

# Dokumentation Praktikum

Jason Hornsby, 4509594, jason.hornsby@mailbox.tu-dresden.de

Friederike Kitzing, 4038721, friederike.kitzing@mailbox.tu-dresden.de

## Entwicklung

### Grundidee

Erstellen eines einfachen Dateimanagers als Webclient.

#### Geplante Funktionen

##### Dateien

- Eigene Dateien anzeigen
- Neue Datei hinzufügen
- Datei löschen
- Dateiname ändern
- Dateien herunterladen
  
- Datei für andere Nutzer freigeben
- Dateifreigaben ändern
- Freigegebene Dateien ansehen

##### Nutzerverwaltung

- Nutzer registrieren
- Nutzer einloggen
- Authentisierung
- Nutzer auslesen

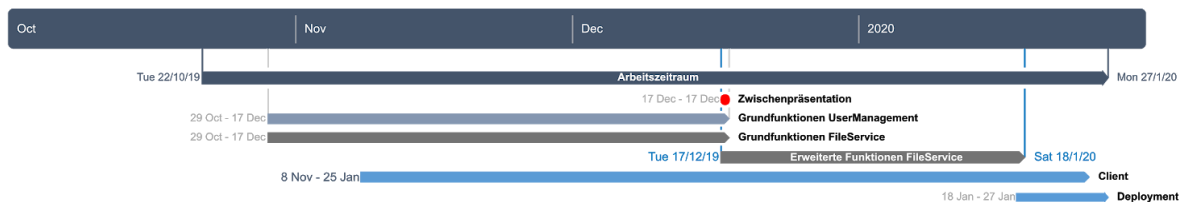
## Team

Um unabhängiges Arbeiten zu ermöglichen haben wir die Services und zusätzlichen Aufgaben wie folgend aufgeteilt:

Jason Hornsby	Friederike Kitzing
<ul style="list-style-type: none"><li>❑ User Management (Registrierung, Login, vergeben von JWT- Tokens)</li><li>❑ Client (Frontend)</li><li>❑ Deployment</li></ul>	<ul style="list-style-type: none"><li>❑ File-Service (Dateien hochladen, herunterladen, freigeben, löschen, ...)</li><li>❑ Dokumentation</li></ul>

Auf diese Weise konnte sich jeder auf die von ihm zu erstellenden Services konzentrieren. Ein Austausch erfolgte wöchentlich bis 2-wöchentlich.

## Vorgehensweise



[Größere Version der Timeline](#)

Wie zuvor erwähnt fand die Arbeit an den Services unabhängig statt. Sobald Funktionen fertiggestellt waren, wurde der Client ergänzt um diese zu unterstützen. Geplant war die grundlegenden Funktionen des User Managements (Anmelden und Login) sowie des FileService (Dateien hochladen, herunterladen und löschen) bis zur Zwischenpräsentation umzusetzen. Dieses Ziel wurde erreicht. Danach lag der Fokus auf den erweiterten Funktionen des FileService (etwa dem Freigeben der Dateien für andere Nutzer), dem Ergänzen der Sicherheitsfeatures (HTTPS, die dafür erforderlichen Zertifikate sowie dem Deployment).

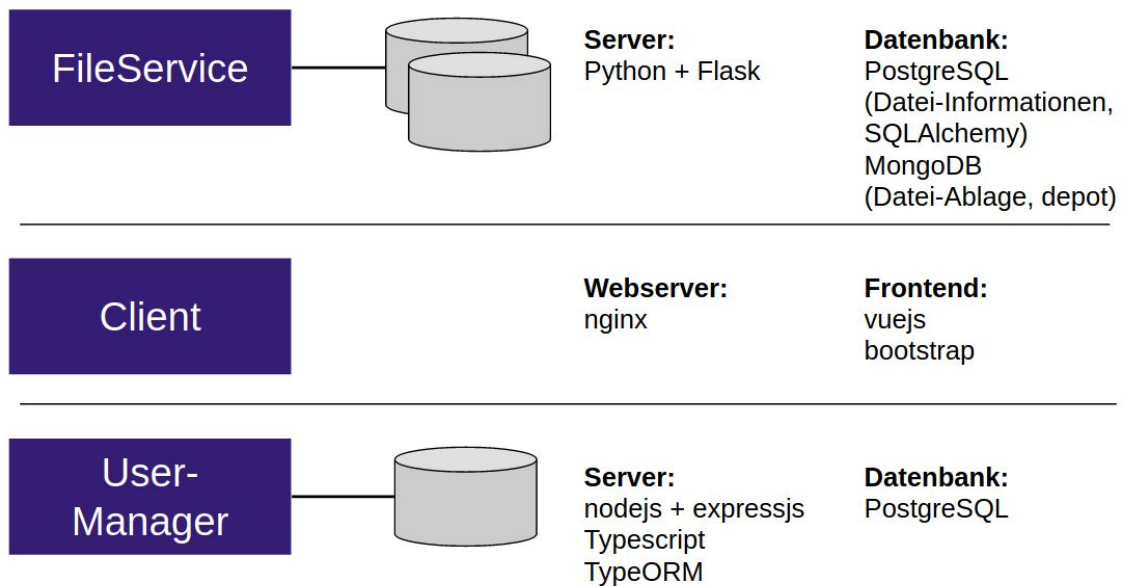
Leider nahmen diese Punkte mehr Zeit in Anspruch als erwartet, weshalb es uns nicht gelang die erweiterte Funktionalität des FileService auch im Client zu implementieren (siehe auch [Lessons Learned](#)).

