Inhaltsverzeichnis

1 Motivation

2 Einleitung

2.1 Aufgabenbeschreibung

Am Anfang stand die Idee, dass ich mich immer tiefer mit Automatisierung beschäftige. Deshalb fiel die Wahl auf ein Projekt in diese Richtung. Im Rahmen dieser wissenschaftliche Vertiefung Modul wird ein auf Docker basierendes Testsystem entwickelt. Dieses Test-Framework soll die studenten helfen, ihre Übungsaufgaben automatisiert zu testen.
Der Student lädt seinen Quellcode über eine Web-Benutzeroberfläche hoch, dieser Service steht den Studenten nach der Registrierung auf der Testplattform zur Verfügung.Für mich ist die Studenten Registrierung in der Tat eine Erstellung eines eigenen Remote-Testraums, der dadurch der Testprozess verwaltet wird, und das, was ich in dieser Ausarbeitung im Detail erklären werde. Nach Abschluss des Testprozesses erhält der Student einen Bericht mit dem Ergebnis seines Tests. Darüber hinaus sollte dieses Test-Framework der Lehrkraft die Möglichkeit geben, die individuellen Testaufgaben der Studenten zu kontrollieren.

2.2 Herangehensweise

In diesem Abschnitt werde ich die Techniken und Technologien vorstellen, die in diesem Projekt verwendet werden. Daher wurde das Konzept des Code-Testens erläutert. Ebenso wird das in diesem Projekt verwendete Unit-Test-System besprochen. Anschließend werden die Rolle des Jenkins-Tools im Projekt und die Bedeutung dieses Tools für die Automatisierung des Testprozesses erläutert, dann folgt eine ausführliche Erläuterung der Verwendung von Docker im Projekt. Abschließend eine Erläuterung von Git-Hosting und Jgit, die einen der wichtigsten Teile des Projekts darstellen.
Im Hauptabschnitt wird dann die Erstellung der Testplattform erläutert. Es wird die Architektur beschrieben, und die Entscheidungen, die getroffen wurde, indem die Techniken ausgewählt wurde, mit denen das Projekt die Anforderungen erfüllt. Außerdem werden der Ablauf des Testverfahrens und die internen Prozesse des Bestandteils der Testplattform vorgestellt. Dann werden die Installationsanweisungen und Verzerrungen der Testplattform klarstellen.

3 Grundlagen

3.1 Code Testing

Beim Softwaretesten wird ein Programm oder eine Anwendung ausgeführt, um die Fehler zu ermitteln und um zu überprüfen, ob die erforderlichen Bedingungen erfüllt sind.
Softwaretestverfahren können in 2 Gruppen unterteilt werden:Statisches / Statisches Testen.

3.1.1 Manual testing(Statisches)

Eine Softwaretesttechnik, bei der die Software getestet wird, ohne den Code auszuführen. Es besteht aus zwei Teilen:

- Review - Wird normalerweise verwendet, um Fehler oder Unklarheiten in Dokumenten wie Anforderungen, Design, Testfällen usw. zu finden und zu beseitigen.
- Statische Analyse - Der von Entwicklern geschriebene Code wird (normalerweise mithilfe von Werkzeugen) auf strukturelle Fehler analysiert.

3.1.2 Automated Testing(Dynamisches)

Beim dynamischen Testen wird die Software auf die Eingabewerte geprüft und die Ausgabewerte analysiert. Es gibt verschiedene Stufen dynamischer Testtechniken:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

In diesem Fall verwenden wir eines Dynamisches Testen, um die Programmieraufgaben der Studenten zu bewerten.

