value	Type
> (1) ObjectId("602e64abedbb985f8cd8b298")	{ 7 fields }	Object
> (2) ObjectId("60376f9799d7e53ded20c83e")	{ 7 fields }	Object
> (3) ObjectId("605a4ba15830265f90136be1")	{ 7 fields }	Object
> (4) ObjectId("605b025ca9f6303892a0a2fc")	{ 7 fields }	Object
> (5) ObjectId("605b2d6d32c17929091dcba7")	{ 7 fields }	Object
> (6) ObjectId("605ba4d85cce6a6070515de2")	{ 7 fields }	Object
> (7) ObjectId("605ba5efc4a3a52149c1d329")	{ 7 fields }	Object
> (8) ObjectId("605ba65aadd3de3cdb7d3c49")	{ 7 fields }	Object
> (9) ObjectId("605ba91ec1b9c07d39175bd7")	{ 7 fields }	Object
> (10) ObjectId("605baaa4fd79196378ae8f5e1")	{ 7 fields }	Object
> (11) ObjectId("605bac4dd79196378ae8f5e4")	{ 7 fields }	Object
> (12) ObjectId("605c37ea0301ae28d96f27ba")	{ 7 fields }	Object
> (13) ObjectId("605c394b4f14326a1fec32e8")	{ 7 fields }	Object
> (14) ObjectId("605c5155084dfa3c04b3253e")	{ 7 fields }	Object

In einer Try-Catch-Exception wird mittels InputStream aus den übergebenen Parametern (temporärer Speicherort auf dem Webserver, Dateiname) ein neues Dateiobjekt erzeugt und in der Datenbank abgelegt. Die Id des neuen MongoDB-Objektes wird in fileID gespeichert, einem Objekt der ObjectId-Klasse. Dies ist ein global eindeutiger Bezeichner für Objekte bestehend aus 12 Bytes.

[Siehe „Methode zur Datenübergabe an MongoDB“ im Anhang](#)

Anschließend wird diese Id in einen String umgewandelt und an den Methodenaufruf zurückgegeben.

### 3.4.4 Erstellen einer Emailfunktion

Vorab mussten folgende JAVA-Bibliotheken importiert werden:

- javax.mail\*
- javax.mail.internet\*

Für die künftige Betreffzeile der E-Mail habe ich den String "sub" erstellt. Dieser enthält verschiedene zwischengespeicherte Werte, welche der Teilnehmer beim Upload in der Session übergeben hat, wie zum Beispiel Name, Vorname und TeilnehmerNr.

Der Inhalt der E-Mail wird durch den String "content" definiert. Hierbei handelt es sich um HTML-Code welcher in den String eingebunden und durch die entsprechende ObjectId für den Download-Link ergänzt wird.

Mit Aufruf der Methode

```
Mailer.send(from,password,to,sub,content)
```

wird der Versand der E-Mail angestoßen.

Mit der Methode `send()` wird ein neues **Properties-Objekt** erzeugt, welchem verschiedene Eigenschaften zum Mailserver, wie `smtp-host` und `port`, zugewiesen werden können.

Diese Eigenschaften werden an ein neues **Session-Objekt** übergeben. Die Klasse Session stellt eine Mail-Sitzung dar.

```
Session session = Session.getDefaultInstance(props,  
new javax.mail.Authenticator() {  
    protected PasswordAuthentication getPasswordAuthentication() {  
        return new PasswordAuthentication(from,password);  
    }  
});
```

`Session.getDefaultInstance(...)` holt das Standard-Session-Objekt. Wenn noch kein Standard eingerichtet wurde, wird ein neues Sitzungsobjekt erstellt und als Standard installiert.

Da die Standardsitzung potenziell für den gesamten Code verfügbar ist, der in derselben virtuellen JAVA-Maschine ausgeführt wird, und die Sitzung sicherheitsrelevante Informationen wie Benutzernamen und Kennwörter enthalten kann, ist der Zugriff auf die Standardsitzung eingeschränkt. Das Authenticator-Objekt, das vom Aufrufer erstellt werden muss, wird indirekt zur Überprüfung der Zugriffsberechtigung verwendet. Das Authenticator-Objekt, das beim Erstellen der Sitzung übergeben wird, wird mit dem Authenticator-Objekt verglichen, das bei nachfolgenden Anfragen übergeben wird, um die Standardsitzung zu erhalten. Wenn beide Objekte gleich sind oder aus demselben ClassLoader stammen, wird die Anforderung zugelassen. Andernfalls wird sie verweigert.

In einer Try-Catch-Exception wird im Anschluss ein MIME-Message Objekt erzeugt und mit Werten aus dem Methodenaufruf wie sub (Betreff) und content (E-Mail-Inhalt) befüllt.

```
MimeMessage message = new MimeMessage(session);
```

```
message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));  
message.setContent(content,"text/html" );
```

Mit

```
Transport.send(message);
```

wird der eigentliche E-Mail-Versand ausgelöst.

*Transport* ist eine abstrakte Klasse, die einen Nachrichtentransport modelliert. Die Nachricht wird an alle in der Nachricht angegebenen Empfängeradressen (wie von der Message-Methode *addRecipients* zurückgegeben) gesendet.

In der Konsole erfolgt eine Ausgabe über den erfolgreichen E-Mail-Versand.

[Siehe Anhang: Programmablauf - Screenshots Schritt 3](#)

Zu Testzwecken wurde ein Gmail-Account als Simulations-Mailserver sowie eine weitere private E-Mail-Adresse als Empfänger genutzt.

Als return der Route “**/UploadMongoDB**” erhält der initialisierende Teilnehmer eine Bestätigung für den erfolgreichen Upload in Form einer HTML-Ausgabe mit dem Dateinamen und er Ansicht des Dokumentes.

Verbunden ist dieser Vorgang mit einer weiteren Testausgabe der *ObjektId* in der Konsole.

### **3.4.5 Erstellen von Webserver Routes zum Download**

Mit der Aktivierung des Download-Links in der E-Mail

[Siehe Anhang: Programmablauf - Screenshots Schritt 4](#)

startet der Empfänger die

“**/DownloadMongoDB/:id**” Route auf dem Webserver. Die *ObjektId* des MongoDB-Objektes wird mit übernommen und auch hier zum Test in der Konsole ausgegeben. Als return wird die “*startDownload.html*” Webseite ausgegeben, über welche sich die empfangende Behörde identifizieren kann.

[Siehe Anhang: Programmablauf Screenshots Schritt 5](#)

Ist dies erfolgt wird über den Login-Button dieser Webseite die Route “**/startDownload**” aktiviert.

Hier können später die Session-Daten wie Password und Behörden-Id abgegriffen werden und zur Zugangsüberprüfung verwendet werden. Da die Struktur der vorhandenen Datenbank nicht bekannt war, wurde von mir hier bewusst auf diesen Schritt verzichtet. Es wird lediglich geprüft, ob ein Zugriff auf den gewählten Speicherort der Datenbank erfolgt ist.

### 3.4.6 Erstellen einer Methode zur Datenübernahme aus MongoDB

Mit: `Object filename = MongoDBInstance.down(objID)`

wird die Methode `public static Object down(String downObjectId) {}` aufgerufen und ihr eine ObjectId aus dem download Link der E-Mail übergeben bzw. der dadurch bereitgestellten Zwischenseite „`startDownload.html`“.

Diese Methode funktioniert ähnlich der Methode zum Upload auf MongoDB, nur halt in die entgegengesetzte Richtung. Es wird wiederum ein Stream erzeugt und lokal mit übernommenem Prä- und Suffix auf dem Webserver im Ordner „public/download/“ zwischengespeichert. Der return-Wert ist der Dateiname als String.

[Siehe „Methode zur Datenübernahme aus MongoDB“ im Anhang](#)

Final wird die entsprechende Datei dann vom Webserver der Behörde bereitgestellt.

