Erstellung einer Java Applikation zur automatisierten Erstellung von Projekten im GitLab und Jenkins Jobs

Projektarbeit Nr.:	2
vorgelegt am:	04.10.2018
Berufsakademie:	[redacted]
Studienbereich:	Technik
Studiengang:	Praktische Informatik
Ausbildungsstätte:	dotSource GmbH Goethestraße 1 07743 Jena
Gutachter Berufsakademie:	

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne und vertrauliche Informationen der Firma dotSource GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma dotSource GmbH.

I Inhaltsverzeichnis

II Tabellenverzeichnis

III Abbildungsverzeichnis

IV Abkürzungsverzeichnis

REST	Representational State Transfer
API	Application Programming Interface
WWW	World Wide Web
DSL	Domain-specific language
CLI	Command-line interface
HTTP	Hyper Text Transfer Protocol
URL	Uniform Resource Locator

1 Einleitung

Die Ausmaße von Softwareprojekten sind in den letzten Jahren stark angestiegen. Softwareprojekte werden größer, komplexer und es werden immer mehr Entwickler, welche zusammen an einem Projekt arbeiten. Um eine strukturierte sowie geordnete Entwicklung zu gewährleisten muss es möglich sein, Änderungen von Quellcode und Dokumenten zu erfassen und rückgängig zu machen. Außerdem muss die Zusammenarbeit mehrerer Entwickler koordiniert werden. Um diese Anforderungen zu erfüllen, setzt man sogenannte Versionskontrollsysteme ein, welche es ermöglichen die Änderungen an Quellcode zu verwalten und zu organisieren.[1] Die dotSource GmbH verwendet als Versionskontrollsystem GitLab, um ihre Projekte zu strukturieren und die Entwickler zu koordinieren.

Eines der Projekte der dotSource ist das C.H. Beck Projekt. Auch dieses läuft über das GitLab Versionskontrollsystem. Das Projekt erfordert häufig das Anlegen von neuen sogenannten Functions in der Microsoft Azure (siehe dazu Kapitel 2). Der Quellcode für diese muss im GitLab hinterlegt werden und dann in die Microsoft Azure deployt werden.

Die Erstellung der Projekte im GitLab für diese Functions erfolgt derzeit manuell auf einem aufwendigen und umständlichen Weg. Der Entwickler muss im GitLab diverse Einstellungen tätigen, um die Grundlage für die Functions zu erstellen. Diese Einstellungen müssen dreimal nacheinander für nahezu identische Projekte getätigt werden. Dieser Prozess nimmt sehr viel Zeit in Anspruch. Um schneller und effektiver arbeiten zu können, soll dieser Vorgang automatisiert werden. Die Automatisierung sollte in Form eines Java Programmes erfolgen, welches leicht und ohne weiteren Aufwand bedienbar sein soll.

[1] Vgl. [Bae05], S. 5

Eine Möglichkeit eine Automatisierung mittels einer Applikation zu realisieren, ist die REST API von GitLab. REST API steht für Representational State Transfer (REST) Application Programming Interface (API) und meint eine „Programmierschnittstelle, die sich an den Paradigmen und Verhalten des World Wide Web (WWW) orientiert und einen Ansatz für die Kommunikation zwischen Client und Server in Netzwerken beschreibt“[2]. Mittels dieser Schnittstelle können dann die Funktionalitäten von GitLab von einem externen Programm angesprochen werden, wodurch es wiederum möglich ist, die Schritte zur Erstellung einer Function zu automatisieren.

In dieser Arbeit soll zum einen der derzeitige Erstellungsprozess für die Functions genauer betrachtet und zum anderen wird die Entwicklung der Applikation zur Automatisierung dieses Prozesses genauer analysiert werden. Nach dieser Betrachtung soll ein Fazit gezogen werden, ob die Entwicklung des Programmes gelungen ist und ob das Programm eine Zeit- und Aufwandsersparnis für die Entwickler mit sich bringt.

[2] [Str17]

2 C.H. Beck

C.H. Beck ist einer der größten deutschen Verlage[3] und wurde im Jahre 1763 in Nördlingen gegründet. Heute liegt der Fokus des Verlages vor allem auf den Bereichen Recht – Steuern - Wirtschaft und Literatur - Sachbuch - Wissenschaft. C.H. Beck verkauft seine verlegten Werke über verschiedene Shops und Plattformen und bieten die Möglichkeit, Produkte direkt über den eigenen Webshop zu kaufen.

Die dotSource GmbH wurde beauftragt, für C.H. Beck eine Online Shop Migration durchzuführen. Es soll das bestehenden monolithischen Systems zu einem Microservice System überführt werden. Für die Microservice Umsetzung wird die Microsoft Azure verwendet. Azure ist ein Cloud Computing Dienst. Dieser verwendet sogenannte Functions. Dies sind kleinere Programme welche einzelne Teile des Shop Prozesses abbilden. Der sogenannte API Manager ruft diese dann je nach Gebrauch auf. Um eine Function zu entwickeln, muss im GitLab ein Projekt angelegt werden. Dieses Projekt durchläuft drei Entwicklungsphasen. Es beginnt mit der CI Phase. CI steht für Continious Integration und während dieses Prozesses befindet sich das Projekt noch in der internen Entwicklung, d.h. es werden die erforderlichen Funktionalitäten entwickelt. Nachdem diese Entwicklung fertig ist, geht das Projekt in die ACC Phase. ACC steht für Acceptance und in dieser Phase muss das Projekt vom Kunden abgenommen werden, der Kunde muss also entscheiden, ob das Projekt alle gewünschten Funktionen erfüllt.
Wenn der Kunde das Projekt abgenommen hat, dann geht es in die Live Phase und wird auf einem laufenden System deployt. Diese einzelnen Phasen werden im GitLab durch sogenannte Branches abgebildet. Eine Function muss von dem CI, über den ACC auf den master Branch kommen, um auf ein Live System deployt zu werden. Der deploy Prozess wurde über die Jenkins Software realisiert, deshalb war es auch notwendig Jobs im Jenkins für das Deployment anzulegen. Dieses Deployment wurde allerdings durch die GitLab CI Funktion abgelöst. GitLab bietet mit dieser Funktion eine

[3] Vgl. [Buc17]

Möglichkeit an die Software direkt zu deployen, wodurch das Erstellen von Jobs im Jenkins nicht mehr nötig ist. Auch in dieser Arbeit wird die Erstellung von Jobs im Jenkins nicht mehr betrachtet.

