...

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 eines Prototyps. 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 nicht 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, ermöglicht das in dieser Dokumentation beschriebene Rest-Service, der dafür verantwortlich ist Schnittstellen zu diesem Zweck bereitzustellen, folgende Funktionalitäten:

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 wurde in folgenden Schritten durchgeführt:

1. Zuerst wurden mittels des Werkzeugs Swagger Editor die Modelle, Schnittstellen sowie ein paar Meta-Daten des Projekts unter anderem 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 ausgewählt.
2. Anschließend wurde aus dieser Beschreibung eine graphische Dokumentation 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 des Swagger-Codengen-Werkzeugs generiert. Swagger Codegen ermöglicht, den Quellcode in vielen Programmiersprachen (Backend sowie Frontend) zu erzeugen. Für dieses Projekt wurde ein Spring-Projekt gewählt, da den angestrebten Prototypen ein Java-basierter Rest-Service sein

soll.

4. Danach wurden in dem generierten Spring-Projekt Maven-Abhängigkeiten hinzugefügt. Diese sind unter anderem 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 diesn Schritt wurde Hibernate benutzt.
6. Abschließend wurde jede von Swagger automatisch generierte Funktion ausgefüllt. Bei der Generierung der Schnittstellen wird von Swagger eine leere Funktion pro Schnittstelle erzeugt. Es bleibt nur noch diese Funktionen auszufüllen.

1.3 Erweiterung der Schnittstellen

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

1.3.1 Encryption-Möglichkeit

Das UserLogin-Modell kann um das Feld encryption erweitert werden. Dieses Feld ist 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 sollen sich die Entwickler entscheiden.

1.4 Rebuild des Projektes nach Schnittstellenbearbeitung

Das Projekt basiert auf einer Menge von Schritten, die nach und nach durchgeführt wurden. 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 Änderungen in der Beschreibung vorkommen? Der neu generierte Code soll integriert werden, ohne den bereits selbstgeschriebenen Code zu überschreiben. Zu diesem Zweck werden in dem Paper (Timo Greifenberg and Wortmann, 2015a) ein paar Methoden vorgestellt, die in solchen Fällen nützlich sind. Im Rahmen dieser Arbeit wurde aus Zeitgründen keine dieser Methoden implementiert, aber eine davon hat sich trotzdem durch die fünf von diesem Paper vorgestellten Kriterien (C1 bis C5) (Timo Greifenberg and Wortmann, 2015b) zu diesem Projekt passend gezeigt. Die heißt Generation Gap (Timo Greifenberg and Wortmann, 2015b). Bei diesem Verfahren werden beispielsweise für 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 dazugehörigen Codes unterschiedlich sind und die Klasse NotePadImpl wird zusätzlich bei einer neuen Erzeugung des Codes nicht überschrieben (Abbildung 1). NotePadImpl ist die Implementierung, die von beiden Codes (der generierte und der Selbstgeschriebene) verwendet werden soll.

Gründe für die Wahl dieser Methode sind:

1. Die Struktur des endgültigen Quellcodes in diesem Projekt weicht nur von dem ursprünglichen Code um die Pakete io.swagger.service, io.swagger.repository, io.swagger.configuration. Ähnlich NotePadImpl in (Timo Greifenberg and Wortmann, 2015b) sind die Klassen wie FileService, FolderService und UserService die selbstgeschriebenen Klassen, die nicht überschrieben werden sollten. Dies erfüllt 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 können beliebig überschrieben werden. Das Kriterium C2 ist erfüllt.
3. Die generierten Schnittstellen können im Laufe des Projekt erweitert werden, ohnen den selbstgeschriebenen Code zu beeinflüssen. Das Kriterium C3 ist erfüllt.
4. Das Kriterium C4 ist ebenfalls erfüllt, da der Selbstgeschriebene Code unabhängig 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 zusätzliche Programmiersprache. So ist das Kriterium C5 erfüllt.

