|  |
| --- |
| Model driven software development mit dem tool swagger |
|  |
| Herval Bernice Nganya Nana1, Oussema Mzoughi2, Aliridha Haouari3, Christian Lange4 |
| 1,2,3,4Technische Hochschule Brandenburg Fachbereich Informatik und Medien Magdeburger Str. 50, 14770 Brandenburg an der Havel  1 nganyanana@th-brandenburg.de  2 oussema.mzoughi@th-brandenburg.de  3aliridha.haouari@gmail.com  4langchri@th-brandenburg.de |

Keywords:

Abstract: Ein Softwareprojekt lebt und wird immer weiter entwickelt. Daher ist es wichtig, dass die Software vom Architekten gut geplant und ebenso auch vom Entwickler gut umgesetzt wird. Problematisch ist, dass die vom Architekten erzeugten Diagramme vom Softwareentwickler fehlinterpretiert werden können. Dadurch entsteht eine Archtikturerosion, die das Erweitern von Software erschwert oder sogar unmöglich macht. Um diesen Punkt entgegenzutreten, gibt Model Driven Software Development. Die Software wird vom Architekten beschrieben und ein Build-Tool wird genutzt um aus dieser Beschreibung eine entsprechende Software zu generieren. Ein Beispiel dafür ist die Beschreibungssprache für REST-Schnittstellen Swagger. Mit der Nutzung dieser Sprache und den dafür vorgesehenen Tools soll eine Serveranwendung entwickelt werden. Dabei wird überprüft, wie sich die neue Herangehensweise an Softwareentwicklung im Team ausprägt. Es konnte erfolgreich eine Serveranwendung geplant und entwickelt werden.

.

# Einleitung4

* 1. **Motivation**

Viele Softwareprojekte stoßen bei der Erweiterung der eigenen Software auf große Probleme. Die Software wird vom Architekten geplant und von den Softwareentwicklern umgesetzt. Jedoch können die Anforderungen von dem Architekten missverstanden werden. Somit driften die geplante und die implementierte Software voneinander ab. Diese Archtikturerosion kann bei der Erweiterung der Software zu großen Problemen führen[1]. Diese dann auszubessern, kann sehr zeitaufwändig und teuer werden.

Zur Eindämmung solcher Probleme gibt es ein Verfahren, das die Software aufgrund von eindeutig beschriebenen Modellen generiert. Dieses Verfahren nennt sich Model Driven Software Development (MDSD). Die gesamte Software wird via Domain-Specific Language (DSL) beschrieben.

Die Planung der Software wird via Top-down vorangetrieben. Dadurch wird vom abstrakten Datenobjekt zur fertigen Funktionalität modelliert.

Es gibt bereits einige nützliche Werkzeuge, die solche DSL´s zur Verfügung stellen. Mithilfe dieser Werkzeuge kann eine Software plattformunabhängig beschrieben und später auf einer bestimmten Plattform gebuildet werden.

Da die Software nun einen eindeutigen Rahmen besitzt, kann die Gefahr einer Architekturerosion verringert werden. Die Grundlage der Software ist ihre Beschreibung. Dadurch haben alle Parteien (z. B. Architekten und Entwickler) eine identische und stetig aktuelle Basis.

Ein Beispiel dafür ist die Beschreibungssprache für REST-Schnittstellen Swagger[2]. Mit deren Hilfe können Serveranwendungen beschrieben und später auf eine beliebige Zielplattform gebuildet werden.

In dieser Ausarbeitung wird die Beschreibungssprache Swagger getestet und eine Serveranwendung mithilfe dieser Sprache beschrieben und implementiert.

* 1. **Ziel**

Diese Ausarbeitung beschäftigt sich mit der Entwicklung einer Software, die via MDSD entwickelt wird. Es soll überprüft werden, wie sich eine solche Mechanik in Projektarbeiten auswirken kann. Weiterhin soll überprüft werden, welche Probleme bei der Umsetzung des MDSD zum Vorschein kommen können.

* 1. **Aufgabenstellung**

Die zentrale Aufgabe ist es eine Serveranwendung zu entwickeln, die mit der DSL von Swagger beschrieben wurde. Um die Anwendung zu builden, muss ein Überblick über die Möglichkeiten mit Swagger erzeugt werden. Der nächste Schritt ist die Planung der Software. Dieses Projekt ist ein Kooperationsprojekt. Es soll eine Serveranwendung mit REST-Schnittstellen entwickelt werden, die wiederum vom anderen Team aufgerufen und für deren Projekt genutzt werden kann. Daher müssen auch die entsprechenden Schnittstellen gemeinsam geplant und umgesetzt werden. Die Daten sollen persistent gespeichert werden können. Dies fordert eine Planung und Integration einer Datenbank. Abschließend soll ein Fazit über die Arbeit mit MDSD gezogen werden.