[Siehe Anhang: Programmablauf Screenshots Schritt 6 und 7](#)

## 3.5 Test

Getestet wurde ausschließlich in der Simulation bereits während der Erstellung. Hierzu habe ich an ausgewählten Punkten des Programms Ausgaben für die Konsole der IDE eingefügt. Zum Beispiel die permanente Ausgabe der Session-Id, um sicherzustellen, dass der komplette Datenfluss noch in der gleichen Session stattfindet. Ebenso werden empfangene und übergebene Werte in der Konsole ausgegeben. Dies entspricht der Vorgehensweise eines Unitest / Modultest, der untersten Teststufe und Grundlage für spätere Integrationstests.

Abbildung 6: Anweisungen für die Konsolenausgabe zu Testzwecken

```
System.out.println("Post /UploadMongoDB gestartet - die Session-ID lautet: " + req.session().id());  
  
System.out.println("Alle Attribute der Session: " + req.session().attributes());  
  
String nachname = ((String) req.session().attribute( name: "nachname"));  
String vorname = ((String) req.session().attribute( name: "vorname"));  
String id = ((String) req.session().attribute( name: "id"));  
  
System.out.println("Nachname: " + nachname);  
System.out.println("Vorname: " + vorname);  
System.out.println("Id: " + id);
```

[Siehe Anhang: Testausgabe Konsole Webserver](#)

[Siehe Anhang: Testausgabe MongoDB mittels robo3T](#)

### 3.6 Refaktorisierung

Der Programmcode wurde nach dem Test von mir nochmals manuell in seiner Struktur geprüft. Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit wurden verbessert, mit dem Ziel, den jeweiligen Aufwand für Fehleranalyse und funktionale Erweiterungen zu senken. Dabei habe ich das Programmverhalten beobachtet, um die Funktionalität nicht negativ zu beeinflussen.

### 3.7 Einrichtung

Die Simulation wurde an den Projektgeber übergeben, dieser prüft eine Weiterführung des Projektes, um zum Beispiel die Verschlüsselung zu integrieren. Erst danach kann über eine Einbindung in das bestehende Livesystem entschieden werden.

## 4. Übergabe, Auswertungs-, und Dokumentationsphase

### 4.1 Dokumentation

Diese Dokumentation wurde fortlaufend während des Projektes (prozessorientiert) erstellt. Während der Programmierphasen wurde die Kommentarfunktion der IDE genutzt. Die Kommentare wurden dann nach erfolgreichen Tests in die Dokumentation übernommen.

### 4.2 Soll-Ist-Vergleich - zeitlich

Abbildung 7: Zeitlicher Soll-Ist-Vergleich

Arbeitsschritt	Geplante Zeit (h)	Benötigte Zeit (h)	Differenz (h)
<b>Planungsphase</b>			
Zeitpuffer	1,00	0,00	-1,00
Besprechungen für Planungsphase	1,00	1,00	0,00
Analyse des Ist-Zustandes	1,00	1,00	0,00
Softwareauswahl	1,00	1,00	0,00
Ressourcenplanung	0,50	0,50	0,00
Kostenplanung	0,50	1,00	<b>0,50</b>
<b>Durchführungsphase</b>			
Zeitpuffer	1,00	0,00	-1,00
Besprechungen für Durchführungsphase	2,00	0,50	<b>-1,50</b>
Herstellen der Arbeitsumgebung	1,00	1,50	<b>0,50</b>
Implementierung der Funktionen	30,00	45,00	<b>15,00</b>
Testen der Funktionen	10,00	8,00	<b>-2,00</b>
Refaktorisierung	1,00	1,00	0,00
<b>Übergabe-, Auswertungs-, und Dokumentationsphase</b>			
Zeitpuffer	1,00	0,00	-1,00
Besprechungen für diese Phase	2,00	0,50	<b>-1,50</b>
Übergabe des Systems	3,00	0,50	<b>-2,50</b>
Erstellung eines Soll-Ist-Vergleiches	4,00	1,50	<b>-2,50</b>
Erstellung einer persönlichen technischen Auswertung	2,00	0,50	<b>-1,50</b>
Erstellung der Dokumentation	8,00	8,00	0,00
<b>Gesamt</b>	<b>70,00</b>	<b>71,50</b>	<b>1,50</b>

Die im Projektantrag aufgeführte Zeitplanung hat sich während der Durchführung des Projektes nur teilweise als realistisch herausgestellt. Vor allem die Implementierung der Funktionen hat mehr Zeit in Anspruch genommen da ich viel Zeit in die Recherche investieren musste. Der geplante Zeitpuffer war für diese Phase zu gering.

Da die Planung der Übergabe und Auswertung hingegen sehr großzügig war wirkte sich dies allerdings nur gering auf das Ende des Projektes aus.

#### 4.3 Datenschutzrechtliche Betrachtung

Sowohl E-Mail-Adressen, die den Vor- und Nachnamen einer Person enthalten, als auch E-Mail-Adressen, die aus reinen Phantasienamen oder sonstigen Kürzeln bestehen, sind personenbezogene Daten. Ihre Erhebung, Verarbeitung und Nutzung unterliegt somit den strengen Vorgaben der DSGVO!

E-Mails mit besonders sensiblen Daten, wie Personaldaten und Geschäftsgeheimnissen müssen verschlüsselt werden.

Eine unverschlüsselte E-Mail ist vergleichbar mit einer Postkarte. Genauso wie eine Postkarte durch die Hände mehrerer Postangestellten geht, ist auch eine E-Mail auf dem Weg durchs Netz potenziell lese- und veränderbar.

Durch E-Mail-Verschlüsselung wird der Inhalt Ihrer Kommunikation für Dritte unlesbar.

Eine Möglichkeit dies zu realisieren wäre das Verfahren der RSA-Verschlüsselung. Dabei werden private und öffentliche Schlüssel eingesetzt. Die abgehende Nachricht wird mit dem öffentlichen Schlüssel des Empfängers (hier der entsprechenden Behörde) verschlüsselt. Dieser kann die Nachricht dann mit seinem privaten Schlüssel decodieren und die Nachricht lesen.

Die Umsetzung der Verschlüsselung war nicht Bestandteil dieses Projektes. Das Modul läuft nur zu Testzwecken außerhalb des bestehenden Systems. Die automatisierte Generierung der E-Mail beim Upload stand hier im Vordergrund. Bei einer Integration in das bestehende System wird eine E-Mail-Verschlüsselungsverfahren zum Einsatz kommen.

Welche Art der Verschlüsselung implementiert wird ist unter anderem abhängig von Auflagen bzw. Möglichkeiten der empfangenden Behörden.

#### 4.4 Ausblick

Das Projekt wurde online an den Projektgeber übergeben, eine persönliche Übergabe und Besprechung war pandemiebedingt nicht möglich. Der Projektgeber prüft ob die Funktionen kompatibel sind und das Projekt weitergeführt werden kann. Hierfür müssten dann weitere Informationen von entsprechenden Behörden bezüglich Verschlüsselung des Emailverkehrs, Dateiformate, Dateibenennung u.a. eingeholt werden.

Eine Entscheidung hierüber stand zum Ende der Dokumentationserstellung noch aus.

## 4.5 Rückblick (Fazit)

Um das Projekt noch genauer auf das bestehende System auszurichten hätten vorab mehr Details, z.B. zur Datenbankstruktur, bekannt sein müssen. Die Kommunikation mit Projektgeber stellte sich, auch pandemiebedingt, als schwierig dar.

Hieraus ergab sich ein erhöhter Eigenaufwand an Recherche und Ausprobieren. Einigen Funktionen liegen Vermutungen zum Aufbau des bestehenden Systems zu Grunde. Das gesamte Projekt wurde nahezu in Eigenregie im Homeoffice entwickelt.