## Technologieauswahl

Die Auswahl der Programmiersprachen zum Erstellen der Services erfolgte von allen Teammitgliedern unabhängig und basierend auf Vorerfahrung.

Die Auswahl für die Datenbanken richtete sich nach benötigter Funktionalität. Für die Metadaten der Dateien war es nötig die Liste der Nutzer, die Zugriffsrechte haben, zu speichern. Die einfachste Lösung dafür war das Verwenden des ARRAY-Types aus PostgreSQL. Die Ablage der Datei-Inhalte hingegen ist im FileService mit dem Python-Framework [Depot](#) umgesetzt, welches eine Implementation für MongoDB mitbringt.

## Technologien (Endstand)



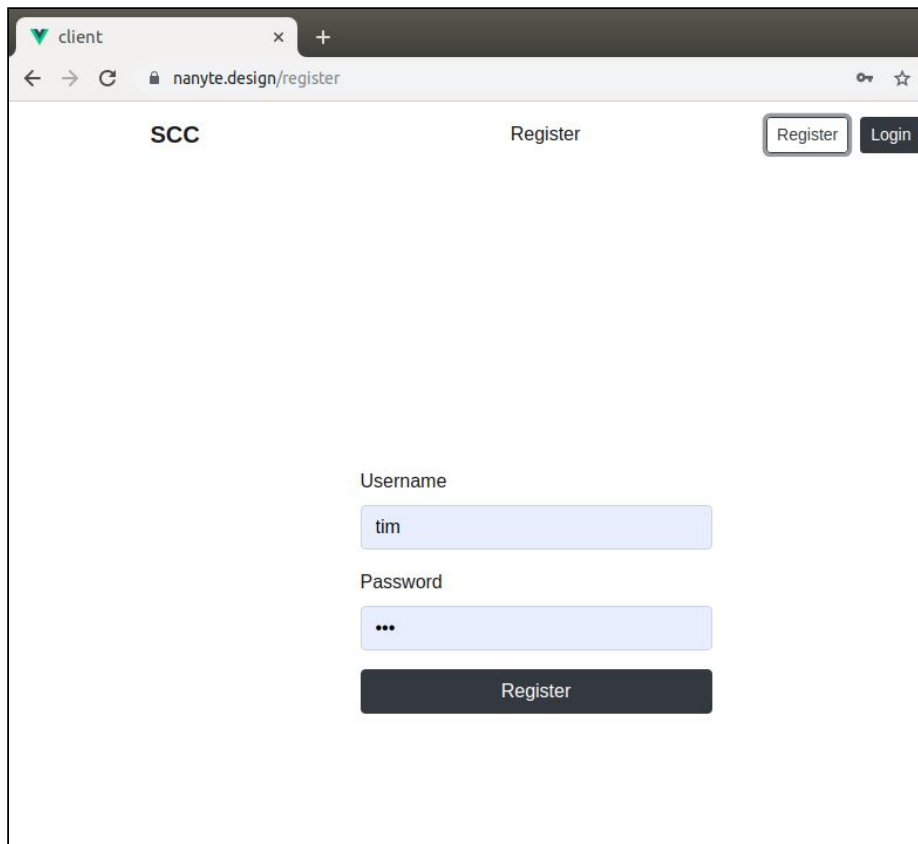
Zusätzlich zu den Haupt-Services wurden noch ein Redirector Service entwickelt der einkommende Requests an den entsprechenden Service weiterleitet. Dieser Service liegt im `services/Redirector` Ordner. Der Redirector ist ein nginx Server der neben dem Weiterleiten noch die HTTPS Verschlüsselung implementiert.