3.2 Das Unit Testsystem

3.3 Jenkins

3.3.1 Einführung

3.3.2 Jenkins buildflow

3.3.3 Jenkins pipeline

3.3.4 Jenkins multijob

3.3.5 Jenkinsfile

3.4 Docker

3.4.1 Einführung

3.4.2 Docker Architektur

Nach meiner Erfahrung besteht eine der einfachsten Möglichkeiten, Docker zu verstehen und anzuwenden, darin, sich einen Überblick darüber zu verschaffen, wie die Docker-Plattform im

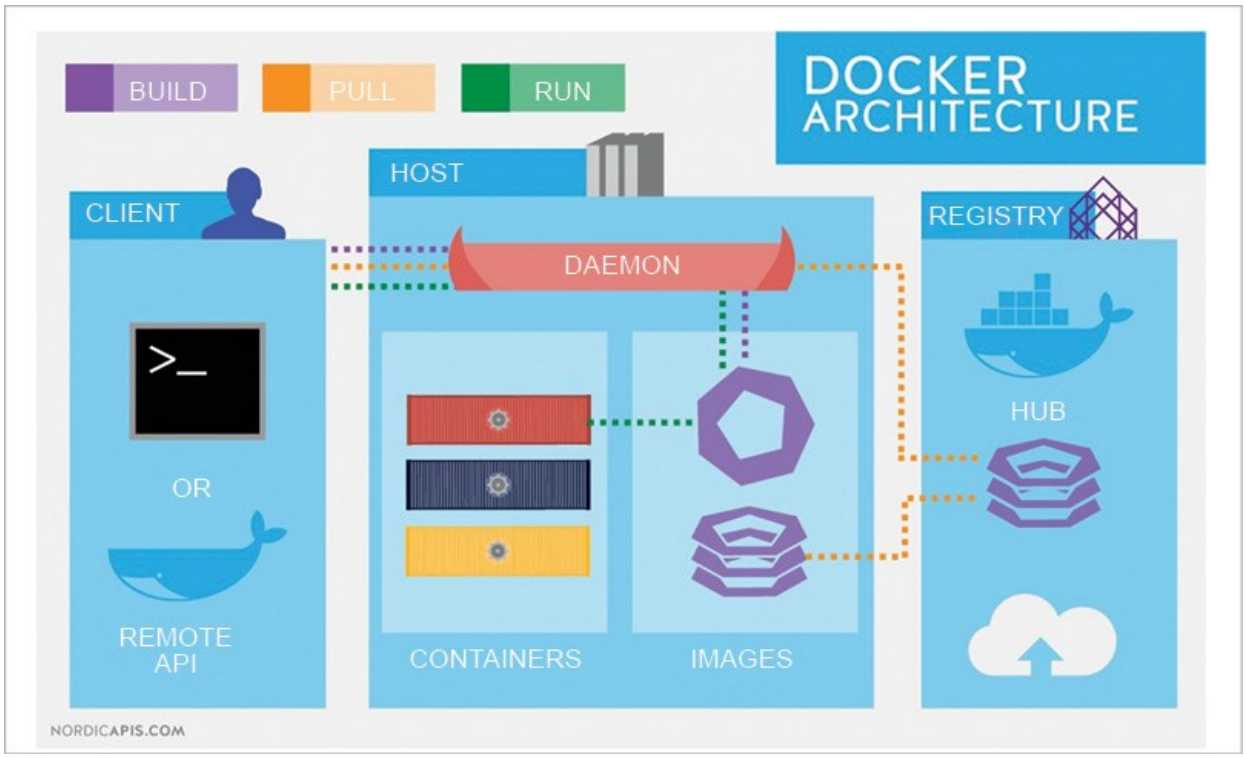

Abbildung 1: Docker Architektur

Hintergrund aufgebaut ist.