* 1. **Abgrenzung**

Diese Ausarbeitung befasst sich leidglich mit Erstellen eines Softwareprojekts mit der Beschreibungssprache Swagger Die erstellten REST-Schnittstellen arbeiten mit keiner Verschlüsselung der Daten. Der Punkt Softwaresicherheit wird gänzlich ausgelassen. Weiterhin wird nicht geprüft, was beim erneuten builden der Software mit dem handgeschriebenen Code passiert.

* 1. **Ergebnis**

Abschließend soll ein Einblick über das Entwickeln einer Software mit MDSD gewonnen werden. Es soll sich herauskristallisieren, wo die Stärken und Schwächen dieser Entwicklung liegen.

# Model-driven-development

* 1. **Allgemeines, was ist das? Usw.**

Text

* 1. **CIM / PIM**

Text

# swagger (Rest service)

* 1. **Was ist Swagger**

Text

* 1. **Wie funktioniert Swagger**

Text

* 1. **Swagger mit PIM-Nutzung**

Text

* 1. **Dokumentation (Auto generated)**
     1. **Screen der Online-Dokumentation**
  2. **Buildvorgang (Von JSON-Datei zum generierten Projekt mit Swagger-2Editor)**

Text

# datenbank (Spring jpa / Hibernate)

# Konzept4

* 1. **Beschreibung der Software**

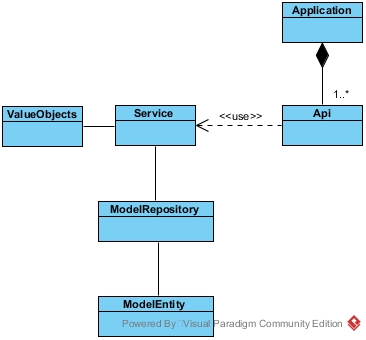
Die erstellte Software soll ein Dateimanagmentsystem für Studierende sein. Die Studenten sollen sich einloggen und verschiedene Daten hoch- bzw. herunterladen, eine eigene komplexe Ordnerstruktur erzeugen und Dateien und Ordner wieder löschen können. Die Anwendung soll über REST-Schnittstellen zugänglich gemacht werden.

* 1. **Planung der Software**

Zu Beginn soll ein Überblick verschafft werden, was für die Erfüllung dieses Projekts gebraucht wird. Da die Applikation und Schnittstellen via Swagger beschrieben werden sollen, muss die Architektur klar strukturiert werden. Dafür werden ein Metamodell, ein Klassendiagramm, ein Komponentendiagramm und für die Datenbank ein ERM erzeugt.

* 1. **Metamodell**

Um einen gesamten Überblick über den Ablauf des Programms zu gewinnen, wurde folgendes Metamodell erzeugt (siehe Abb. ?).



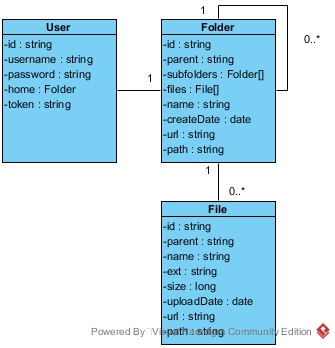
*Abb. ? Metamodell vom Softwareprojekt*

Die Applikation besteht aus einer oder mehrerer Schnittstellen (Api). Diese Schnittstellen werden vom Nutzer via REST aufgerufen. Diese Schnittstellen rufen den entsprechenden Service auf. Dieser bekommt von der Schnittstelle die Parameter als ValueObjects übergeben. Der Service arbeitet mit dem ModelRepository, um Zugriff zu einzelnen Entitäten des Modells zu bekommen. Das ModelRepository liest dafür die Daten aus einer Datenbank und erstellt die einzelnen Entitäten. Der Service kann diese Entitäten nun manipulieren, um so die gewünschte Aufgabe zu erfüllen.

Auf diese Art sollen alle Aufgaben gelöst werden können.

* 1. **Klassendiagramm**

Ein Überblick über die gesamten Datenstrukturen ist ein wesentlicher Bestandteil bei der korrekten Planung einer Software. Aus diesem Softwareprojekt erstellt sich folgendes Klassendiagramm (siehe Abb. ??).