## Technische Informationen

### Bedienungsanleitung Client

Der Client verfügt über vier verschiedene Funktionsseiten: Anmeldung (Anmelden eines neuen Nutzers), Login, die Übersicht über alle hochgeladenen Dateien mit der Option neue Dateien hochzuladen sowie die Detailansicht zu jeder einzelnen Datei. Im folgenden findet sich eine kurze Beschreibung der jeweiligen Funktionalitäten sowie Abbildungen der Benutzeroberfläche.

## Anmeldung & Login



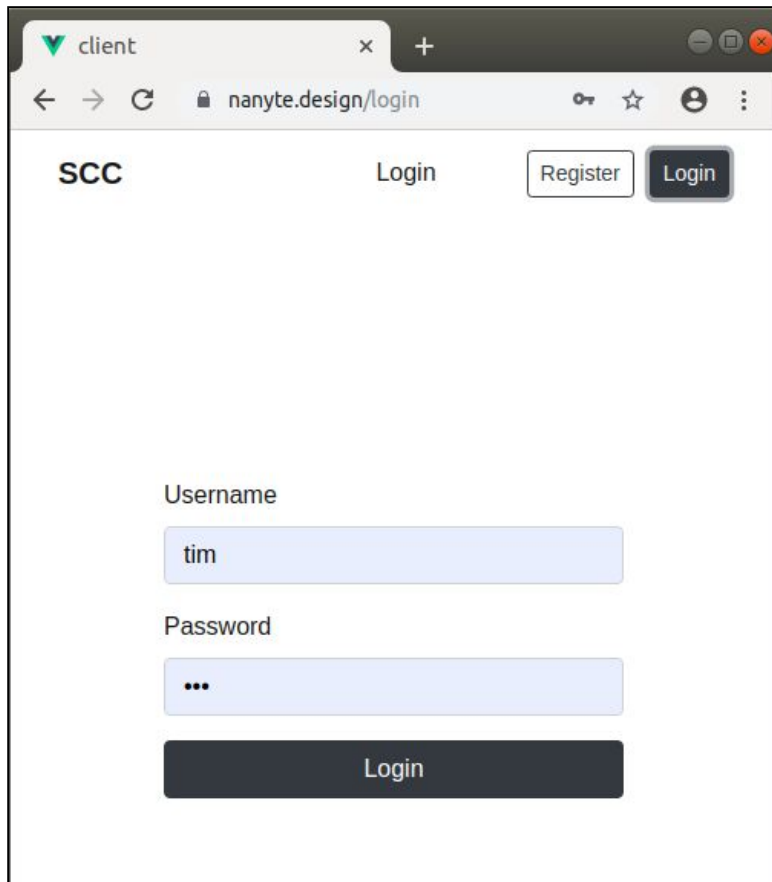
The screenshot shows a web browser window with the address bar displaying 'nanyte.design/register'. The page has a header with 'SCC' on the left, 'Register' in the center, and two buttons, 'Register' and 'Login', on the right. The main content area contains a registration form with two input fields: 'Username' with the value 'tim' and 'Password' with masked characters '\*\*\*'. Below these fields is a dark 'Register' button.

Der Client verfügt über eine einfache Anmelde-Seite in der sich neue Benutzer mit einem Nutzernamen und einem Passwort anmelden können. Danach kann der Nutzer<sup>1</sup> mit einem Klick auf den Button **Login** auf die Login-Seite begeben.