3 Verwendete Technologien

3.1 Gradle

Gradle ist ein auf Java basierendes Build-Management-Automatisierungs-Tool, welches eine auf Groovy (Programmiersprache) basierende domänenspezifische Sprache (DSL) zur Beschreibung der zu bauenden Projekte nutzt.[4] Der Software Build umfasst häufig das Kompilieren von Quellcode, das Verpacken der kompilierten Dateien in kompensierte Formate (z.B. JAR oder ZIP), das Erstellen von Installern und das Erstellen oder Aktualisieren von Datenbankschemen. Eine Automatisierung liegt dabei vor, wenn diese Aktionen wiederholbar sind und kein Eingreifen eines Entwicklers erfordern und keine weiteren Informationen benötigt werden, außer die bereits hinterlegten.[5] Gradle ist also ein Tool zur Automatisierung des Build Prozesses.

3.2 Git

Git ist ein freies Open Source Versionskontrollsystem. „Systeme zur Versionskontrolle dienen der Dokumentation und Wiederherstellung von Dateien. Dabei ermöglichen es Versionskontrollsysteme Änderungen an beliebigen Dateien - sowohl Text- als auch Binär-Dateien - nachzuvollziehen.“[6]

Zentrale Versionsverwaltung	Verteilte Versionsverwaltung
Alle Nutzer greifen auf ein Repository zu und jeder besitzt ein lokales Arbeitsverzeichnis, von dem aus Änderungen auf das zentrale Repository geladen werden	Jeder Nutzer besitzt ein Repository. Der Austausch findet über Anfragen zwischen den einzelnen Repositories statt.

Tab. 1: Vergleich zentrale und verteilte Versionsverwaltung

[4] Vgl. [Gra18]
[5] Vgl. [Agi]
[6] [Itw12]

Es gibt grundsätzlich zwei unterschiedliche Arten von Versionsverwaltungssystemen. Der wesentliche Unterschied zwischen den zwei Arten der Versionsverwaltung (siehe Tabelle 1) liegt in der Aufteilung der Repositories. Bei einem Repository handelt es sich um eine Art Archiv, welches Entwicklungsstände eines Projektes, also Versionen von Dateien und dazugehörige Informationen, enthält. Bei der zentralen Versionsverwaltung gibt es nur ein Repository und bei der verteilten verfügt jeder Nutzer über ein eigenes. Es ist auch in der verteilten Versionsverwaltung üblich, dass ein zentrales Repository existiert auf welchem sich der allgemeine Projektstand befindet. Git dient also dazu ein Software Projekt strukturiert zu versionieren und Änderungen Rückgängig machen zu können, bzw. auch verschiedene Projektstände von Entwicklern zu verbinden. Git gehört zu den verteilten Versionskontrollsystemen.

3.3 GitLab

GitLab ist eine weit verbreitete Software zur Versionsverwaltung von Software Projekten. Es basiert auf dem Git Versionskontrollsystem und ist in zwei verschiedenen Versionen verfügbar, der frei verfügbaren Community und der kostenpflichtigen Enterprise Version. GitLab stellt eine Reihe an nützlichen Funktionen, zur effizienten Versionskontrolle bereit.

GitLab ist von Grund auf dafür gebaut, den DevOps Prozess – „DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high qualitiy“[7] – zu unterstützen[8]. Dazu stellt es ein zentrales Repository bereit, auf welchem die Projekte liegen. Dieses wird als sogenanntes Remote Repository bezeichnet. Jeder Entwickler besitzt sein eigenes Repository, an welchem gearbeitet wird. Ist eine neue Änderung fertig, dann lädt der Entwickler diese in das GitLab und andere Entwickler können sich diese Änderung nun auf ihr eigenes Repository laden.

[7] [BWZ15]
[8] Vgl. [Git]

Wie in der Einleitung bereits erwähnt stellt GitLab außerdem noch eine REST API bereit. Mit dieser API können die meisten Web-Funktionen von Git über externe Programme angesprochen werden. Um dies zu tun kann mittels Hyper Text Transfer Protocol (HTTP) mit der API kommuniziert werden. Dazu wird der Uniform Resource Locator (URL) zum GitLab Server mit anschließender API-Version genutzt. Über diese URL wird GitLab mitgeteilt, welche Funktionalität genutzt werden soll.

3.4 GitLab4J

GitLab4J ist eine Java API, welche es ermöglicht mittels Java auf die GitLab REST API zuzugreifen.[9] Die GitLab Schnittstelle stellt diverse API's zur Verfügung, die jeweils die einzelnen GitLab Funktionen ansprechen. GitLab4J ist eine Open Source Java Bibliothek, welche über GitHub erhältlich ist. GitLab4J wird über eine Gradle Abhängigkeit in das Projekt eingebunden.