Eine zeitliche Verschiebung einzelner Projektschnitte war die Folge.

Auch wegen des hohen Eigenaufwandes fand ich das Projekt sehr spannend. Die Projektplanung mit der Aufteilung in die einzelnen Projektphasen hat mir sehr geholfen.

Weiterhin finde es sehr interessant den Datenstrom vom Front- bis zum Backend zu verfolgen.

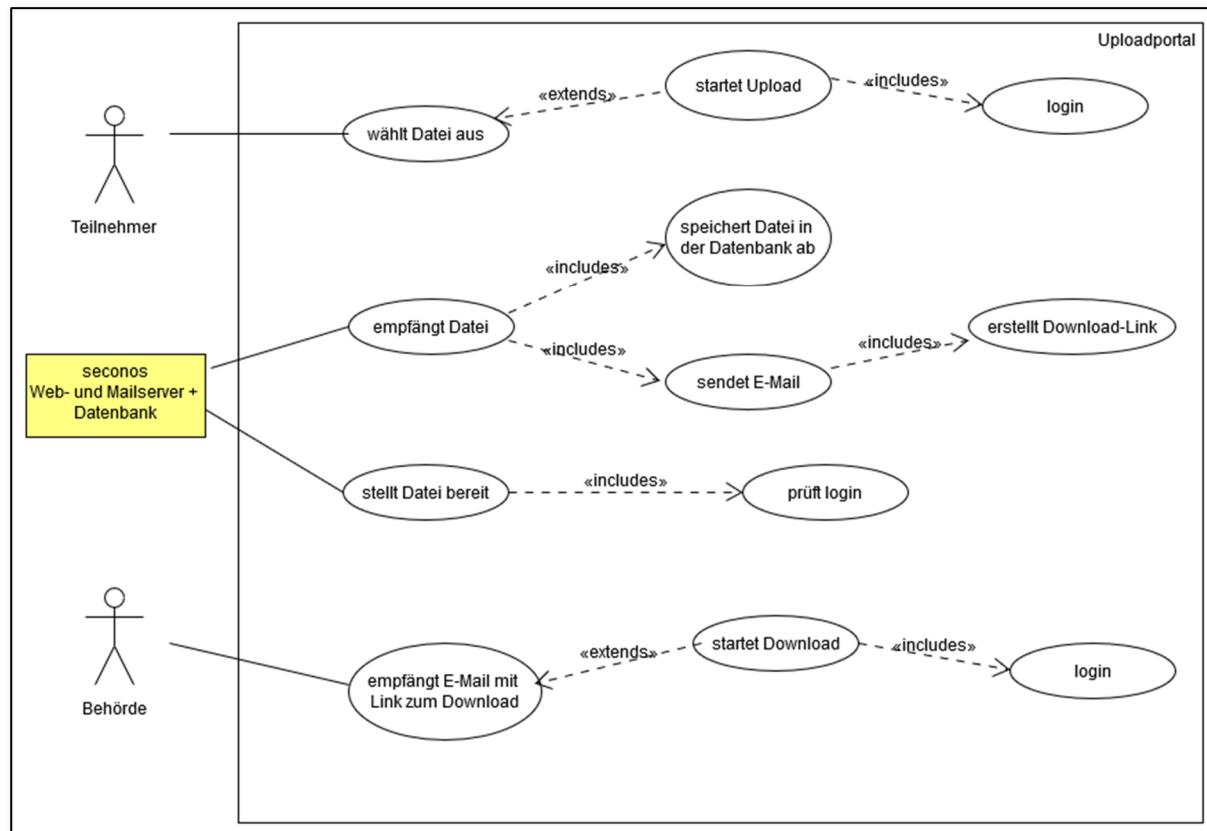
## 4.6 Persönliche technische Auswertung

Eine große Herausforderung war sowohl die Einarbeitung in ein mir völlig unbekanntes Framework als auch das Arbeiten mit einer NoSQL-Datenbank. Zu Beidem habe ich über viel Recherche und in kleinen Schritten Zugang gefunden.