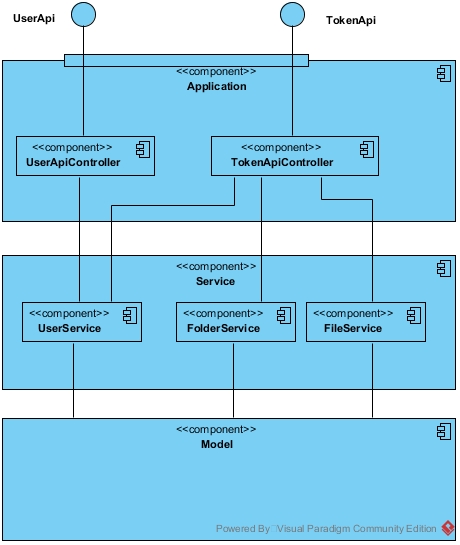
*Abb. ? Klassendiagramm vom Softwareprojekt*

Aus Überlegungen über eine optimale Datenstruktur, kristallisierten sich die Klassen User, Folder und File heraus. Der User soll alle Daten enthalten, die gebraucht werden, um ihn zu identifizieren und zu autorisieren. Daher müssen Id, Username und Password gesetzt sein. Unsere Software plant einen Login-Vorgang, daher muss für die Autorisierung auch ein entsprechendes Token generiert werden. Damit jeder Nutzer auch wirklich auf seine eigenen Ordner zugreifen kann, wurde das Home-Verzeichnis zum User hinzugefügt.

Jeder Folder besitzt eine Id um ihn eindeutig referenzieren zu können. Weil der Nutzer eigene Ordnerstrukturen anlegen kann, muss der Folder, seinen Parent kennen. Weiterhin kann jeder Folder beliebig viele Unterordner und Files besitzen. Es werden auch allgemeine Daten, wie das Erstellungsdatum, der eigentliche Pfad und die korrekte URL gespeichert. Der Unterschied zwischen Pfad und URL besteht darin, dass der Pfad den Weg von der Wurzel zum Ordner beschreibt, während bei der URL der konkrete Speicherort hinterlegt wird.

Damit die Files eindeutig bleiben, wird auch in diesem Objekt eine eindeutige Id hinterlegt. Der Parent der File ist der Ordner, in dem sich die Datei befindet. Es werden weitere Metadaten wie die Dateiendung, die Größe der Datei, das Uploaddatum, die URL und der Pfad gespeichert.

* 1. **Schnittstellenbeschreibung**



*Abb. ? Komponentendiagramm vom Softwareprojekt*

Die Planung der Schnittstellen ist eine der wichtigsten Aufgaben in diesem Projekt. Sie spiegelt die Funktionalität nach außen wider und muss daher so kompakt wie möglich sein. Während dieser Planung haben sich folgende Aufgaben für die zu erstellende Software ergeben:

* Userlogin und –logout
* Registrierung eines neuen Users
* Auflisten aller Datenstrukturen in einem Ordner
* Erstellung neuer Ordner
* Umbenennung von Ordnern
* Löschen von Ordnern
* Hochladen von Dateien
* Herunterladen von Dateien
* Ändern von Dateien
* Löschen von Dateien.

Alle diese Aufgaben sind mit REST konform zu lösen. Damit ein Userlogin und –logout korrekt ausgeführt wird, muss eine extra Schnittstelle für die Nutzerverwaltung erstellt werden.

Da der Nutzer die anderen Ordner- und Dateifunktionalitäten erst benutzen darf, nachdem er eingeloggt ist, wurden alle Operationen auf eine extra Schnittstelle verlegt. Da jeder Nutzer einen Token nach dem Einloggen erhält, wurde dieses Token als Basis genommen und somit alle Funktionalitäten unter einer Token-Schnittstelle gebündelt.

Aus diesen Informationen konnte das Komponentendiagramm erzeugt werden (siehe Abb. ??).

Für jede der beiden Schnittstellen gibt es einen entsprechenden Controller, der die entsprechenden Services aufruft. Die Services setzen die gewünschte Aufgabe um. Anschließend wird dem Nutzer ein Response zurückgeschickt.

Damit der REST-Service auch korrekt aufgerufen werden kann, müssen noch die entsprechenden URL´s aufgebaut werden.