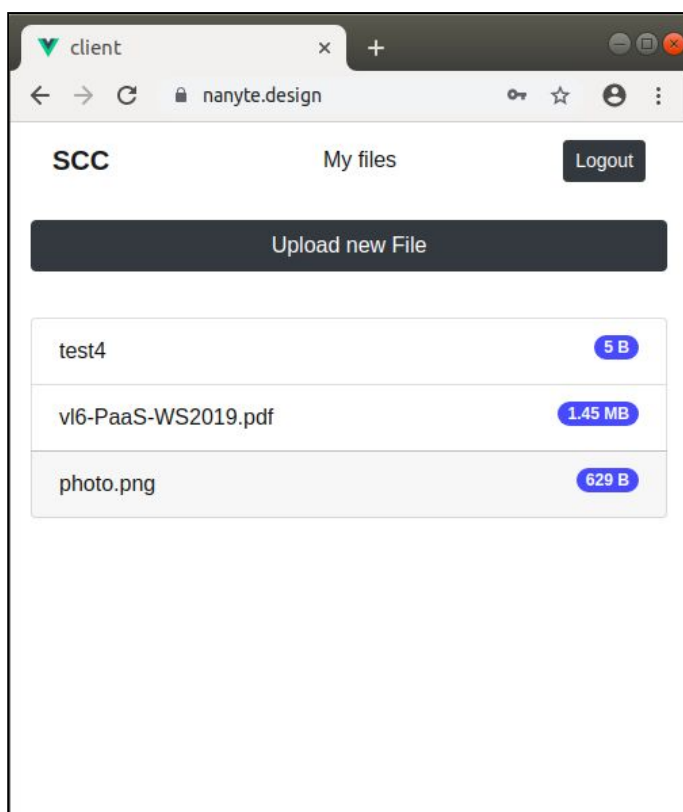
Hier kann der Nutzer sich mit den zuvor eingegebenen Zugangsdaten einloggen um die Funktionen des Services zu nutzen.

---

<sup>1</sup> Zur einfachen Lesbarkeit wird in der Beschreibung der Funktionalitäten des Clients die männliche Form verwendet. Diese Anleitung möchte hiermit ausdrücklich weibliche/... Nutzer\*innen mit einschließen ;).



## Dateiübersicht

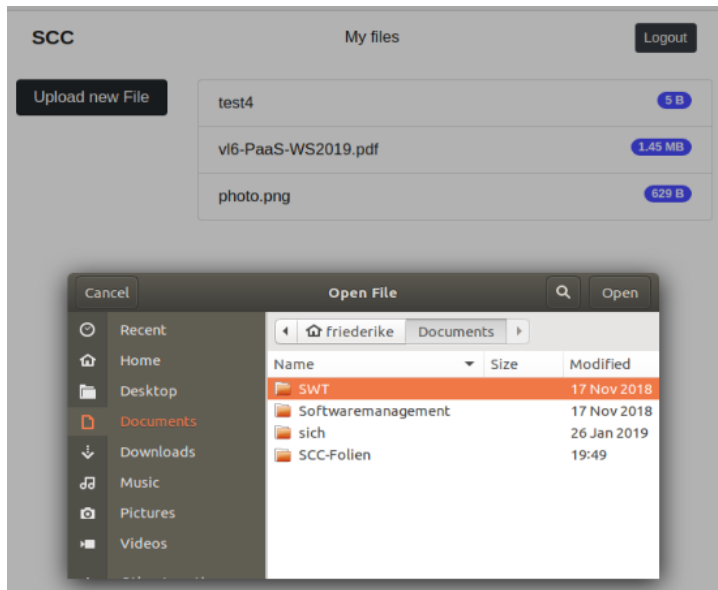


Sobald der Nutzer eingeloggt ist, sieht er die Übersicht seiner schon hochgeladenen Dateien. Zu jeder Datei wird die Größe mit angezeigt.

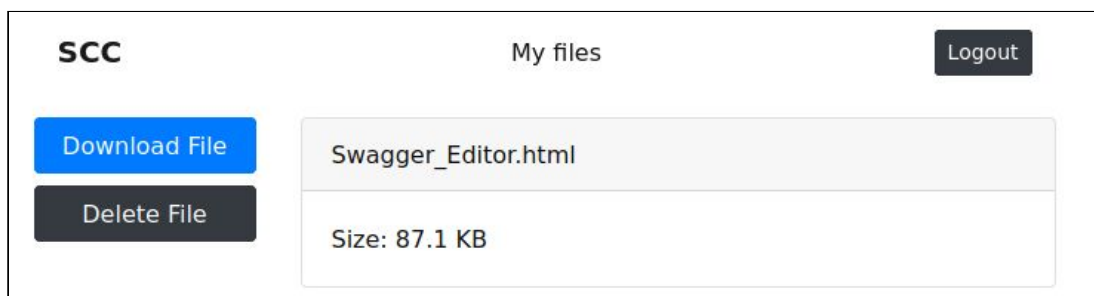
Ein Klick auf eine Datei in der Liste bereits hochgeladener Dateien führt den Nutzer auf die [Detailseite](#) der jeweiligen Datei.