## Anhang

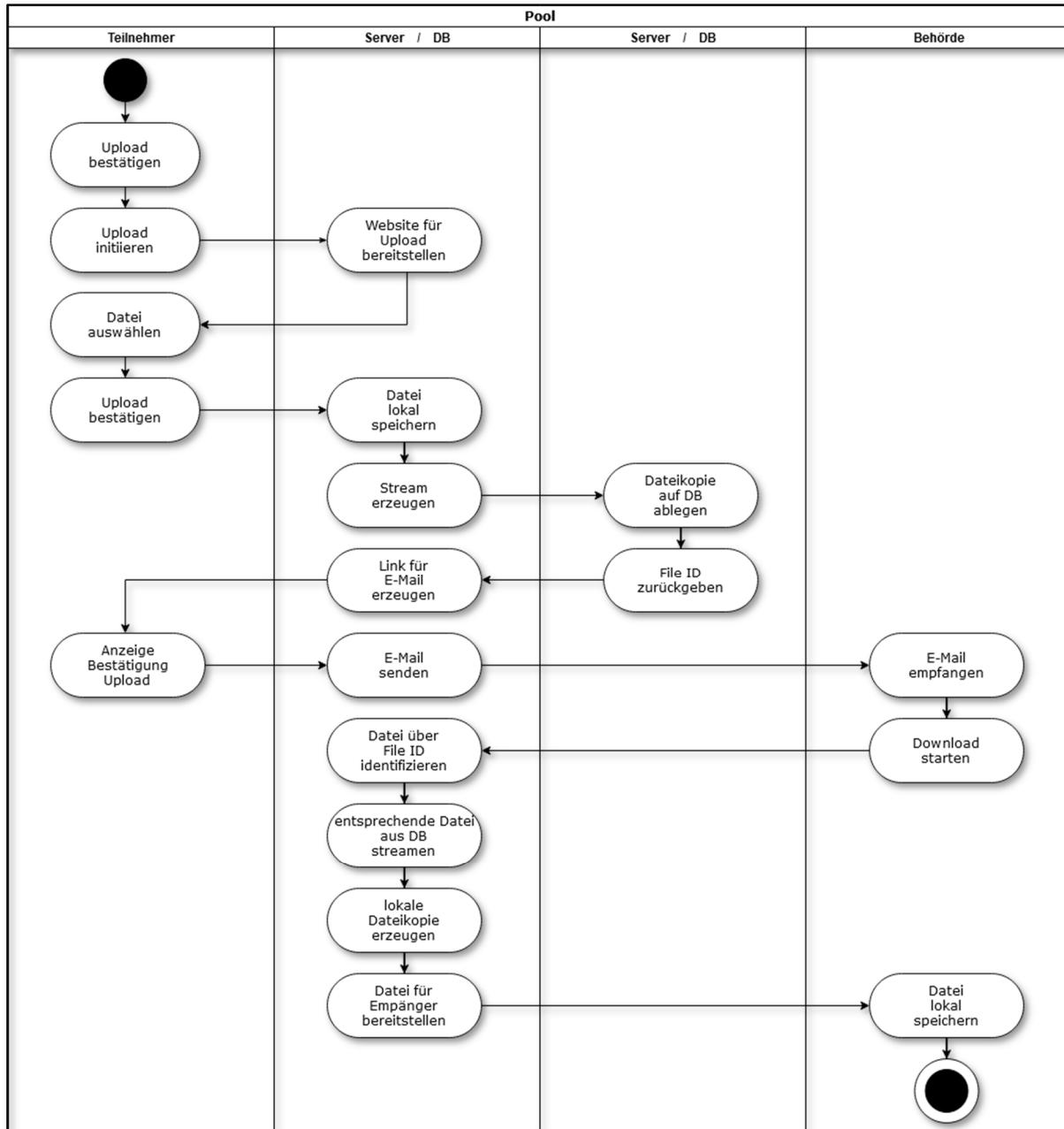
### Use-Case-Diagramm

Abbildung 8: Use-Case-Diagramm



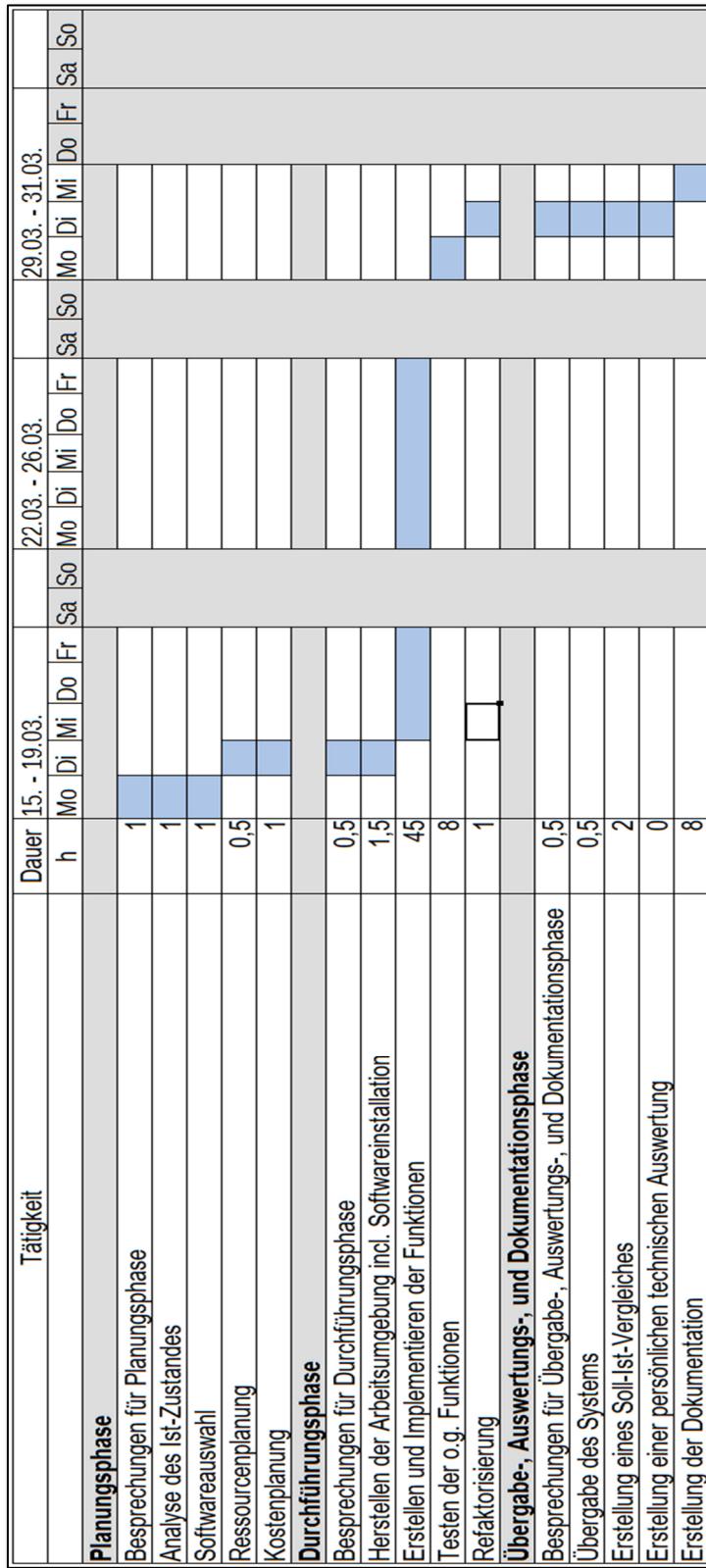
## Aktivitätsdiagramm

Abbildung 9: Aktivitätsdiagramm



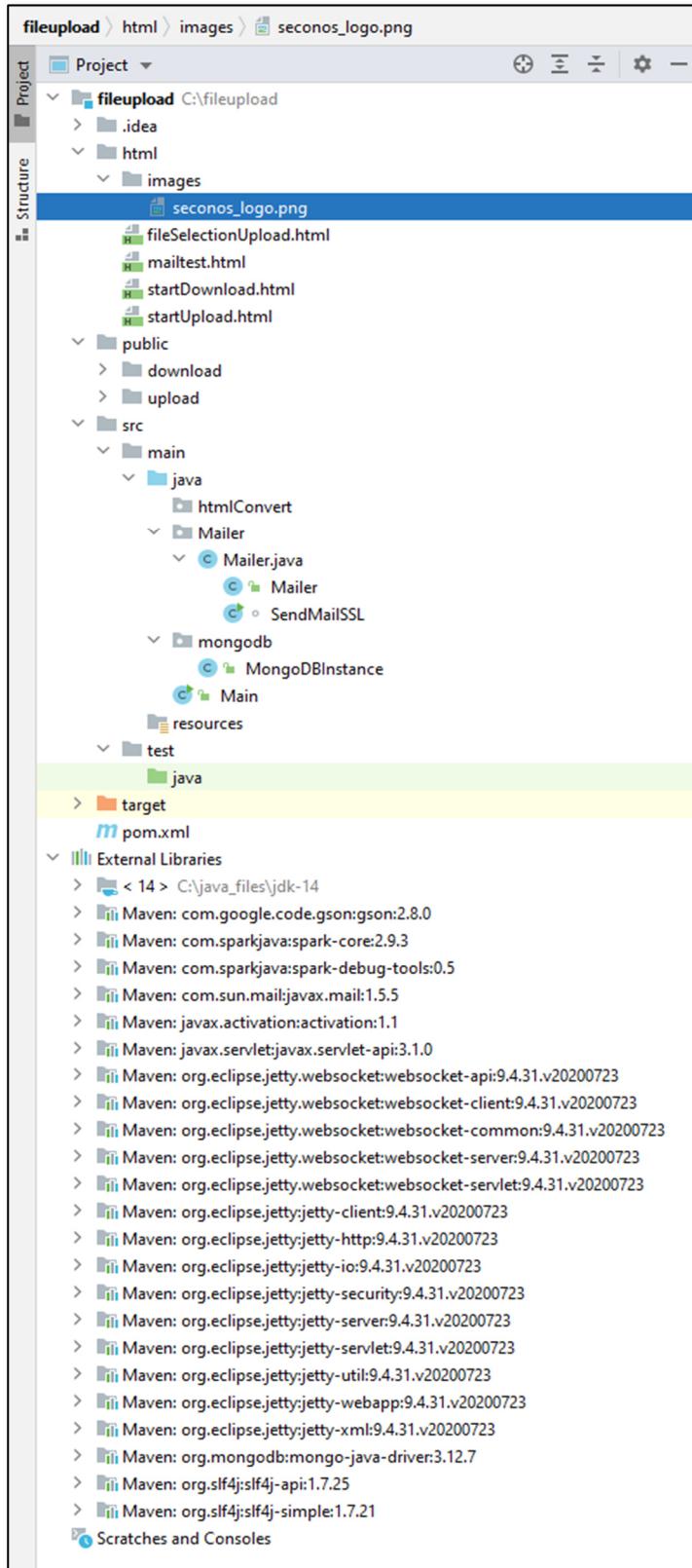
## Gantt – Diagramm

Abbildung 10: Gantt -Diagramm



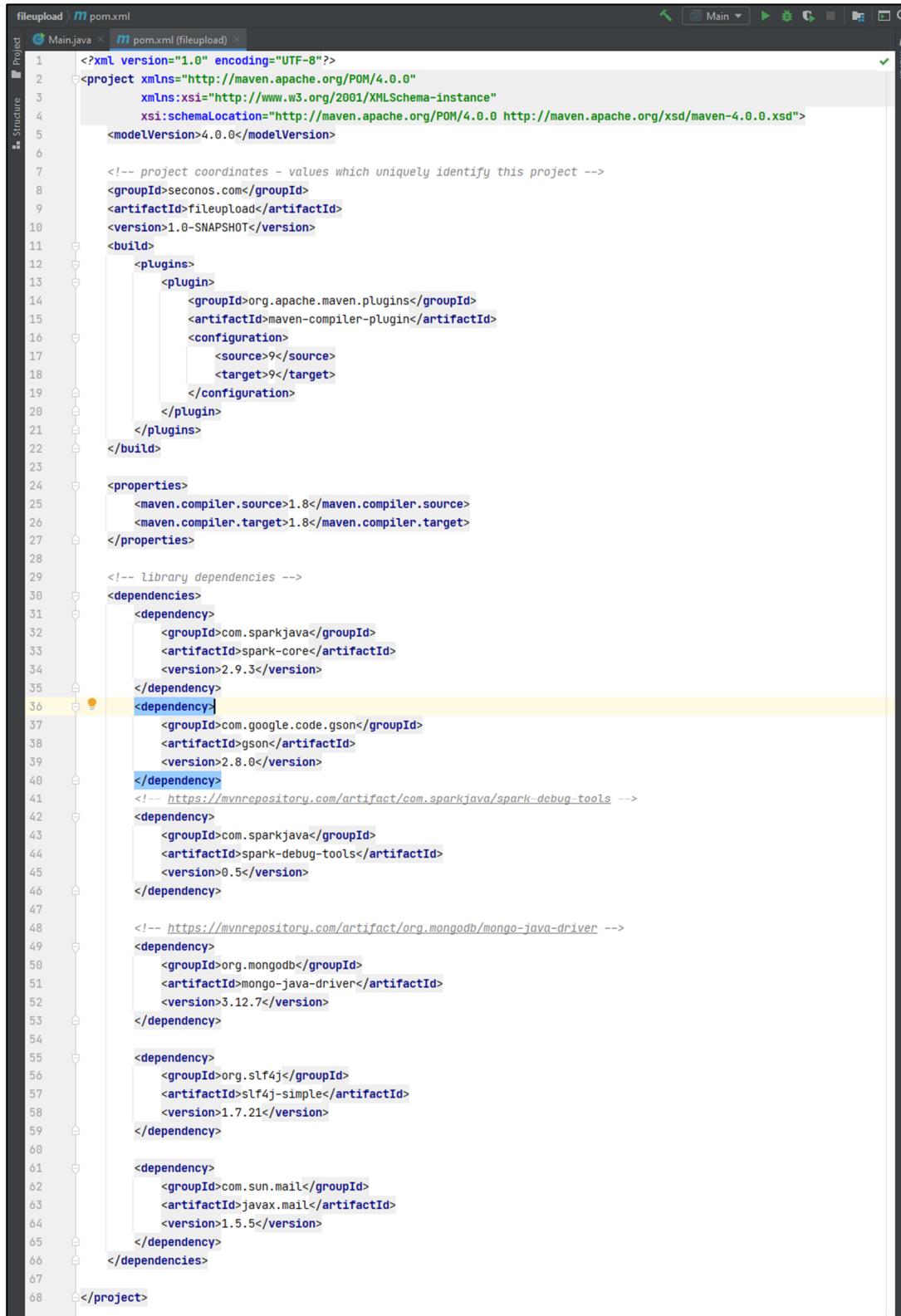
## IDE – Projektstruktur

Abbildung 11: IDE - Projektstruktur



## IDE - pom.xml – dependencies

Abbildung 12: IDE - pom.xml - dependencies



The screenshot shows an IDE interface with the pom.xml file open. The code editor displays the XML structure of a Maven project. Several sections of the XML code are highlighted with yellow background, specifically the build section and the dependency sections. The build section includes configuration for the maven-compiler-plugin. The dependency sections list various libraries used in the project, such as spark-core, gson, and mongo-java-driver. The code is well-formatted with proper indentation and syntax highlighting.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <!-- project coordinates - values which uniquely identify this project -->
  <groupId>seconos.com</groupId>
  <artifactId>fileupload</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>9</source>
          <target>9</target>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <!-- library dependencies -->
  <dependencies>
    <dependency>
      <groupId>com.sparkjava</groupId>
      <artifactId>spark-core</artifactId>
      <version>2.9.3</version>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.sparkjava/spark-debug-tools -->
    <dependency>
      <groupId>com.sparkjava</groupId>
      <artifactId>spark-debug-tools</artifactId>
      <version>0.5</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver -->
    <dependency>
      <groupId>org.mongodb</groupId>
      <artifactId>mongo-java-driver</artifactId>
      <version>3.12.7</version>
    </dependency>