Für die Registrierung neuer User muss die URL „/users“ aufgerufen werden. Ein POST Befehl an dieser Schnittstelle erzeugt mit korrektem Body eine Registrierung. Das eigentliche Login findet an der Schnittstelle „/users/login“ statt. Auch hier werden Nutzername und Password im Body mitgeschickt. Serverseitig wird ein neuer Token angelegt. Damit sich der Nutzer korrekt ausloggen kann, muss er die URL „/users/logout/{token}“ via DELETE-Befehl aufrufen. Durch das mitgeschickte Token kann die Software den korrekten User ausloggen.

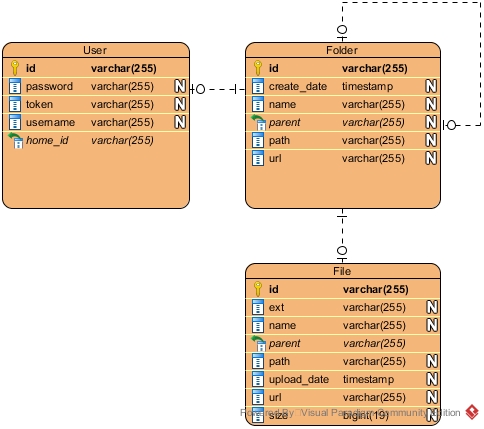
Die Folder und File-Operationen verlaufen alle ähnlich. Es werden die CRUD-Operationen via „/{token}/{folderid}“aufgerufen. Das Token authentifiziert den Nutzer und die Folderid den korrekten Speicherort. Für die Folder werden die CRUD Operationen mit entsprechenden Body genutzt.

Beim File wird die URL zu „/{token}/{folderid}/files“ erweitert. Mit einem POST-Befehl kann eine Datei zu dem bestimmten Ordner hochgeladen werden. Die restlichen CRUD-Operationen müssen auf die einzelnen Files spezialisiert werden: „/{token}/{folderid}/files/{fileid}“. Damit können Dateien heruntergeladen, geändert und gelöscht werden.[?1]

* 1. **Persistenz**

Damit der Server mit der gesamten Anwendung nicht überfordert wird, soll die Speicherung der Dateien ausgelagert werden. Dafür eignet sich der S3-Service von Amazon Web Services. Es soll eine S3 Instanz aufgebaut und eine serverseitige Verbindung hergestellt werden. Weiterhin soll diese Lösung auch die Datenbank nicht überfordern, indem dort Dateien beliebiger Größe gespeichert werden.

Für den Aufbau der Datenbank wurde überlegt, wie die Objektstruktur des Klassendiagramms effektiv in eine Datenbank gespeichert werden kann. Im Zuge der Überlegungen wurde folgendes ERM-Diagramm erzeugt (siehe Abb.??).



*Abb. ? ERM vom Softwareprojekt*

Die Tabelle User besitzt einen Fremdschlüssel von Folder. Folder kann auf sich selbst referenzieren und jede File muss auf einen Folder referenzieren.

* 1. **Erreichbarkeit**

Da dieses Projekt in Kooperation mit einem anderen Projekt steht, wurde beschlossen, dass diese Anwendung auf eine Amazon EC2 Instanz installiert werden soll, damit das andere Team stetig mit der Software arbeiten kann.

# prototyp

* 1. **Funktionalität**

Text

* 1. **Schnittstellenerzeugung**

Text

* 1. **Erweiterung der Schnittstellen (mit Datenbankerstellung)**
     1. **Encryption-Möglichkeit**

Text

* 1. **Rebuild des Projektes nach Schnittstellenbearbeitung**
     1. **Paper lesen, eine Möglichkeit wählen und im Zusammenhang mit dem Thema beschreibung**

# Ergebnisse/Probleme

# Unübersichtlicher, generierter Code

# Projektstruktur (Packages, ...)

Text

# Zusammenfassung ausblick

Text

# Literaturverzeichnis

[1] A. R. Yazdanshenas and R. Khosravi „Using domain-specific languages to describe the development viewpoint of software architectures “, veröffentlicht in „2009 14th International CSI Computer Conference “, Tehran, 2009, 146 - 151

Zuletzt aufgerufen am 19.01.2018 unter:

http://ieeexplore.ieee.org/document/5349322/

[2] offizielle Spezifikation von Swagger, OpenAPI Specification 3.0.0 Zuletzt aufgerufen am 19.01.2018

<https://swagger.io/specification/>

[?1] Von Swagger erzeugte Dokumentation des Softwareprojekts. Zuletzt aufgerufen am 19.01.2018 unter: http://34.238.158.85/MDSD-2017\_2018/doc/swagger-ui-master/dist/index.html