

...

Herval Bernice Nganya Nana<sup>1</sup>

<sup>1</sup>*Fachbereich Informatik und Medien, Technische Hochschule Brandenburg, Deutschland  
nganyana@th-brandenburg.de*

Keywords: ...

Abriss: ...

Abstract: ...

## 1 PROTOTYP

In diesem Kapitel geht es um die Beschreibung des Weges zur Erzeugung des ersten Produkts. Dieses wurde mittels der Werkzeuge Swagger (swa, ) (Swagger UI, Swagger Editor und Swagger Codegen), Amazon Web Services und Hibernate erarbeitet. Für eine Modellgetriebene Entwicklung wurde während der Durchführung dieses Projekts Bottom-Up entwickelt, sondern Top-Down. Also wurde direkt von angefertigten Modellen zum generierten Produkt entwickelt.

### 1.1 Funktionalität

Da das Ziel des ganzen, ursprünglichen Projekts ein Anwendung zur Datenverwaltung ist. Das in dieser Dokumentation beschriebene Rest-Service, der dafür verantwortlich ist Schnittstellen zu diesem Zweck bereitzustellen, ermöglicht:

1. einen Benutzer zu erstellen
2. sich als Benutzer ein- und auszuloggen
3. Dateien hochzuladen, herunterzuladen, zu löschen, umzubenennen und aufzulisten
4. Ordner zu erstellen, zu löschen, umzubenennen und zu visualisieren.

Insgesamt sind von dem Rest-Service 11 Schnittstellen zur Verfügung gestellt.

### 1.2 Schnittstellenerzeugung

Der Prozess zur Erzeugung der Schnittstellen ist in Folgende gelaufen:

1. zuerst wurden mittels des Werkzeugs Swagger Editor die Modelle, Schnittstellen sowie einpaar Meta-Daten des Projekts unter Anderen der Titel, die Beschreibung, die Version beschrieben. All das wird entweder unter einer JSON- oder YAML-Datei gespeichert. In diesem projekt wurde die JSON-Datei bevorzugt.
2. Anschließend wurde aus dieser Beschreibung eine graphische Beschreibung der Schnittstellen als Webseite mittels Swagger UI erzeugt. Die enthält einerseits die Schnittstellenbeschreibung andererseits die Modellbeschreibung.
3. Dann wurde ebenfalls aus der JSON-Beschreibung der Quellcode anhand Swagger Codengen generiert. Swagger Codegen ermöglicht, den Quellcode in vielen Programmiersprachen (Backend sowie Frontend) zu erzeugen. Für dieses Projekt wurde ein Spring-Projekt gewählt, da das angestrebte Produkt ein Rest-Service sein soll.
4. Danach wurden in dem generierten Spring-Projekt Abhängigkeiten hinzugefügt. Diese sind unter Anderen AWS S3, MySql und Spring-Data.

5. Der nächste Schritt war dann die Verbindung zu der MySQL-Datenbank und die Modelle gemäß der verschiedenen Attribute, die in der Datenbank zu speichern sind, zu annotieren. Für dieses ist Hibernate zuständig.
6. Abschließend wurde jede von Swagger automatisch generierte Funktion ausgefüllt. Bei der Generierung der Schnittstellen ist von Swagger eine leere Funktion pro Schnittstelle generiert. Es bleibt nur noch diese Funktionen auszufüllen.

### 1.3 Erweiterung der Schnittstellen (mit Datenbankerstellung)

Der Rest-Service könnte noch Funktionalitäten bereitstellen, die in diesem Prototyp noch nicht entwickelt worden sind.

#### 1.3.1 Encryption-Möglichkeit

In dem UserLogin-Modell ist ein Feld encryption so gedacht, dass die übertragenen Daten vor der Übertragung verschlüsselt werden. Bei True sollen die Daten chiffriert werden und bei False nicht. Über das Verschlüsselungsverfahren soll sich der Entwickler entscheiden.

### 1.4 Rebuild des Projektes nach Schnittstellenbearbeitung

#### 1.4.1 Paper lesen, eine Möglichkeit wählen und im Zusammenhang mit dem Thema beschreibung

## 2 Ergebnisse und Probleme

### 2.1 Ergebnisse

Schließlich liegt nach einer Schritt für Schritt Arbeitsweise der gezielte Rest-Service vor. Der ist in der Lage die 11 Funktionalitäten anzubieten, die vorher angekündigt waren. Die Daten sind dank einer Datenbank und einer Speicherung der auf AWS S3 tatsächlich persistent. Der Rest-Service kann ebenfalls entsprechende Antworten zurückgeben:

1. 400 Bad Request
2. 403 Forbidden
3. 404 Not Found

### 2.2 Probleme

Während der Entwicklung ist das Team auf Schwierigkeiten gestoßen. Diese waren einerseits aufgrund der Verwendung eines Modellgetriebene-Software-Entwicklungswerkzeugs ausgelöst andererseits aufgrund der Datensicherheit bei der Nutzung vom AWS S3.