    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>1.7.21</version>
    </dependency>

    <dependency>
      <groupId>com.sun.mail</groupId>
      <artifactId>javax.mail</artifactId>
      <version>1.5.5</version>
    </dependency>
  </dependencies>
</project>
```

## Methode zur Datenübergabe an MongoDB

Abbildung 13: Methode zur Datenübergabe an MongoDB

```
public class MongoDBInstance {  
    private static String DATABASE = "uploadserver";  
  
    public static ObjectId init(Path filepath, String filename) {  
        // Verbindung zum lokalen MongoDB-Server wird hergestellt  
        MongoClient mongoClient = new MongoClient( host: "localhost", port: 27017);  
        // Verbindung mit der Datenbank namens "uploadserver"  
        MongoDatabase database = mongoClient.getDatabase(DATABASE);  
        // GridFS  
        // Erstellt einen gridFSBucket(Sammelbehälter) mit einem benutzerdefinierten Bucket-Namen "uploadfiles".  
        GridFSBucket gridFSFilesBucket = GridFSBuckets.create(database, bucketName: "uploadfiles");  
        ObjectId fileId = null;  
        //Upload  
        //!!! Neue Datei erstellen aus InputStream,....  
        try {  
            InputStream streamToUploadFrom = new FileInputStream(new File( pathname: ""+filepath+""));  
            // Parameter für die Dateispeicherung vergeben  
            GridFSUploadOptions options = new GridFSUploadOptions()  
                .chunkSizeBytes(358400)  
                .metadata(new Document("type", "TestUploadFile" ));  
  
            fileId = gridFSFilesBucket.uploadFromStream(filename,streamToUploadFrom,options);  
            System.out.println("Uploaded file id : " +fileId);  
  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        } finally {  
        }  
        //return String.valueOf(fileId);  
        return fileId;  
    }  
}
```

## Methode zur Datenübernahme aus MongoDB

Abbildung 14: Methode zur Datenübernahme aus MongoDB