Die Funktionsweise von GitLab4J besteht darin, dass diese API diverse Klassen zur Verfügung stellt, welche jeweils die Funktionen der GitLab API's bereitstellen. Will der Entwickler ein neues Projekt erstellen, dann nutzt er die ProjectsApi Klasse und die Methode createProject(). GitLab4J generiert aus den übergebenen Parametern dann eine HTTP Anfrage und schickt diese an die GitLab API. Die entsprechenden Rückmeldungen werden dann wieder abgefangen und bei Erfolg wird meistens, je nach Anfrage, ein Objekt mit Informationen beispielsweise über das neu erstellte Projekt zurückgeliefert. Bei Fehlschlägen wird eine GitLabApiException geworfen, welche den HTTP Response Code enthält.

[9] Vgl. [Mes18]

4 Planung und Vorüberlegungen

Für die Planung und Vorüberlegungen werden für alle nötigen funktionellen Bestandteile des Programmes ermittelt. Dementsprechend werden dann die nötigen Teile der GitLab API herausgesucht. Die nötigen Funktionen von GitLab4J werden bei der Beschreibung der Umsetzung genauer betrachtet.

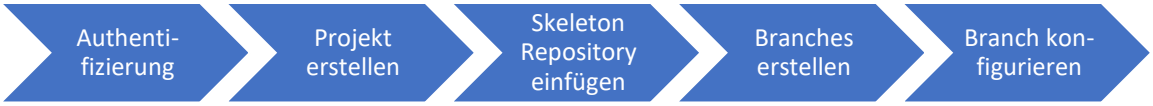

Abb. 1: Funktionsablauf

Die Abbildung 1 zeigt die zu realisierenden Kernfunktionen. Zunächst einmal muss dafür gesorgt werden, dass das Programm sich gegenüber der GitLab API authentifiziert. Die GitLab API stellt dafür diverse Methoden zur Verfügung, die wichtigste davon ist die Möglichkeit, sich über einen sogenannten personal access token zu authentifizieren. Dieser wird einfach mit in der HTTP Anfrage übergeben. Jeder Nutzer kann diesen in seinen Profil Einstellungen generieren lassen.

Im zweiten Schritt muss das Projekt erstellt werden. Dafür wird die Projects API von GitLab angesprochen. Übergibt man dieser einen Namen und Pfad, dann kann das Projekt erstellt werden. Da diese Projekte allerdings standardmäßig unter den eigenen Nutzer Namespace angelegt werden muss zusätzlich noch eine Namespace ID für den gewünschten Erstellungsort mit angegeben werden. Die Functions sollen unter einer bereits im GitLab befindlichen Gruppe angelegt werden. Eine Gruppe ist nichts weiter, als eine Sammlung von Projekten.

Nun kann das Skeleton Projekt Repository eingefügt werden. Dazu gibt es zwei Möglichkeiten, welche im Laufe der Umsetzung beide betrachtet werden. Zum einen kann man bei der Projekterstellung bereits eine Import URL mit angeben, d.h. wenn das Skeleton Projekt im GitLab liegt, kann es direkt beim erstellen des neuen Projektes importiert werden.

Eine andere Möglichkeit wäre es, die einzelnen Dateien und Ordner des Skeleton Projektes einzeln anzulegen, Dafür wird die Repository API benötigt. Diese ermöglicht es Dateien in einem Repository zu manipulieren.

Nachdem das Skeleton Repository eingefügt wird, müssen der ACC- und CI-Branch erstellt werden. Dazu kann ebenfalls die Repository API verwendet werden, welche die Möglichkeit Create Branch bereitstellt. Um die Zugriffe auf einen Branch zu konfigurieren, bzw. sie auf protected zu setzen muss dann die Protected Branches API genutzt werden, welche die Branches protected und außerdem Möglichkeiten bietet die Push und Merge Rechte zu konfigurieren. Also die Rechte, Änderungen an einem Branch vorzunehmen.

Diese Funktionalitäten stellen die Kernfunktionen des Programmes dar. Was allerdings noch fehlt ist eine Möglichkeit das Programm ordentlich auszuführen. Hierzu soll im ersten Entwurf die Applikation über das Command Line Interface (CLI) angesprochen werden, es ist also ebenfalls eine Nutzerinteraktion zu erstellen. Die Realisierung der Anwendung soll mittels Java geschehen.

5 Umsetzung

In diesem Kapitel wird die Umsetzung des Programmes betrachtet, dazu wird zuerst die Entwicklung der bereits in Kapitel 4 erwähnten Funktionen betrachtet. Hierbei wird sowohl die Code Entwicklung, als auch auftretende Probleme und Lösungswege beschrieben. Nach der Betrachtung der Kernfunktionen wird die Erstellung der Nutzerinteraktion analysiert und auch hier werden wieder auftretende Probleme und Lösungswege aufgeführt. Zum Schluss wird zusammenfassend die Projektstruktur betrachtet und eine Zusammenfassung der Klassen verfasst.

5.1 Projekt Setup

Um die Applikation zu Erstellen und GitLab4J nutzen zu können muss zunächst Gradle eingerichtet werden. Dazu wird einfach der Gradle ZIP Ordner heruntergeladen und entpackt. Nun muss nur noch in den Systemeinstellungen der Pfad zur bin Datei als Environmental Variable konfiguriert werden. Gradle ist damit nutzungs- und einsatzbereit.

Für die Entwicklung des Programmes wird die IntelliJ IDE genutzt. Diese stellt einige hilfreiche Funktion für Gradle Projekte bereit. IntelliJ erlaubt es einem beispielsweise ganz einfach neue Gradle Projekte zu erstellen und alle nötigen Dateien und Konfigurationen automatisch zu treffen. Außerdem kann IntelliJ für das neue Gradle Projekt alle nötigen Abhängigkeiten automatisch aktualisieren.

