...

Herval Bernice Nganya Nana[1]
[1]*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 (SmartBear,) (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 enwickelt.

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 auszulogen
3. Dateien hochzuladen, herunterzuladen, zu löschen, umzubennenen und aufzulisten
4. Ordner zu erstellen, zu löschen, umzubennenen 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 bleib 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

Das Projekt wird basiert auf einer Menge von Schritten nach und nach durchgefhrt. Gleich nach der Beschreibung der Modelle und der Schnittstellen bis zum generierten Projekt wird jede Code-Zeile automatisch generiert. Was ist denn, zu tun, falls nderungen in der Beschreibung vorkommen? Der neu generierte Code soll integriert werden, ohne den bereit Selbstgeschriebene Code zu berschreiben. Zu diesem Zweck werden in dem Paper (Timo Greifenberg and Wortmann, 2015a) ein paar Methoden vorgestellt, die in solchen Fllen ntzlich sind. Im Rahmen dieser Arbeit wurde aus Zeitgrnden keine dieser Methoden implementiert, aber eine davon hat sich trotzdem durch die fnf von diesem Paper vorgestellten Kriterien zu diesem Projekt passend gezeigt. Die heit Generation Gap (Timo Greifenberg and Wortmann, 2015b). Bei diesem Verfahren werden beispielsweise fr eine Klasse NotePad ein Interface NotePad und eine Standard-Implementierung NotePadBaseImpl generiert. Diese Klasse zur Implementierung unterscheidet sich von der eigenen Implementierung, die zum Beispiel in der Klasse NotePadImpl geschrieben ist, in sofern als die dazu gehrigen Codes unterschiedlich sind und die Klasse NotePadImpl wird zustzlich bei einer neuen Erzeugung des Codes nicht berschrieben (Abbildung 1). NotePadImpl ist die Implementierung, die von beiden Codes (der generierte und der Selbstgeschriebene) verwendet werden soll.

Grnde fr die Wahl dieser Methode ist:

1. Die Struktur des endgltigen Quellcodes in diesem Projekt weicht nur von dem ursprnglichen Code um die Pakete io.swagger.service, io.swagger.repository, io.swagger.configuration. hnlich NotePadImpl in (Timo Greifenberg and Wortmann, 2015b) sind die Klassen wie FileService, FolderService und UserService die Selbstgeschriebenen Klassen, die nicht berschrieben werden sollten. Dies erfllt das Kriterium C1.

2. Die anderen Klassen in io.swagger.api, io.swagger.api, io.swagger.model und io.swagger sind hier generierte pakete. Ihre Inhalte knnen beliebig berschrieben werden. Das Kriterium C2 ist erfllt.

3. Die generierten Schnittstellen knnen im Laufe des Projekt erweitert werden, ohnen den Selbstgeschriebenen Code zu beeinflssen. Das Kriterium C3 ist erfllt.

4. Das Kriterium C4 ist ebenfalls erfllt, da der Selbstgeschriebene Code unabhngig von dem generierten Code ist. Dies ist auf die Tatsache, dass der generierte Code keinen Einfluss auf den Selbstgeschriebenen hat.

5. In diesem Projekt wurde der Code in Spring (Java) generiert. Nach einer neuen Erzeugung ist der generierte Code immernoch auf die gleiche Programmiersprache. Dies fordert deshalb keine zustzliche Programmiersprache. So ist das Kriterium C5 erfllt.

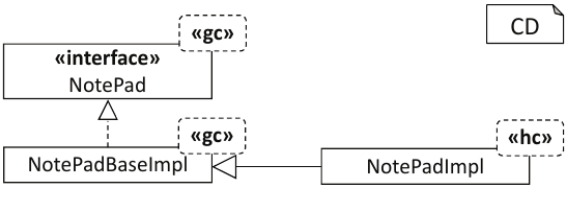

Figure 1: Generation Gap Muster fr das NotePad-Beispiel
gc: generated code, hc: hand code