Außerdem findet sich in dieser Ansicht ein Button **Upload new File** zum Hochladen neuer Dateien. Je nach Fenstergröße ist er entweder über oder links neben der Liste der Dateien zu finden.

Bei einem Klick auf den Button öffnet sich einen Dateiauswahldialog zum wählen welche Datei hochgeladen werden soll.



## Einzelne Datei



Die Detailseite einer hochgeladenen Datei enthält je einen Button zum Löschen der Datei (**Delete File**) und zum Herunterladen (**Download File**).

## Schnittstellendokumentation

Die Dokumentation der beiden Backend-Services erfolgte mittels Swagger. Sie liegt im Projekt-Repository unter *swagger\_UserManagement.yml* bzw *swagger\_FileService.yml*. Eine pdf-Version der Dokumentation (hergestellt mittels [swagger to pdf](#)) ist im Anhang dieses Dokumentes zu finden.

Die API-Dokumentation des FileService kann zudem [lokal](#) eingesehen werden.

# Deployment & Installationshinweise

## Online Deployment

Die Applikation ist über <https://www.nanyte.design> bis zu dem Abschalten der zur Verfügung gestellten VM erreichbar.

## FileService

### Lokales Deployment

Der FileService kann durch die unter `/services/FileService/docker-compose.yml` liegende docker-compose Datei als separater Service gestartet werden.

Dabei wird der Service selbst sowie zwei Datenbanken gestartet; eine Postgres-Datenbank, die die Metadaten der hochgeladenen Dateien enthält, etwa die NutzerID des Eigentümers sowie der Nutzer die die Datei herunterladen dürfen. Die andere, eine MongoDB-Datenbank, enthält den von *depot* verwalteten Inhalt der hochgeladenen Dateien.

Während die Zugriffsdaten für die Postgres-Datenbank in der Dockerfile als Umgebungsvariablen hinterlegt sind, erfolgt das Setzen des Nutzers für die MongoDB mittel eines Scriptes (`/services/FileService/init-mongo.js`).

Für das einzelne Testen des FileServices können mit [jwt](#) die notwendigen Authentication-Tokens erzeugt werden. Diese müssen als Payload das Attribute `'id'` mit der ID des Nutzers enthalten und mit dem Schlüssel `'secret'` signiert sein.

### API Docs

Die Dokumentation dieses Backends kann (im lokalen Setup) unter <http://localhost:5000/apidocs/> eingesehen werden.

## UserManagement

Um den UserManagement Service alleine zu starten kann man die Dockerfile im Ordner `services/UserManager` mit docker ausführen.

Für die Entwicklung (start ohne Container) braucht man die node und npm Command Line Utilities. Danach können die Abhängigkeiten mit `"npm install"` installiert werden. Um einen Entwicklungsserver dann zu starten muss nur noch `"npm run dev"` ausgeführt werden.

## Client

Um den Client Service alleine zu starten kann man die Dockerfile im Ordner `services/Client` mit docker ausführen. Dies startet einen nginx Container der den Client zurückliefert.

Für die Entwicklung (start ohne Container) braucht man die node und npm Command Line Utilities. Danach können die Abhängigkeiten mit `"npm install"` installiert werden. Um einen Entwicklungsserver dann zu starten muss nur noch `"npm run serve"` ausgeführt werden.

## Gesamtanwendung

Um die gesamte Anwendung zu starten kann man die docker-compose.yml Datei im Root Verzeichnis des Projektes starten. Hiermit werden alle Services und benötigten Datenbanken gebaut, gepullt und gestartet. Falls die Application in einem Kubernetes Cluster deployed werden soll kann dies mit dem Garden tools gemacht werden. Diese Configurationen sind aber noch nicht komplett vollständig und müsste an das Cluster bzw. den Cloud Anbieter angepasst werden. Für weitere Informationen siehe <https://garden.io>.

## Sicherheit

Um eine SSL Verschlüsselung zu erlauben müssen beim deployment valide fullchain.pem und privchain.pem Dateien in den services/Redirector Ordner gelegt werden. Diese Keys müssen von einem Certificate Authority für die entsprechenden domains aktiviert sein. Zusätzlich müssen in der nginx.conf die Server Attribute für die neue Domain angepasst werden.