Um nun GitLab4J in das Projekt einzubinden, muss in der build.gradle Dateien unter dem Punkt dependencies nur noch eine kleine Konfiguration getroffen werden (siehe Abbildung 3). Nachdem Gradle die Abhängigkeiten aktualisiert hat, können alle Funktionen von GitLab4J genutzt werden. Der eigentliche Programmcode wird unter src/main/java abgelegt und weitere nötige Dateien werden unter dem resources Verzeichnis (siehe Abbildung 2) platziert.

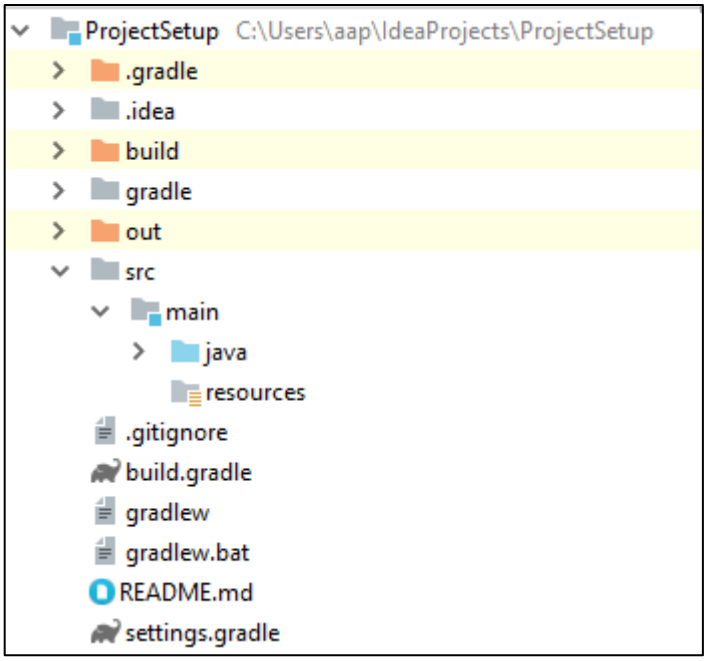

Abb. 2: Gradle Projekt

```
dependencies {
    compile group: 'org.gitlab4j', name: 'gitlab4j-api', version: '4.8.37'
}
```

Abb. 3: Gradle Konfiguration

5.2 Authentifizierung

Als erstes wird die Verbindung zum GitLab und die Authentifizierung realisiert. Um eine Verbindung mit der REST Schnittstelle zu bekommen, wird in GitLab4J die GitLabApi Klasse bereitgestellt. Dieser übergibt man beim instanziieren als Parameter die URL zu dem Server auf dem GitLab läuft und das bereits erwähnte personal access token. Es gibt auch die Möglichkeit die Authentifizierung über eine Login Funktion der GitLabApi Klasse zu machen. Dieser Funktion müssten dann der Nutzername und das dazugehörige Passwort des Nutzers übergeben werden. In diesem Fall wurde die Token Methode gewählt, da die Realisierung damit einfacher war.

Das einzige Problem bei der Authentifizierung über den Token ist, dass bei der Instanziierung der Klasse keine Fehlermeldung zurückkommt, wenn das Token falsch ist. Es würde erst beim ausführen von Methoden, welche HTTP Anfragen an das GitLab senden eine Exception geworfen werden. Mit der Login Methode über den Nutzernamen und das Passwort hingegen würde sofort eine Exception im Programm auftreten. Um dieses Problem abzufangen wird bereits nach der Erstellung der Instanz der derzeitige Nutzer abgefragt. Wenn die Authentifizierung nicht gelungen ist, wird an dieser Stelle eine Exception geworfen.

```
private GitLabApi gitLabApi;

public ApiHelper(final String token) {
    gitLabApi = new GitLabApi(HOST_URL, token);
}

public boolean isUserAuthenticated() {
    try {
        gitLabApi.getUserApi().getCurrentUser();
        return true;
    }
    catch(GitLabApiException e) {
        System.out.println("ERROR " + e.getHttpStatus() + ": " + e.getReason());
        return false;
    }
}
```

Abb. 4: Nutzer Authentifizierung

Um den Code zu strukturieren wird die Klasse ApiHelper erstellt. Diese soll alle möglichen Funktionen zur Interaktion mit GitLab4J zur Verfügung stellen. Wie in der Abbildung 4 zu erkennen ist, befindet sich eine Methode und ein Konstruktor in der Klasse. Der Konstruktor dient dazu eine GitLabApi Instanz zu erzeugen, welche dann von den anderen Funktionen innerhalb der ApiHelper Klasse genutzt werden kann. Die Methode isUserAuthenticated() prüft, wie bereits erwähnt, ob der aktuelle Nutzer abgerufen werden kann und wirft andern falls eine Exception. Damit ist die Grundlage für die weiteren Funktionalitäten geschaffen.

5.3 Projekt Erstellung und Skeleton Import

Nachdem die Authentifizierung erstellt wurde, wird nun die Erstellung eines Projektes in GitLab realisiert. Für die Erstellung wird ein Skeleton Projekt benutzt. Dieses Skeleton Projekt ist eine Vorlage, welche die nötigen Dateien beinhaltet, um eine Function zu erstellen. Um zunächst einmal das Projekt an sich zu erstellen, wird die Projects API von GitLab4J verwendet. Diese stellt einige Funktionen zur Projekterstellung bereit, welche jeweils die Projects API von GitLab ansprechen und je nach Parametern ein neues Projekt erstellen. Es wird auch eine Funktion bereitgestellt, die beim Erstellen von Projekten den Import des Repositories eines anderen Projektes ermöglicht. Für diese Funktion muss die URL zu dem Repository übergeben werden.