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 2).

```
    @ApiOperation(value = "Edit a folder", notes = "This can only be done by the logged in user.", response = Folder.class, tags
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "Successful operation", response = Folder.class),
        @ApiResponse(code = 401, message = "Unauthorized", response = Folder.class),
        @ApiResponse(code = 404, message = "Not found", response = Folder.class) })
    @RequestMapping(value = "/{token}/{folderId}",
        produces = { "application/json" },
        method = RequestMethod.PUT)
    ResponseEntity<Folder> editFolder(
@ApiParam(value = "Id of the folder to edit",required=true ) @PathVariable("folderId") String folderId

,
@ApiParam(value = "token of the current user",required=true ) @PathVariable("token") String token


,

@ApiParam(value = "Edit a folder in data storage" ,required=true ) @RequestBody FolderCreate folder

);

    @ApiOperation(value = "Get a file", notes = "This can only be done by the logged in user.", response = File.class, tags={ "F
    @ApiResponses(value = {
        @ApiResponse(code = 200, message = "Successful operation", response = File.class),
        @ApiResponse(code = 401, message = "Unauthorized", response = File.class),
        @ApiResponse(code = 404, message = "Not found", response = File.class) })
    @RequestMapping(value = "/{token}/{folderId}/files",
        produces = { "application/json" },
        method = RequestMethod.GET)
    ResponseEntity<File> getFile(
@ApiParam(value = "Token of the current user",required=true ) @PathVariable("token") String token


,
@ApiParam(value = "Id of the parent folder",required=true ) @PathVariable("folderId") String folderId


);
```

Figure 2: Quellcode der Schnittstellen getFile und editFile

2.2.2 Projektstruktur

Nach Entwicklern oder Organisationen kann die Paketestruktur variieren. Daher ist die von Swagger angebotene Paket Struktur immer auf dem ersten Blick ungewöhnt (Abbildung 3). In dieser Abbildung ist das Paket io.swagger.service nicht von Swagger sondern von dem Entwicklungsteam generiert.

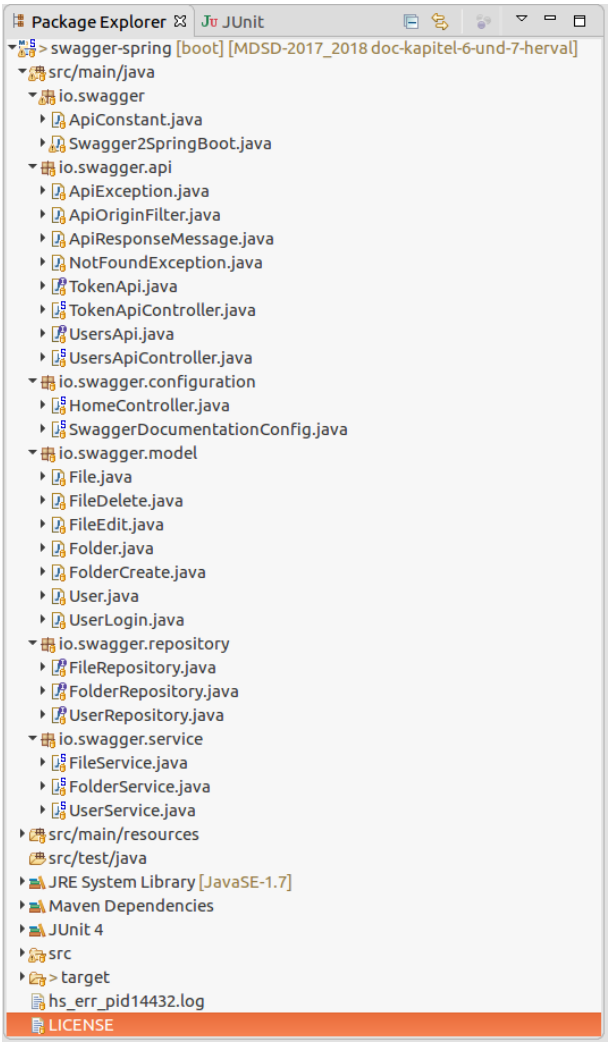

Figure 3: Von Swagger generierte Paketstruktur

2.2.3 Sicherheit

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

SmartBear. Swagger - world's most popular api framework.

Timo Greifenberg, Katrin Hlldobler, C. K. M. L. P. M. S. N. K. M. A. N. P. D. P. D. R. A. R. B. R. M. S. and Wortmann, A. (2015a). *A Comparison of Mechanisms for Integrating Handwritten and Generated Code for Object-Oriented Programming Languages*. Software Engineering, RWTH Aachen University, Institute for Building Services and Energy Design, TU Braunschweig.

Timo Greifenberg, Katrin Hlldobler, C. K. M. L. P. M. S. N. K. M. A. N. P. D. P. D. R. A. R. B. R. M. S. and Wortmann, A. (2015b). *A Comparison of Mechanisms for Integrating Handwritten and Generated Code for Object-Oriented Programming Languages, Page 76, Chapter 3.1*. Software Engineering, RWTH Aachen University, Institute for Building Services and Energy Design, TU Braunschweig.

List of Tables

List of Figures