In Abbildung 1 sehen Sie die Hauptkomponenten einer Docker-Installation:

- Im Zentrum steht der Docker-Daemon, der für das Erstellen, Ausführen und Überwachen von Containern sowie für das Erstellen und Speichern von Images zuständig ist. Der Docker-Dämon wird vom Host-Betriebssystem verwaltet.
- Der Docker-Client befindet sich auf der linken Seite und wird für die Kommunikation mit dem Docker-Daemon über HTTP verwendet. Ein Docker-Client kann mit mehr als einem Daemon kommunizieren, da sich der Docker-Client auf demselben Host wie der Daemon oder über eine Verbindung auf einem Remote-Host befinden kann. Der Docker-Client bietet eine Befehlszeilenschnittstelle (Command Line Interface, CLI), mit der Sie Anwendungsbefehle für einen Docker-Dämon erstellen, ausführen und stoppen können.
- Docker-Registrys speichern und verteilen images, die Standardregistrierung heißt Docker Hub und hostet Tausende von öffentlichen Images. Viele Unternehmen führen eigene Register, in denen kommerzielle oder vertrauliche Images gespeichert werden können. Der Docker-Daemon lädt als Reaktion auf Docker-Pull-Requests Images von Registrys herunter. Es werden auch automatisch Images heruntergeladen, die in Docker-Run-Requests und in der FROM-Anweisung von Dockerfiles angegeben sind, wenn sie nicht lokal verfügbar sind.

Wenn in diesem Abschnitt etwas nicht verstanden wird, wie z. B. Bilder, Dockerfile ..., machen Sie sich keine Sorgen, da in den folgenden Abschnitten alles im Detail erläutert wird.

3.4.3 Docker Images

3.4.4 Docker Container

3.4.5 Docker Volumes

3.4.6 Data Container

3.4.7 Docker Compose

3.4.8 DockerFile

3.5 Git-Hosting mit Gogs

3.5.1 Einführung

3.5.2 Datenbankeinstellung

3.5.3 JGit

3.5.4 Webhook

4 Test-C Plattform

4.1 Einführung

In diesem Abschnitt wird die Test C-Plattform detailliert beschrieben, wobei zunächst die Zusammensetzung der gesamten Programmarchitektur erörtert wird, anhand derer die Programmkomponenten separat diskutiert werden können. Nach der Erklärung der Komponenten des Programms auf separate Weise, wird im Abschnitt (Interne Prozesse) genau erläutert, wie sie zusammenarbeiten.
Wenn der Leser das oben Genannte versteht, kann er verstehen, was ich im Abschnitt (Ablauf des Testverfahrens) erläutert wird.

4.2 Architektur

Wie in der Abbildung 2 gezeigt, ist die Test-C-Plattformarchitektur in drei Hauptteile unterteilt: Client-Seite, Serverseite und Remote-TestUmgebung.
Die Wahl der Technologien beruhte auf ihrer Modernität und ihrer Verbreitung auf dem Arbeitsmarkt. In Teil eins des Projekts (Client-Seite/Frontend) wurde die folgenden Techniken verwendet : Angular 7, NGX-Bootstrap, Html, Css, im zweiten Teil des Projekts (Serverseite / BackEnd) wurden Maven und Spring Boot, Spring Security und Ldap ... verwendet. Der dritte Teil des Projekts, in dem der Testprozess durch Jenkins abgeschlossen ist.
Im folgenden Abschnitt werde ich jeden Teil des Architektur separat erklären.