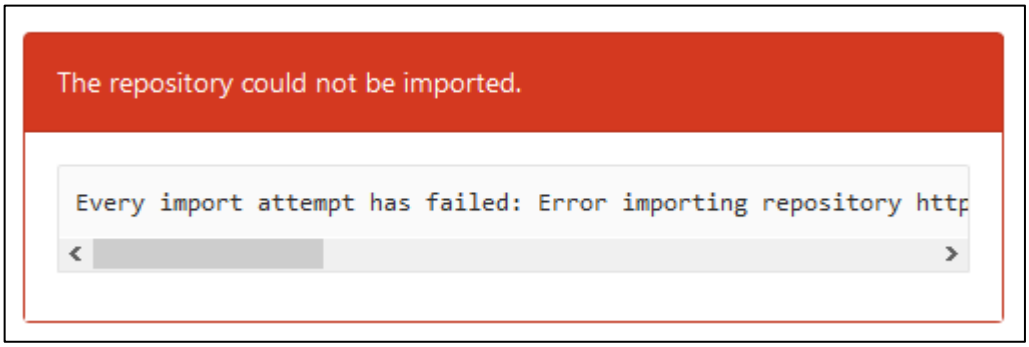

Abb. 5: Projekt Import Fehlermeldung

Hier stellte sich eines der ersten Probleme des Projektes heraus. Um einen Import durchzuführen muss der URL zusätzlich noch der aktuelle Benutzername und das Passwort übergeben werden. Die URL muss dann diese Form haben:
https://username:password@gitlab.company.com/group/project.git

Das Problem, hierbei ist, dass GitLab4J bevor es eine Anfrage an den Server mit der URL verschickt diese noch validiert. Sollte der Nutzername oder das Passwort Sonderzeichen beinhalten, dann kommt es zu einem Fehler (siehe Abbildung 5). Um das Problem zu umgehen wird zunächst die Commit API genutzt.

```
private String getFileContent(File file) {
    try {
        return new String(Files.readAllBytes(Paths.get(file.getAbsolutePath())));
    }
    catch(IOException e) {
        System.out.println("ERROR: " + e);
    }
    return null;
}

List<GitFile> getFilesUnderDirectory(List<GitFile> gitFileList, String directoryPath) {
    File[] directoryFiles = new File(directoryPath).listFiles();
    if(directoryFiles == null) {
        return null;
    }
    for(File file : directoryFiles) {
        if(file.isDirectory()) {
            getFilesUnderDirectory(gitFileList, file.getPath());
        }
        else {
            //check encoding to upload zip
            GitFile gitFile = new GitFile(
                    file.getPath().substring(defaultPath.length() + 1).replace( target: "\\",  replacement: "/"),
                    getFileContent(file)
            );
            gitFileList.add(gitFile);
        }
    }
    return gitFileList;
}
```

Abb. 6: FileHelper Klasse

Die Commits API ermöglicht es Dateien im Git über einen Commit zu erstellen. Geplant ist daher, dass Skeleton Projekt lokal zu Speichern und dann die einzelnen Dateien in das GitLab hochzuladen. Ein weiteres Problem, welches sich hier herausstellt ist, dass für die Erstellung einer Datei der Inhalt als String übergeben werden muss, d.h. das Ganze ist mit dem Aufwand verbunden jede einzelne Datei auszulesen. Um dies zu realisieren wird eine Klasse geschrieben, welche dies übernehmen soll.

Die FileHelper Klasse (siehe Abbildung 6) erhält im Wesentlichen zwei Funktionen, mit deren Hilfe nötige Informationen aller Dateien aus einem Verzeichnis ausgelesen werden können. Ein Commit über den Befehl createCommit() in der Commits API benötigt eine ID für das Zielprojekt, diverse Angaben über den Commit, also Beschreibung, Autor, usw. und zudem eine CommitAction Liste. In dieser Liste befinden sich alle Aktionen, welche commited werden sollen. In den CommitAction Objekten muss zunächst definiert werden, was passieren soll. Dabei gibt es vier Möglichkeiten: Eine Datei erstellen, eine Datei ändern, eine Datei löschen oder eine Datei verschieben. In diesem Fall müssen Dateien erstellt werden. Zusätzlich dazu müssen noch ein Dateipfad und der Inhalt der zu erstellenden Datei angegeben werden. Es werden zunächst die Dateien im Skeleton ausgelesen, in einer Liste gespeichert und anschließend werden mit Hilfe dieser Informationen CommitAction Objekte erzeugt, welche in einer Liste dem Commit als Parameter übergeben werden. Nun wird noch ein Ziel Projekt benötigt. Die bereits erwähnte Methode zur Projekt Erstellung kann auch ohne die Import URL verwendet werden und wird daher auch hier verwendet. GitLab4J liefert immer Informationen über das beispielsweise erstellte Projekt zurück, wodurch auch die ID für den Commit Parameter erfassbar ist. Beim Auslesen eines Verzeichnisses werden der Dateipfad und der Inhalt in einem GitFile Objekt festgehalten, diese gehören zu einer Klasse, welche allein für diesen Zweck erstellt wurde. Nun kann eine Commit Anfrage an GitLab erstellt werden, welche alle Dateien erstellt.

Diese Methodik hat das Problem, dass keine Binärdateien committed werden können. Dies hat zur Folge, dass beispielsweise ausführbare oder komprimierte Dateien nach dem Commit nicht mehr verwendbar, bzw. beschädigt sind. Das Skeleton Projekt hatte zunächst keine Binärdateien. Erst im weiteren Projektverlauf wurde das Skeleton allerdings geändert und es kamen JAR Dateien hinzu. Damit ist die provisorische Lösung nicht mehr möglich.