## Lessons Learned

### Dokumentation

Die Schnittstellendokumentation erfolgte zum Schluss größtenteils per Hand. Dies erforderte am Ende noch einmal einiges an Zeit. Die Wahl und Verwendung von geeigneten Frameworks die eine automatische Erstellung ermöglichen, schon in der Planungsphase, hätte damit eine merkliche Arbeitseinsparung bedeutet.

### Zeitplanung

Die grobe Zeitplanung ("Grundfunktionen in 2019; Erweiterungen, Sicherheit und Deployment dann im Januar") die wir aufgrund der übersichtlichen Anzahl von Aufgaben und der kleinen Gruppengröße für ausreichend hielten, hat sich doch als ungenügend herausgestellt. Eine konkretere Planung, etwa durch Tickets und feste Daten, sowie eine realistischere Zeiteinschätzung, die auch das steigende Arbeitsvolumen mit der beginnenden Prüfungszeit mit einkalkuliert, wären nötig gewesen.

Insgesamt haben wir zu wenig Zeit eingeplant die erweiterten Funktionen der Services im Client einzubauen, da ab Mitte Januar der Fokus schon auf der Migration der Datenbanken und der Vorbereitung des Deployments liegen musste.

## Feedback Praktikum

### Friederike

Es war sehr gut, einen kleinen praktischen Einblick in die Entwicklung von Webservices zu erhalten. Da ich mich mit dem Thema zuvor noch nicht beschäftigt habe, habe ich viel



Neues gelernt. Danach habe ich jetzt auch das Gefühl das Prinzip von REST-Services verstanden zu haben, was mir zuvor eher noch nebulös war.

Als Verbesserungsvorschlag hätte ich nur die Idee, schon etwas eher auf die Möglichkeit von JWT-Tokens zur Authentifizierung hinzuweisen (etwa in einer weiteren Übung zum Thema Sicherheitsfeatures unserer Services?), da dieser Aspekt in den Vorlesungen doch erst recht spät Erwähnung findet.

## Jason

Das Praktikum was sehr interessant und gab einen guten Einblick in das Entwickeln von Service basierten Applikationen. Vorallem die Arbeit mit docker Container, docker-compose und kubernetes war für mich neu und ich hab viel neues gelernt. Als Verbesserungsvorschlag würde ich meinen das viel Zeit im Praktikum mit der Entwicklung von eher irrelevanten Teilen verbracht wurde. Zum Beispiel wurde viel Zeit damit verbracht Features zu implementieren die nicht direkt mit der Service Architektur in Verbindung standen. Ich hätte gerne mehr Zeit mit der Orchistrierung und Planung der Services verbracht. Vielleicht könnte man bestimmte Services in verschiedenen Formaten bereits zu Verfügung stellen damit mehr Zeit mit der Kommunikation zwischen Services und dem Deployment verbracht werden kann.

## Anhang 1: Quellen und Ressourcen

- Erstellen der PDF-Version der Schnittstellendokumentation: **Swagger to pdf** (<https://www.swdoc.org/>, abgerufen: )
- Python-Framework zum Verwalten von Dateien: **depot** (<https://depot.readthedocs.io/en/latest/index.html>, abgerufen: 23.01.2020)
- Einfaches Verwalten eines Cluster: **garden**
- Frontend Framework: **VueJs** (<https://vuejs.org/>, abgerufen: 23.01.2020)
- Backend Framework: **ExpressJS** (<https://expressjs.com/>, abgerufen: 23.01.2020)
- Liste aller verwendeten libraries im Client: siehe services/Client/package.json -> [dependencies]
- Liste aller verwendeten libraries im UserManager: siehe services/UserManager/package.json -> [dependencies]

## Anhang 2: Timeline



Die Zeitplanung kann auch als Bilddatei ("Plan.png") im Ordner "Dokumentation" gefunden werden.

## Anhang 3: Schnittstellenbeschreibung im pdf-Format

Aufgrund der Schwierigkeit, pdf-Dokumente in ein Google-Doc einzufügen, kann die generierte Schnittstellenbeschreibung unter

- *“Dokumentation/api-documentation-FileService.pdf”* (FileService)
- sowie
- *“Dokumentation/api-documentation-UserManagement.pdf”*  
(User-Management)

eingesehen werden.

Das Generierungs-Tool fügt bei Rückgabe-Parametern unnötigerweise entweder “optional” oder “required” als Eigenschaft an. Diese Zusätze können ignoriert werden: Die Datenobjekte, die von den Service zurückgeschickt werden enthalten in jedem Fall alle in der API-Dokumentation aufgeführten Werte.