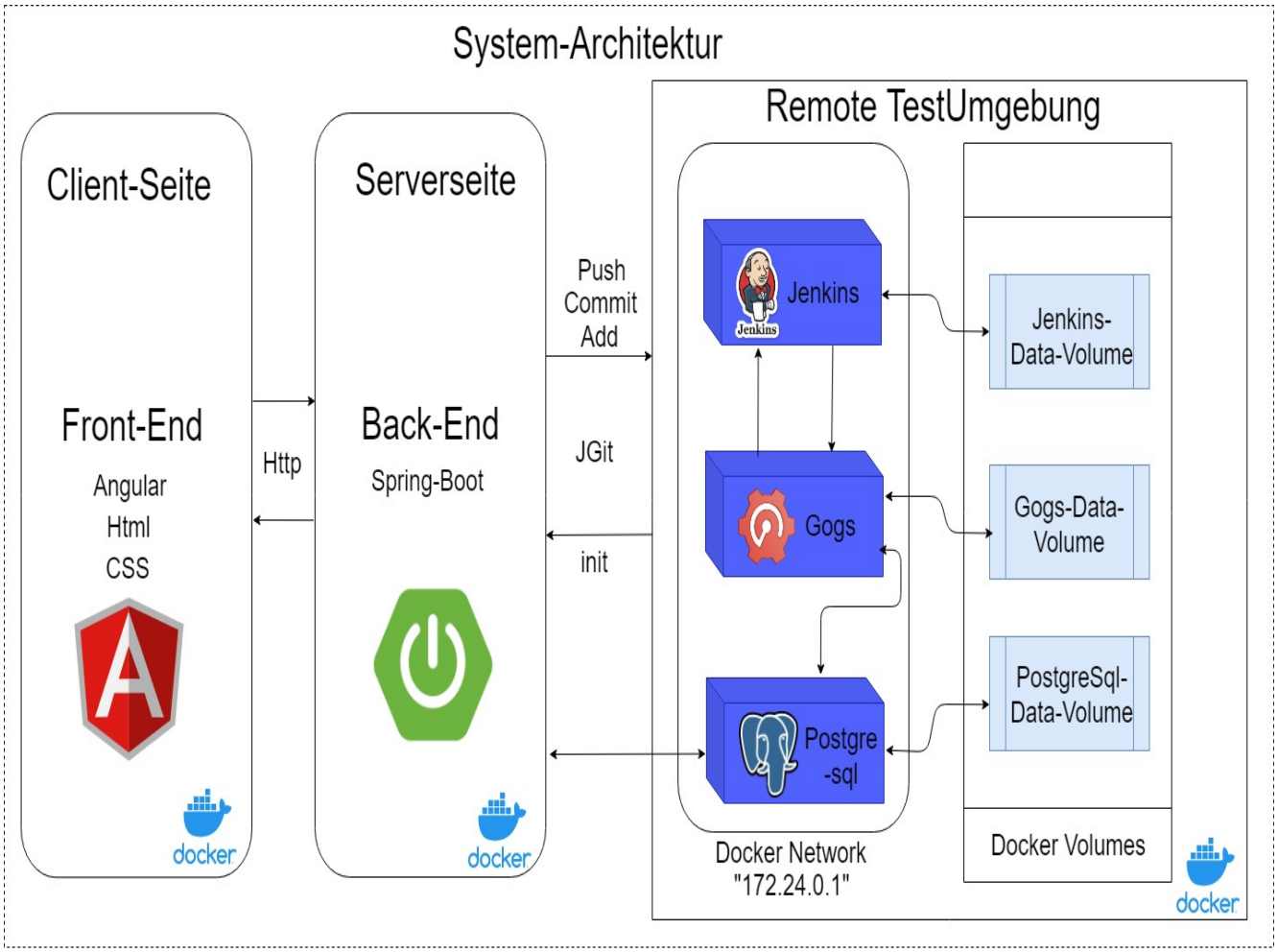

Abbildung 2: Architektur Test-C-Plattform

4.2.1 Client-Seite (Front-End)

In diesem Projekt ist die Benutzeroberfläche in zwei Teile unterteilt: das Administrationsportal und das Studentenportal. Beide Portale wurden mit Angular 7, Html, Css und NGX-Bootstrap erstellt. Über das Administrations Portal kann der Admin Benutzer ein neues Aufgabenblatt erstellen, dies bedeutet, dass ein virtueller Bereich für den Aufgabenblatttest erstellt wird, in dem die Studenten ihre Lösungen für Aufgaben testen können. Der Administrator kann alle Testdateien für jedes Aufgabenblatt in den entsprechenden Testbereich hochladen. Der Administrator kann auch die Testbereiche anzeigen. Jeder Testbereich ist mit dem Namen des zu testenden Aufgabenblatts gekennzeichnet. Für jeden Testbereich werden die spezifischen Testdateien angezeigt und der Administrator kann davon Dateien löschen oder andere hinzufügen. Über das Studentenportal kann der Student die zu testende Lösungsdatei hochladen und dann an den Testbereich senden, der den Test automatisch durchführt und dem Studenten das Testergebnis anzeigt.