```
52 @ ...
53
54     public static Object down(String downObjectId) {
55
56         ObjectId mongoObjId = new ObjectId(downObjectId);
57
58         MongoClient mongoClient = new MongoClient( host: "localhost", port: 27017 );
59         MongoDatabase database = mongoClient.getDatabase("DATABASE");
60         GridFSBucket gridFSFilesBucket = GridFSBuckets.create(database, bucketName: "uploadfiles");
61
62         // Dateinamen aus ObjectId ermitteln
63         MongoCollection<Document> collection = database.getCollection( collectionName: "uploadfiles.files" );
64         System.out.println("Anzahl der Dokumente in 'uploadfiles.files': " + collection.countDocuments());
65
66         // Abfrage der Objekt-Daten mittels übergebener Object-Id:
67         BasicDBObject searchQuery = new BasicDBObject();
68         searchQuery.put("_id",mongoObjId);
69         Document myDoc = collection.find(searchQuery).first();
70
71         // Ausgabe im JSON-Format
72         System.out.println("Datensatz des Downloadobjektes im JSON-Format: " + myDoc.toJson());
73
74         // Auslesen des Dateinamens und speichern als String
75         String filename = myDoc.get("filename").toString();
76         System.out.println("filename" aus dem JsonObject: " + filename);
77
78         // Download per OutputStream und übergebener Objekt Id
79         try {
80             FileOutputStream streamToDownloadTo = new FileOutputStream( name: "public/download/" + filename );
81             gridFSFilesBucket.downloadToStream(mongoObjId, streamToDownloadTo);
82             streamToDownloadTo.close();
83             // System.out.println(streamToDownloadTo.toString());
84         } catch (IOException e) {
85             e.printStackTrace();
86         }
87
88         return filename;
89     }
90 }
```

## Testausgaben

### Testausgabe Konsole Webserver

Abbildung 15: Entsprechende Ausgabe der Konsole mit Session-Id und -Attributen