Nach weiterer Betrachtung der GitLab REST API, konnte festgestellt werden, dass die URL zum importieren des Projektes statt dem Passwort auch alternativ den Token beinhalten kann. Der Token besteht nur aus alphanumerischen Zeichen und kann daher auch in GitLab4J verwendet werden. Das Projekt anlegen wird daher umgebaut und der FileHelper wieder entfernt. Das Skeleton Projekt kann im GitLab hinterlegt werden und GitLab importiert automatisch das Repository beim erstellen des Projektes. Mittels dieses Vorgehens traten weitere Probleme auf. Zum einen werden bei einem Import die Projekteinstellungen nicht mit übernommen, dies ist erstmal nicht weiter tragisch, allerdings erfordert das Skeleton Projekt eine Konfiguration der CI Einstellungen. Die Einstellungen für das CI sind nötig für den Deployment Prozess. Die Einstellungen müssen manuell getroffen werden, da die GitLab API und somit auch GitLab4J keine Anbindung an die GitLab CI Einstellungen bereitstellt. Dies kann also nicht automatisiert werden und muss manuell vom Entwickler nach Ausführung des Programmes erledigt werden.

Zum anderen wird beim Ausführen der createProject() Methode eine Anfrage an den Server gesendet. Der Server verarbeitet diese und noch bevor der Repository Import vollständig abgelaufen ist, wird eine Rückmeldung an GitLab4J gesendet. D.h. im Programm ist an diesem Punkt nicht festzustellen, ob der Import bereits beendet wurde, sicher ist nur, dass das Projekt als solches erstellt worden ist. Um dieses Problem zu umgehen wird ein Timeout eingebaut, welcher prüft ob eine bestimmte Datei – welche definitiv in dem Skeleton vorhanden sein muss – vorhanden ist. Sollte dies nicht der Fall sein, dann wird 3 Sekunden gewartet und die Anfrage erneut gestartet. Wenn das Ganze beim dritten Versuch immer noch kein positives Ergebnis zurückliefert, dann wird eine Exception geworfen und das Programm wird an dieser Stelle beendet.

5.4 Branch Erstellung und Konfiguration

Nachdem nun das Projekt erstellt werden kann und das Repository des Skeletons importiert wird, können nun die Branches erstellt und konfiguriert werden. Dazu wird die Repository API angesprochen und der Befehl createBranch() genutzt. GitLab erstellt auf Basis des Master Branch den CI und ACC Branch. Diese beiden Branches müssen noch Zugriffsrechte erhalten.

```
public List<ProtectedBranch> createProtectedBranches(final Project targetProject) {
    return Arrays.asList("RELEASE_ACC", "RELEASE_CI").stream()
            .map(branchName -> {
                try {
                    gitLabApi.getRepositoryApi().createBranch(targetProject.getId(), branchName, ref: "master");
                    return gitLabApi.getProtectedBranchesApi().protectBranch(
                            targetProject.getId(),
                            branchName,
                            AccessLevel.MAINTAINER,
                            AccessLevel.DEVELOPER
                    );
                }
                catch(GitLabApiException e) {
                    System.out.println("ERROR " + e.getHttpStatus() + ": " + e.getReason());
                    return null;
                }
            })
            .collect(Collectors.toList());
}
```

Abb. 7: Funktion für Branch Zugriffsrechte

Dazu wird die Protected Branches API genutzt. Mit dieser können die Zugriffsrechte für Merge und Push Anfragen geändert werden. Push wird dabei auf Maintainer gesetzt und Merge auf Developer. Dies wird für alle drei Branches einzeln konfiguriert, d.h. jeder wird nacheinander einzeln geschützt (siehe Abbildung 7). Es ist auch möglich alle Branches über eine sogenannte Wildcard gleichzeitig zu schützen, allerdings kommt es dabei zu einer doppelten Konfiguration des Master Branches, welche vermieden werden soll.

5.5 Nutzerschnittstelle

Die Kernfunktionen für das Programm sind erstellt worden. Jetzt muss noch ein User Interface erstellt werden. D.h. es wird eine neue Klasse benötigt, welche über die CLI die nötigen Daten, wie beispielsweise den personal access token, vom Nutzer abfragt und dann die Funktionen der ApiHelper Klasse aufruft und ihnen die Daten als Parameter übergibt. Weiterhin bekommt der Nutzer Rückmeldung, ob die ausgeführten Aktionen erfolgreich waren oder Fehler aufgetreten sind.

Die Klasse muss also im Wesentlichen nacheinander die Authentifizierung, die Erstellung des Projektes und die Erstellung der Branches sowie deren Konfiguration erledigen. Dabei sollte zum einen nach einem Fehler die Möglichkeit gegeben sein, das Ganze zu wiederholen. Beispielsweise sollte es also nach einem fehlerhaften Login möglich sein, diesen zu wiederholen. Zum anderen sollte es zum Zeitpunkt der Nutzereingabe möglich sein, das Programm zu beenden. Der Nutzer muss also an jeder Stelle, an der eine Eingabe von ihm gefordert ist auch eine Möglichkeit haben, das Programm zu beenden.