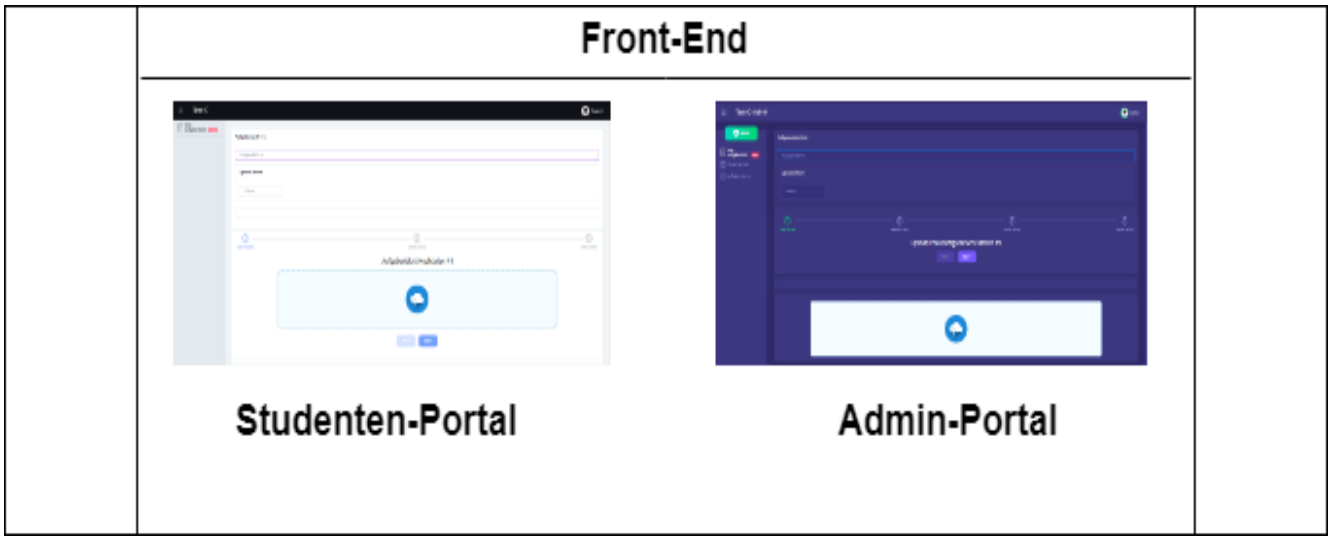

Abbildung 3: Front-End

4.2.2 Remote TestUmgebung

Die Erklärung dieses Teils des Projekts vor der Erklärung des Mittelteils 'Serverseite' ist eine absichtliche Sache, damit der Leser einfach verstehen kann, was im Back-End passieren wird.
Die Remote-Testumgebung besteht aus drei miteinander verbundenen Docker-Containern (Jenkins-Container / Gogs-Container / PostgreSql-Container). Jeder Container verfügt über ein eigenes Datenvolumen zum Speichern der Daten. Dadurch wird vermieden, dass Informationen gelöscht werden, wenn der Container ausgeschaltet oder gelöscht wird.
Um einen Docker-Container in diesem Teil des Projekts zu erstellen, werden zwei verschiedene Methoden verwendet, entweder über eine Dockerfile Datei oder über eine Docker-Compose Datei.
Für den Jenkins-Container wurde eine Dockerfile Datei verwendet. Der Jenkins-Container besteht hauptsächlich aus einem Jenkins, den Compilern: gcc und g ++, die für den Build verantwortlich sind, und doxygen, das für den Testbericht verantwortlich ist.
Die Frage, die sich jetzt stellt, ist: Was ist eine Dockerfile Datei?
Ein Dockerfile ist einfach eine Textdatei mit einer Reihe von Anweisungen (instructions), die genutzt werden können, um ein Docker-Image zu erzeugen.