```
Uploaded file id : 605b2d6d32c17929091dcba7
2
Objekt ID für Download-Link: 605b2d6d32c17929091dcba7
Email wurde erfolgreich versandt
ObjectId der Uploaddatei (return von MongoDBInstance.init()) 605b2d6d32c17929091dcba7
Uploaded file 'Datei_A.png' saved as 'public\upload\Datei_A_2021-03-24-13-15_15881158804767827298.png'
[qtp2080171168-21] INFO spark.http.matching.MatcherFilter - The requested route [/Datei_A_2021-03-24-1
Objekt-ID aus Email übergeben: 605b2d6d32c17929091dcba7
post /DownloadMongoDB - Session ID : node0gtw13fjcl9fvgbzextbt3091
[qtp2080171168-21] INFO spark.http.matching.MatcherFilter - The requested route [/favicon.ico] has not
post /startDownload - Session ID : node0gtw13fjcl9fvgbzextbt3091
Alle Attribute der Session: [behoerde, objID, password]
Behoerde: Test
Passwort: 0815
ObjectId: 605b2d6d32c17929091dcba7
```

### Testausgabe MongoDB mittels robo3T

Abbildung 16: Ausgabe von robo3T zur Prüfung der Object-Id , filename und metadaten

The screenshot shows the robo3T interface with the following details:

- Connection:** New Connection, localhost:27017, uploadserver
- Query:** db.getCollection('uploadfiles.files').find({})
- Results:** A table showing the structure of the document found in the collection.

Key	Value	Type
> (1) ObjectId("602e64abedbb985f8cd8b298")	{ 7 fields }	Object
> (2) ObjectId("60376f9799d7e53ded20c83e")	{ 7 fields }	Object
> (3) ObjectId("605a4ba15830265f90136be1")	{ 7 fields }	Object
✓ (4) ObjectId("605b025ca9f6303892a0a2fc")	{ 7 fields }	Object
↳ _id	ObjectId("605b025ca9f6303892a0a2fc")	ObjectId
↳ filename	Datei_A.png	String
↳ length	131111	Int64
↳ chunkSize	358400	Int32
↳ uploadDate	2021-03-24 10:11:56.161+01:00	Date
↳ md5	7633cbd36604be54ccfd1b9096a665d	String
↳ metadata	{ 1 field }	Object
↳ type	TestUploadFile	String

## Programmablauf Screenshots

Abbildung 17: Programmablauf - Schritt 1 - Der Teilnehmer initialisiert nach Anmeldung den Uploadvorgang

The screenshot shows a web browser window titled "Uploadstart". The address bar indicates the URL is "C:/fileupload/html/startUpload.html". The main content area has a light blue background and displays the following text in large blue letters: "Teilnehmer initialisiert Dateiupload mit Anmeldung". Below this text is the SECONOS logo. The form fields are as follows:

- Vorname\*: Frank
- Nachname\*: Burkert
- ID: 12345
- A button labeled "Dokument hochladen"

Abbildung 18: Programmablauf -Schritt 2 - Der Webserver stellt dem Teilnehmer eine angepasste Seite zur Dateiauswahl

The screenshot shows a web browser window with the URL "127.0.0.1:4567/StartUpload". The main content area has a light blue background and displays the following text in large blue letters: "Vom webserver generierte Antwortseite für die Dateiauswahl". Below this text are two buttons:

- Datei auswählen (disabled)
- Hochladen

Below the buttons is a warning message:

Achtung! Dateien dürfen nur im pdf-Format  
und mit einer maximalen Dateigröße von 3MB  
hochgeladen werden!

## Programmablauf – Fortsetzung

Abbildung 19: Programmablauf - Schritt 3 - Der Absender erhält eine Bestätigungsanzeige für den erfolgten Upload



Abbildung 20: Programmablauf - Schritt 4 - Der Empfänger erhält eine E-Mail mit Teilnehmerdaten und einem Link zum Download



## Programmablauf – Fortsetzung

Abbildung 21: Programmablauf - Schritt 5 – Der Empfänger hat den Link geöffnet und muss sich nun identifizieren

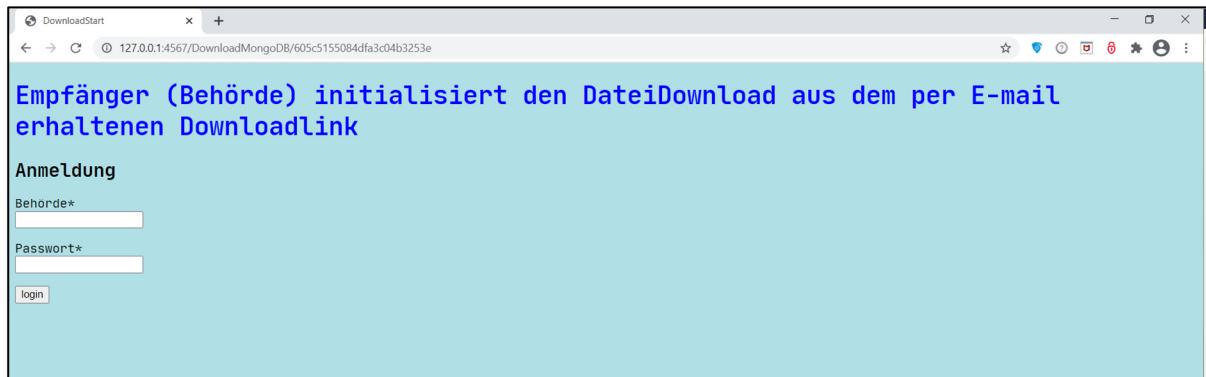
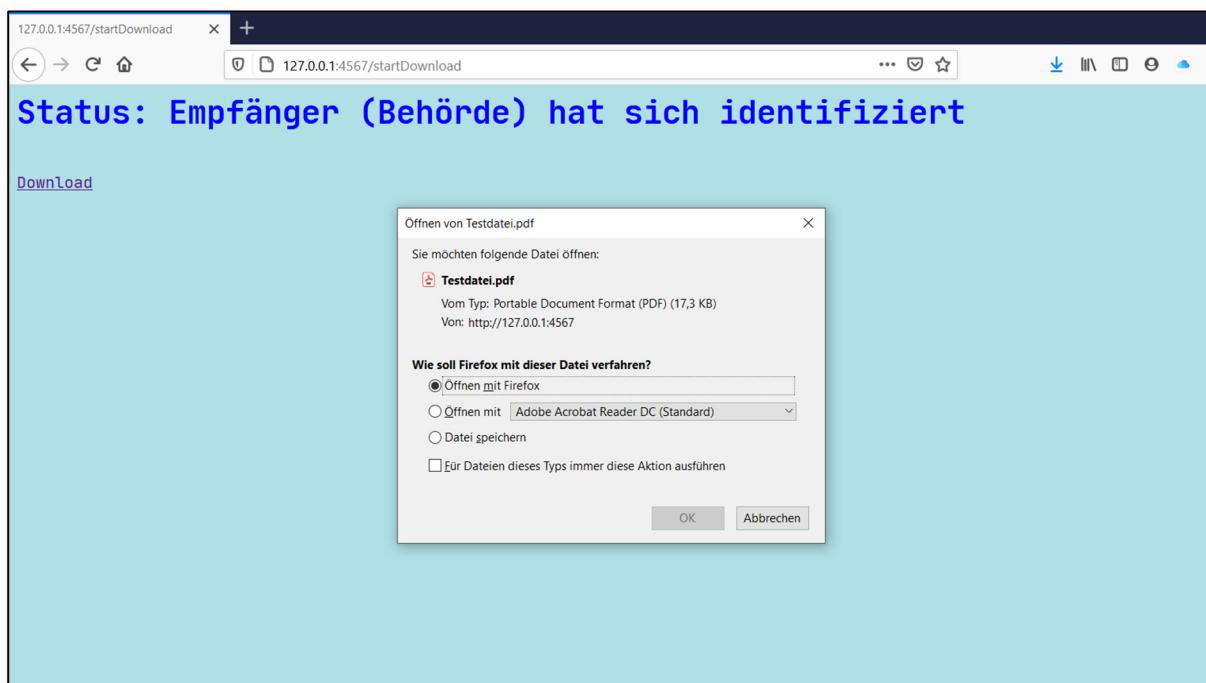


Abbildung 22: Programmablauf - Schritt 6 – Der Empfänger hat sich identifiziert und der Download wurde freigegeben



## Programmablauf – Fortsetzung

Abbildung 23: Programmablauf - Schritt 7 – Anzeige des vom Empfänger heruntergeladenen Dokumentes



## Glosar

### BSON

BSON steht einfach für "Binary JSON". Die binäre Struktur von BSON kodiert Typ- und Längeninformationen, wodurch es viel schneller geparsst werden kann. Die Kodierung von Daten in BSON und die Dekodierung aus BSON kann in den meisten Sprachen aufgrund der Verwendung von C-Datentypen sehr schnell durchgeführt werden. Seit seiner ursprünglichen Formulierung wurde BSON erweitert, um einige optionale nicht-JSON-native Datentypen wie Datumsangaben und binäre Daten hinzuzufügen, ohne die MongoDB einige wertvolle Unterstützungen fehlen würden. BSON ist so konzipiert, dass es leicht durchlaufen werden kann. Dies ist eine wichtige Eigenschaft in seiner Rolle als primäre Datendarstellung für MongoDB. MongoDB speichert Daten im BSON-Format sowohl intern als auch über das Netzwerk, aber das bedeutet nicht, dass MongoDB nicht auch als JSON-Datenbank betrachtet werden kann. Alles, was in JSON dargestellt werden kann, kann nativ in MongoDB gespeichert und genauso einfach in JSON abgerufen werden.

<http://bsonspec.org/>

<https://www.mongodb.com/json-and-bson>

### ECLIPSE - Foundation

Gemeinschaft für Open Innovation und Kollaboration

Projekte: Eclipse IDE, Jetty Webserver

<https://projects.eclipse.org/>

### Enctype - multipart/form-data

Der Body des Request hat das Format multipart/form-data. Mit diesem Format ist es möglich, mehrere voneinander unabhängige Teile auf einen HTTP-Server zu übertragen. Die einzelnen Teile verfügen über einen Header und einen Body. Sie halten sich an das MIME-Format (RFC 2045, 2046). Durch dieses Format ist es möglich, mehrere Komponenten gleichzeitig auf den Server zu übertragen. Tritt beim Ablegen einer Komponente ein Fehler auf, so wird die gesamte Aktion rückgängig gemacht.

### GridFS -

GridFS ist eine Spezifikation zum Speichern und Abrufen von Dateien in einer MongoDB-Datenbank, die die **BSON**-Dokumentgrößengrenze von 16 MB überschreiten.

Anstatt eine Datei in einem einzigen Dokument zu speichern, unterteilt GridFS die Datei in Teile, sogenannte Chunks [1], und speichert jeden Chunk als separates Dokument. Standardmäßig verwendet GridFS eine Chunk-Größe von 255 kB, d. h. GridFS unterteilt eine Datei in Chunks von 255 kB mit Ausnahme des letzten Chunks. Der letzte Chunk ist nur so groß wie nötig. In ähnlicher Weise haben Dateien, die nicht größer als die Chunk-Größe sind, nur einen letzten Chunk, wobei nur so viel Platz wie nötig plus einige zusätzliche Metadaten verwendet werden.

GridFS verwendet zwei Sammlungen zum Speichern von Dateien. Eine Sammlung speichert die Datei-Chunks, die andere speichert die Datei-Metadaten. Der Abschnitt GridFS-Sammlungen beschreibt jede Sammlung im Detail.

Wenn Sie GridFS nach einer Datei abfragen, setzt der Treiber die Chunks bei Bedarf neu zusammen. Sie können Bereichsabfragen für Dateien durchführen, die über GridFS gespeichert sind. Sie können auch auf Informationen aus beliebigen Abschnitten von Dateien zugreifen, z. B. um zur Mitte einer Video- oder Audiodatei zu "springen".

GridFS eignet sich nicht nur zum Speichern von Dateien, die größer als 16 MB sind, sondern auch zum Speichern beliebiger Dateien, auf die Sie zugreifen möchten, ohne die gesamte Datei in den Speicher laden zu müssen.

Quelle: <https://docs.mongodb.com/manual/core/gridfs/>

## **JSON -**

JSON ist eine JAVA-Bibliothek, die verwendet werden kann, um JAVA-Objekte in ihre JSON-Darstellung zu konvertieren. Sie kann auch verwendet werden, um einen JSON-String in ein äquivalentes JAVA-Objekt zu konvertieren.

## **JSON**

**JavaScript Object Notation**, besser bekannt als JSON, wurde in den frühen 2000er Jahren als Teil der JavaScript-Sprache definiert. JavaScript-Objekte sind einfache assoziative Container, in denen ein String-Schlüssel auf einen Wert abgebildet wird (der eine Zahl, ein String, eine Funktion oder sogar ein anderes Objekt sein kann). Dank dieser einfachen Spracheigenschaft lassen sich JavaScript-Objekte bemerkenswert einfach in Text darstellen:

## **Jetty - Servlet-Engine und Http-Server**

Eclipse Jetty bietet einen Webserver und javax.servlet-Container sowie Unterstützung für Web Sockets, OSGi, JMX, JNDI, JASPI, AJP und viele andere Integrationen. Diese Komponenten sind Open Source und für die kommerzielle Nutzung und Verbreitung verfügbar. Jetty wird in einer Vielzahl von Projekten und Produkten eingesetzt. Jetty kann in Geräte, Tools, Frameworks, Anwendungsserver und Cluster eingebettet werden.