#### 2.2.1 Unübersichtlicher, generierter Code

Der von Swagger generierte Code lässt ist an vielen Stellen kaum übersichtlich. Der Code in den Kontrollern und Interfaces ist schwer zu verstehen (Abbildung 1).

```

115 @ApiOperation(value = "Edit a folder", notes = "This can only be done by the logged in user.", response = Folder.class, tags = {"folders"})
116 @ApiResponse(code = 200, message = "Successful operation", response = Folder.class),
117 @ApiResponse(code = 401, message = "Unauthorized", response = Folder.class),
118 @ApiResponse(code = 404, message = "Not Found", response = Folder.class) })
119 @RequestMapping(value = "/{token}/{folderId}",
120 produces = { "application/json" },
121 method = RequestMethod.PUT)
122 ResponseEntity<Folder> editFolder(
123 @ApiParam(value = "Id of the folder to edit", required=true ) @PathVariable("folderId") String folderId
124 )
125
126
127
128 @ApiParam(value = "token of the current user", required=true ) @PathVariable("token") String token
129
130
131
132
133 @ApiParam(value = "Edit a folder in data storage", required=true ) @RequestBody FolderCreate folder
134 );
135
136
137
138 @ApiOperation(value = "Get a file", notes = "This can only be done by the logged in user.", response = File.class, tags = {"files"})
139 @ApiResponse(code = 200, message = "Successful operation", response = File.class),
140 @ApiResponse(code = 401, message = "Unauthorized", response = File.class),
141 @ApiResponse(code = 404, message = "Not Found", response = File.class) })
142 @RequestMapping(value = "/{token}/{folderId}/files",
143 produces = { "application/json" },
144 method = RequestMethod.GET)
145 ResponseEntity<File> getFile(
146 @ApiParam(value = "Token of the current user", required=true ) @PathVariable("token") String token
147 )
148
149
150
151 @ApiParam(value = "Id of the parent folder", required=true ) @PathVariable("folderId") String folderId
152
153
154 );
155

```

Figure 1: Quellcode der Schnittstellen getFile und editFile

#### 2.2.2 Projektstruktur (Packages, ...)

Nach Entwicklern oder Organisationen kann die Paketstruktur variieren. Daher ist die von Swagger angebotene Paket Struktur immer auf dem ersten Blick ungewohnt (Abbildung 2). In dieser Abbildung ist das Paket io.swagger.service nicht von Swagger sondern von dem Entwicklungsteam generiert.

#### 2.2.3 Sicherheit

AWS bietet den S3-Dienst an, der ermöglicht Dateien auf dem Cloud zu speichern. Dies erfolgt per Quellcode mittels Access- und Secret-Keys. Damit das Team Transaktionen mit S3 während der Entwicklung durchführen kann, soll es diese Schlüssel besitzen. Das Problem ist an der Stelle die Verteilung dieser Schlüssel. Diese kann zu bösen Zwecken verwendet werden und schließlich hohe Kosten für den offiziellen Besitzer verursachen. Außerdem wären

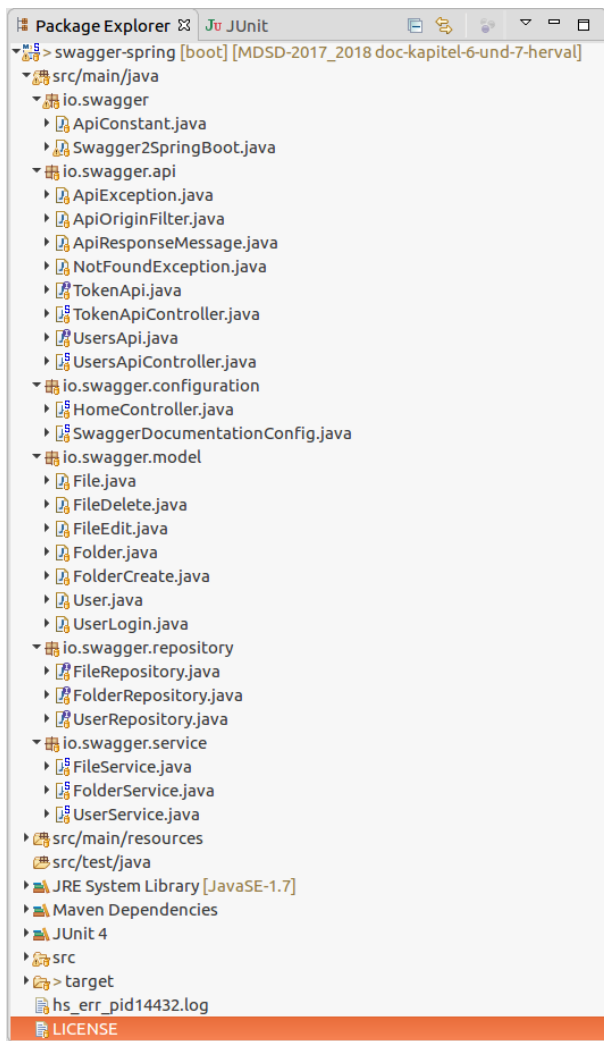


Figure 2: Von Swagger generierte Paketstruktur

sie öffentlich für Hackers durch GitHub gewesen, da GitHub zur Versionsverwaltung verwendet wurde. Aus diesen Gründen wurde ein Nexus-Repository entwickelt und geheim gehalten, damit diese Access- und Secret-Keys nicht überall stehen. Dieses Repository stellt ein Object S3Transaction zur Verfügung, das seinerseits Funktionen wie upload, rename und delete zur Verfügung stellt. Die S3Transaction-Abhängigkeit ist anhand des Folgenden aufzurufen:

```
<dependency>
<groupId>com.mdsd-2017-2018.s3-transactions</groupId>
<artifactId>s3-transactions</artifactId>
<version>1.2.2</version>
</dependency>
```

## REFERENCES

### List of Tables

### List of Figures