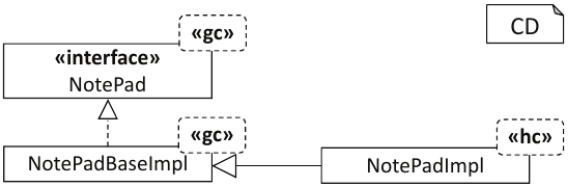

Figure 1: Generation Gap Muster für das NotePad-Beispiel
gc: generated code, hc: hand code

2 Ergebnisse und Probleme

2.1 Ergebnisse

Schließlich liegt nach der Arbeitsweise der gezielte Rest-Service vor. Dieser ist in der Lage die gewünschten Funktionalitäten anzubieten, die vorher abgesprochen waren. Die Daten sind durch die Datenbank und Speicherung der Daten auf einer AWS-S3-Instanz tatsächlich persistent. Der Rest-Service kann ebenfalls entsprechende Antworten zurückgeben:

1. 200 OK
2. 400 Bad Request
3. 403 Forbidden
4. 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 ist an vielen Stellen unübersichtlich. Der Code in den Controllern 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

Jeder Entwickler und jede Organisation knnen ihre eigene Paketstruktur besitzen. Daher ist die von Swagger angebotene Paketstruktur immer auf dem ersten Blick ungewöhnlich (Abbildung 3). In dieser Abbildung ist das Paket io.swagger.service nicht von Swagger sondern von dem Entwicklungsteam generiert.

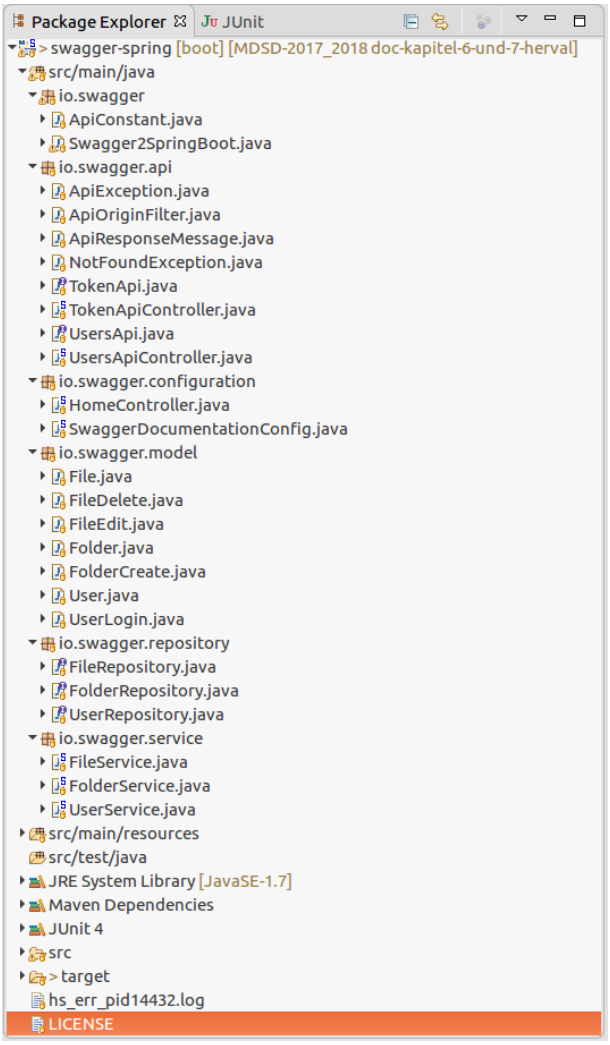

Figure 3: Von Swagger generierte und vom Entwicklungsteam veränderte Paketstruktur

2.2.3 Sicherheit

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

sie öffentlich für Hacker 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 verteilt werden. Dieses Repository stellt ein Object S3Transaction zur Verfügung, das seinerseits Funktionen wie upload, rename und delete zur Verfügung stellt. Die S3Transaction-Abhängigkeit ist anhand der folgenden Dependency aufzurufen:

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

2.2.4 Abstürtzen der EC2-Instanz

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