<http://www.eclipse.org/jetty/>

## **Maven**

Maven ist ein auf JAVA basierendes Build-Management-Tool der Apache Software Foundation, mit dem insbesondere die Erstellung von JAVA-Programmen standardisiert verwaltet und durchgeführt werden kann.

Der Name Maven kommt aus dem Jiddischen und bedeutet so viel wie „Sammler des Wissens“.

Normalerweise werden die Informationen für ein Softwareprojekt, das von Maven unterstützt wird, in einer XML-Datei mit dem Dateinamen pom.xml (für **Project Object Model**) gespeichert. Diese Datei enthält alle Informationen zum Softwareprojekt und folgt einem standardisierten Format. Wird Maven ausgeführt, prüft es zunächst, ob diese Datei alle

nötigen Angaben enthält und ob alle Angaben syntaktisch gültig sind, bevor es weiterarbeitet.

In der pom.xml werden Softwareabhängigkeiten angegeben, die ein von Maven unterstütztes Softwareprojekt zu anderen Softwareprojekten hat. Diese Abhängigkeiten werden aufgelöst, indem Maven zunächst ermittelt, ob die benötigten Dateien in einem lokalen Verzeichnis, dem lokalen Maven-Repository, bereits vorhanden sind. Sind sie es, verwendet Maven z.B. beim Kompilieren die lokal vorhandene Datei von dort, also ohne sie in das Projektverzeichnis zu kopieren.

Kann die Abhängigkeit nicht lokal aufgelöst werden, versucht Maven, sich mit einem konfigurierten Maven-Repository im Intranet oder Internet zu verbinden und von dort die Dateien in das lokale Repository zu kopieren, um sie von nun an lokal verwenden zu können.

[https://de.wikipedia.org/wiki/Apache\\_Maven](https://de.wikipedia.org/wiki/Apache_Maven)

<https://maven.apache.org/what-is-maven.html>

## MONGODB

ist eine allgemein einsetzbare, dokumentbasierte, verteilte NoSQL Datenbank.

**NoSQL-Datenbanken** sind nicht tabellarisch und speichern Daten anders als relationale Tabellen. NoSQL-Datenbanken gibt es in einer Vielzahl von Typen, basierend auf ihrem Datenmodell. Sie bieten flexible Schemata und skalieren leicht mit großen Datenmengen und hohen Benutzerlasten.

Ein häufiges Missverständnis ist, dass NoSQL-Datenbanken Beziehungsdaten nicht gut speichern können. NoSQL-Datenbanken können Beziehungsdaten speichern - sie speichern sie nur anders als relationale Datenbanken. Tatsächlich finden viele im Vergleich zu SQL-Datenbanken, dass die Modellierung von Beziehungsdaten in NoSQL-Datenbanken einfacher ist als in SQL-Datenbanken, weil verwandte Daten nicht zwischen Tabellen aufgeteilt werden müssen.

SQL wird verwendet, wenn man mit relationalen Datenbanken interagiert, bei denen die Daten in Tabellen mit festen Spalten und Zeilen gespeichert werden. SQL-Datenbanken wurden in den frühen 1970er Jahren immer beliebter. Zu dieser Zeit war der Speicherplatz extrem teuer, daher normalisierten Software-Ingenieure ihre Datenbanken, um die Datenduplikation zu reduzieren.

NoSQL-Datenmodelle erlauben die Verschachtelung zusammengehöriger Daten innerhalb einer einzigen Datenstruktur. Im Laufe der Zeit haben sich vier Haupttypen von NoSQL-Datenbanken herausgebildet:

- Dokumentdatenbanken,
- Key-Value-Datenbanken,
- Wide-Column-Stores und
- Graphdatenbanken

MongoDB ist eine Dokumentendatenbank.

Dokumentdatenbanken speichern Daten in Dokumenten, ähnlich wie **JSON**-Objekte (JavaScript Object Notation). Jedes Dokument enthält Paare von Feldern und Werten. Die Werte können eine Vielzahl von Typen sein, z. B. Zeichenketten, Zahlen, Boolesche Werte, Arrays oder Objekte, und ihre Strukturen entsprechen den Objekten, mit denen Entwickler im Code arbeiten. Dokumentdatenbanken verwenden dynamische Schemata, was bedeutet, dass man Datensätze erstellen kann, ohne etwas vordefinieren zu müssen. Die Struktur eines Datensatzes kann einfach geändert werden, indem neue Felder hinzugefügt oder vorhandene gelöscht werden. Aufgrund ihrer Vielfalt an Feldwerttypen und leistungsfähigen Abfragesprachen eignen sich Dokumentdatenbanken für eine Vielzahl von Anwendungsfällen und können als Allzweckdatenbank verwendet werden. Sie können horizontal skaliert werden, um große Datenmengen unterzubringen.

### **Vergleich SQL – NoSQL (Dokumentdatenbank MongoDB) am Beispiel von Adressdaten**

Der SQL-Fall. Bei einer SQL-Datenbank beginnt das Einrichten einer Datenbank für Adressen mit dem logischen Aufbau des Formats und der Erwartung, dass die zu speichernden Datensätze relativ unverändert bleiben werden. Nach der Analyse der erwarteten Abfragemuster könnte eine SQL-Datenbank die Speicherung in zwei Tabellen optimieren, eine für grundlegende Informationen und eine, die sich auf den Kunden bezieht, wobei der Nachname der Schlüssel für beide Tabellen ist. Jede Zeile in jeder Tabelle ist ein einzelner Kunde, und jede Spalte hat die folgenden festen Attribute:

Nachname:: Vorname :: Adressfelder :: E-Mail-Adresse :: Telefonnummer  
Nachname :: Geburtsdatum :: Kontonummer :: Kundenjahre :: Präferenzen

Der NoSQL-Fall. Bei der Dokumentdatenbank (MongoDB) kann man sowohl die Basisinformationen als auch die Kundeninformationen in einem JSON-Dokument zusammenfassen. In diesem Fall wäre jedes der SQL-Spaltenattribute ein Feld und die Details eines Kundendatensatzes wären die mit jedem Feld verbundenen Datenwerte.

Zum Beispiel: Nachname: "Jones", Vorname: "Mary", Mittlerer\_Anfang: "S", usw.

Analogien zwischen MongoDB und einem traditionellen MySQL-System:

Table in MySQL wird in MongoDB zu Collection

Row wird zu Document

Column wird zu Field

Joins sind als linking und embedded Dokumente definiert.

**pom.xml (siehe Maven)**

**robo3T**

Freie MongoDB-GUI. Das Tool robo3T (früher robomongo) wurde von 3T Software Labs, den Entwicklern des MongoDB-Clients Studio 3T (früher MongoChef), übernommen.

## Routes

Webdienste im Spark JAVA Web Framework sind auf Routen und deren Handlern aufgebaut. Routen sind wesentliche Elemente in Spark. Jede Route besteht aus drei einfachen Teilen - einem Verb, einem Pfad und einem Rückruf (Callback).

- Das Verb ist eine Methode, die einer HTTP-Methode entspricht. Zu den Verb-Methoden gehören: get, post, put, delete, head, trace, connect und options.
- Der Pfad (auch Routenmuster genannt) bestimmt, welche URI(s) die Route abhören und eine Antwort bereitstellen soll.
- Der Callback ist eine Handler-Funktion, die für ein bestimmtes Verb und einen bestimmten Pfad aufgerufen wird, um eine Antwort auf die entsprechende HTTP-Anfrage zu erzeugen und zurückzugeben. Ein Callback nimmt ein Request-Objekt und ein Response-Objekt als Argumente.

Struktur für eine Route, die das get-Verb verwendet:

```
get("/Route-Pfad/", (request, response) -> {  
    // Callback-Code  
});
```

## Servlets,

Ein Servlet ist eine Klasse die HTTP-Requests auf einem JAVA-Webserver (bzw. genauer einem Servlet Container) entgegennehmen und beantworten kann. Eine Standardimplementierung des javax.servlet.Servlet Interfaces ist javax.servlet.http.HttpServlet. Servlets die in einem Servlet-Container wie Jetty oder in einem JEE Appserver HTTP-Requests entgegennehmen, erben von der Klasse javax.servlet.http.HttpServlet. Die Klasse enthält Methoden für die diversen HTTP Verben (GET, POST, PUT, DELETE), die von der Implementierung überschrieben werden können.

## Spark Framework

Spark Framework ist ein einfaches und ausdrucksstarkes JAVA-Web-Framework.

Es läuft mit einem eingebetteten **Jetty-Webserver**, einem OpenSource Projekt der Eclipse-Foundation.