Zunächst wird die Authentifizierung ermöglicht. Dafür müssen vom Nutzer der Nutzername und das personal access token abgefragt werden. Im Falle, dass die Authentifizierung nicht erfolgreich sein sollte muss der Nutzer die Möglichkeit haben, den Authentifizierungsprozess zu wiederholen. Dafür wird zunächst eine Hauptmethode run() geschrieben. Diese Funktion soll die notwendigen Daten vom Nutzer abfragen und anschließend die ApiHelper Funktionen aufrufen. Für das abfragen der Nutzereingabe wird eine neue Funktion authenticateUser() geschrieben. Diese gibt zunächst aus, dass es sich hierbei um den Authentifizierungsprozess handelt und fragt dann eine Nutzereingabe zum Nutzernamen und Token ab. Des Weiteren wird eine Programmabbruch Bedingung benötigt. Dazu wird geprüft ob die Eingabe des Nutzers gleich „#exit“ ist. Gibt der Benutzer diese Zeichenkette ein, dann wird das Programm beendet.

Um nun zu prüfen ob der Nutzer mit diesen Daten angemeldet werden kann, wird eine ApiHelper Instanz erstellt und die Methode isUserAuthenticated() aufgerufen. Sollte der Nutzer erfolgreich authentifiziert worden sein, dann wird die Instanz zurückgegeben. Im Fehler Fall wird eine Exception geworfen. Der Nutzer bekommt dann die Meldung, dass der Login fehlgeschlagen ist. Um eine Wiederholung zu ermöglichen, wird im Fehlerfall die Methode nochmal aufgerufen.

Nun benötigt das Programm einen Namen, für das neu zu erstellende Projekt. Dafür wird ebenfalls eine neue Funktion geschrieben, welche nahezu demselben Schema folgt wie die Funktion zur Nutzerdaten Abfrage. Der einzige Unterschied ist, dass kein Fehler geworfen wird, d.h. jede Nutzereingabe ist valide. Sollte es zu unzulässigen Projektbezeichnungen kommen, wird spätestens bei der Projekterstellung ein Fehler geworfen.

Jetzt wird eine Funktion zum erfassen des Speicherortes des Projektes geschrieben. Diese dient lediglich dazu die ID der Gruppe im GitLab, unter der das Projekt angelegt werden soll, einzubinden. Die ID wird in einer globalen Variable gespeichert und von dieser Methode aufgerufen. Dies ist vor allem nützlich, wenn es zu einer Änderung des Speicherortes kommen sollte, dann muss nur die globale Variable geändert werden.

Nachdem alle Nutzerdaten erfolgreich abgefragt wurden, ruft die run() Methode zunächst die createProject() Methode auf und schaut mit Hilfe eines Timeouts ob der Import des Skeleton Repository erfolgreich war. Sollte das Projekt nicht erstellt werden können, oder der Import nicht erfolgreich ist, dann wird eine Fehlermeldung ausgegeben und das Programm beendet. Sollte beides erfolgreich gewesen sein, dann wird die createProtectedBranches() Methode aufgerufen. Bei Erfolg und Misserfolg wird jeweils eine Meldung ausgegeben und das Programm beendet.

5.6 Projekt Überblick

In diesem Kapitel wird noch einmal die finale Projektstruktur dargestellt und die Funktionalität der fertigen Klassen zusammengefasst.

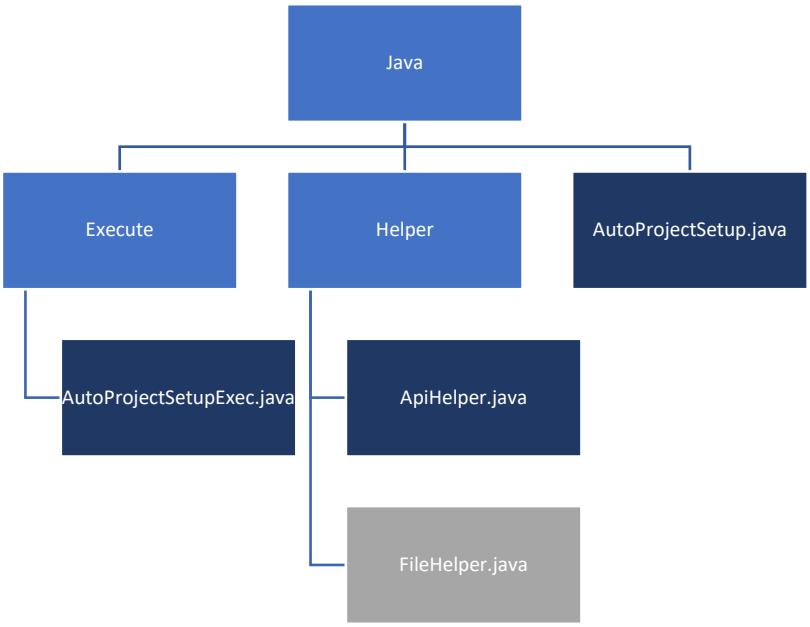

Abb. 8: Projekt Struktur

In der Abbildung 8 sieht man die Projektstruktur. Die Java Dateien enden jeweils auf „.java“. Grau ist dabei die Datei, welche im Verlaufe des Projektes wieder entfernt wurde. Die Java Packages wurden hellblau markiert. Direkt unterhalb des Java Package befindet sich die AutoProjectSetup Klasse. Diese dient lediglich dazu die Ausführung des Programmes zu starten, d.h. sie ruft die run() Methode aus der AutoProjectSetupExec Klasse auf. Des Weiteren befinden sich die beiden Packages Execute und Helper unter dem Java Package. Die AutoProjectSetupExec Klasse stellt die Schnittstelle zwischen den ApiHelper Funktionen und der Abfrage der Nutzereingaben bereit.

Unter dem Helper Package befindet sich die ApiHelper Klasse, welche für die Interaktionen mit der GitLab REST API zuständig ist. Außerdem befand sich dort eine Klasse, welche die Interaktionen mit Dateien ermöglichen sollte. Im ApiHelper (siehe Abbildung 10) befinden sich Methoden zur Nutzer Authentifizierung, zur Projekterstellung und zur Branch Erstellung sowie zur Konfiguration mit GitLab4J.

```
public class ApiHelper {

    private static final String HOST_URL = "https://gitlab.example.de";
    private static final String SKELETON_PATH = "@gitlab.example.de/functions.git";
    private String importUrl;

    private GitLabApi gitLabApi;

    public ApiHelper(final String userName, final String token) {...}

    public boolean isUserAuthenticated() {...}

    public Project createProject(final String projectName, final int groupId) {...}

    public List<ProtectedBranch> createProtectedBranches(final Project targetProject) {...}

    public boolean isProjectCreated(final int projectId) {...}
}
```

Abb. 9: ApiHelper Klasse

```
public class AutoProjectSetupExec {

    private static final String CONSOLE_SEPARATOR_LINE = "====================";
    private static final String EXIT_STRING = "#exit";

    private final int FUNCTIONGROUPID = 1;
    private final int LAMBDAGROUPID = 2;

    private Scanner scanner = new Scanner(System.in);

    public void run() {...}

    private ApiHelper authenticateUser() {...}

    private int getGroupId() {...}

    private String getProjectName() {...}

}
```

Abb. 10: AutoProjectSetupExec Klasse

6 Fazit

Im Rahmen dieser Projektarbeit wurde eine Java Applikation erstellt, welche die Entwicklungsprozesse innerhalb des C.H. Beck Projektes der dotSource GmbH optimiert. Dazu wurde zunächst betrachtet, wie der aktuelle Prozess innerhalb des Projektes abläuft. Nach dieser Betrachtung wurden zunächst einmal die Grundlagen für das weitere Vorgehen, sowie die zu nutzenden Technologien beschrieben. Durch die Schaffung dieser Informationsgrundlage konnte daraufhin die Planung der Kernfunktionen und die absehbaren zu nutzenden Technologien analysiert werden. Auf Basis dieser Planung wurde dann die Umsetzung durchgeführt und beschrieben.

Die Umsetzung der Kernfunktionen lief nach den Vorüberlegungen aus der Planung ab. Hier wurde zuerst die Nutzer Authentifikation erstellt, welche den Nutzer gegenüber der GitLab API authentifiziert. Nachdem dies realisiert wurde, konnte mit der Erstellung des Projektes fortgefahren werden. Die Projekterstellung realisierte gleichzeitig den Skeleton Import, welcher nötig war um die Grundlage für die Azure Functions zu bilden. Nachdem dieser Teil erfolgreich erstellt werden konnte, wurde die Erstellung und Konfiguration der Branches entwickelt. Damit war die Automatisierung des Projekterstellungsprozesses abgeschlossen.

Durch diverse Änderungen am C.H. Beck Projekt während der Entwicklungszeit kam es in diesem Projekt zu einigen Problemen bzw. Umstellungen. Zum einen wurde die Aufgabe, zur Automatisierung, wesentlich einfacher, nachdem der Deployment Prozess von Jenkins auf die GitLab CI Funktionalität umgestellt wurde. Dadurch lag in diesem Bereich bereits eine Automatisierung vor. In dieser Arbeit wurde daher auch keine Erstellung von Jenkins Jobs betrachtet, da diese nun redundant geworden sind. Ein weiteres Problem äußerte sich durch die Änderung des Skeletons für die Azure Functions, denn dadurch mussten einige Programmteile entfernt werden, welche in der Entwicklung viel Zeit in Anspruch genommen haben.

Zusammengefasst betrachtet, konnte die gestellte Anforderung realisiert werden. Durch die Applikation ist eine Automatisierung des Projekterstellungsprozesses möglich und somit kann auch eine zeitliche Optimierung für das Projektvorgehen erreicht werden. Außerdem lässt sich das Programm auch für zukünftige Projekte mit ähnlichen Rahmenbedingungen wiederverwenden.

Literaturverzeichnis

[Agi] Agile Alliance, „Automated Build”, o.O., o.J.
https://www.agilealliance.org/glossary/automated-build
Abruf am: 03.09.2018

[Bae05] Baerisch, S.: „Versionskontrollsysteme in der Softwareentwicklung”, Informationszentrum Sozialwissenschaften der Arbeitsgemeinschaft Sozialwissenschaftlicher Institute e.V. (ASI), Bonn, 2005

[Buc17] Buchreport, „Die 100 größten Verläge”, o.O., 2017.
https://www.buchreport.de/die-100-groessten-verlage/
Abruf am: 19.09.2018

[BWZ15] Bass, L., Weber, I., Zhu, L., „DevOps: A Software Architect's Perspective”, 1. Auflage, Addison Wesley, o.O. 2015

[Git] GitLab, „Integrated teams working together”,. o.O., o.J.
https://about.gitlab.com/
Abruf am: 28.08.2018

[Gra18] Gradle, „Gradle User Manual”, o.O., 2018.
https://docs.gradle.org/current/userguide/userguide.html
Abruf am: 28.08.2018

[Itw12] ITWissen, „Versionskontrolle”, o.O., 2012.
https://www.itwissen.info/Versionskontrolle-version-control-VCS.html
Abruf am: 03.09.2018

[Mes18] Messner, G. „GitlLab API for Java (gitlab4j-api)”, o.O., 2018.
https://github.com/gmessner/gitlab4j-api
Abruf am: 28.08.2018

[Str17] Strocke, D., „Was ist eine REST API?”, o.O., 2017
https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/
Abruf am: 03.09.2018

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Studienarbeit mit dem Thema:

   Erstellung einer Java Applikation zur automatisierten Erstellung von Projekten im GitLab und Jenkins Jobs

   ohne fremde Hilfe angefertigt habe,
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

_____ _____

Ort, Datum Unterschrift