Listing 1: Dockerfile - jenkins

```
FROM jenkins/jenkins

USER root

RUN apt-get -y update && apt-get -y upgrade

# install gcc g++ gfortran
RUN apt-get -y install build-essential
```

```
# install static analysis
RUN apt-get -y install doxygen graphviz
```

Die FROM-Anweisung legt fest, welches Basisimage zu verwenden ist, wie in diesem Fall Jenkins. RUN-Anweisunggen geben einen Schell-Befehl an, der im Image ausgeführt werden soll. In diesem Fall werden g++, gcc und doxygen ausgeführt.
Für den Gogs-Container und den PostgreSql-Container wurde eine Docker-Compose Datei verwendet. Docker Compose wird zum Definieren und Ausführen von Docker-Anwendungen mit mehreren Containern verwendet. Mit Compose wird eine YAML-Datei (docker-compose.yml) verwendet, um die Dienste (services) von der Anwendung zu konfigurieren, wobei jeder Dienst (service) einen Container mit der zum Erstellen erforderlichen Konfiguration darstellt.

Listing 2: docker-compose.yml - Gogs/PostgreSql

```
version: '2'
services:
app:
image: gogs/gogs:latest
volumes:
- /tmp/gogs-data:/data
ports:
- "3000:3000"
links:
- postgres:postgres
postgres:
image: postgres:alpine
volumes:
- /tmp/gogs-postgres:/var/lib/postgresql/data
environment:
- POSTGRES_USER=postgres
- POSTGRES_PASSWORD=password123
- POSTGRES_DB=gogs
```

Gogs ist ein einfacher selbst-gehosteter Git Service, der einfach einzurichten und zu betreiben ist und auf fast allem ausgeführt werden kann. Es ist zu Hundertprozent Open Source unter der MIT OSS-Lizenz. Gogs bietet die Anzeige und Bearbeitung von Repository-Dateien, die Verfolgung von Projektproblemen und ein eingebautes Wiki für die Projektdokumentation. Gogs benötigt eine Datenbank, um den Quellcode zu speichern, daher wurde hier postgresql ausgewählt.
Nun stellt sich die Frage, wie die Container miteinander kommunizieren ?
Damit Docker-Container über den Host-Computer miteinander und mit der Außenwelt kommunizieren können, wird in diesem Fall eine Netzwerkschicht (Docker Networking) eingesetzt. Docker unterstützt verschiedene Arten von Netzwerken, die jeweils für bestimmte Anwendungsfälle geeignet sind: Bridge mode, Host mode, Container mode, und No networking.

Listing 3: Docker Network

```
[
```

```
{
"Name": "gogs-compose_default",
"Id": "e19d76ee7d9f4132329d44e618bea3c5d93466527dd991d35b01fe4262ba6d73",
"Created": "2019-09-09T17:14:13.0772793Z",
"Scope": "local",
"Driver": "bridge",
"EnableIPv6": false,
"IPAM": {
"Driver": "default",
"Options": null,
"Config": [
{
"Subnet": "172.23.0.0/16",
"Gateway": "172.23.0.1"
}
]
},
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
"Network": ""
},
"ConfigOnly": false,
"Containers": {
"0e69b0fffaf865665b4c2788f0d43a450301dff06ab1d50fedc4d24c0c97f0b4": {
"Name": "jenkins/jenkins",
"EndpointID": "0fc2f145af6d60b8da71e592299d9a0e2d7f578a392832baa0eb4f9e33b1c6e0",
"MacAddress": "02:42:ac:17:00:04",
"IPv4Address": "172.23.0.4/16",
"IPv6Address": ""
},
"264529e624168c9a64a308f2bbadeb26eb3676e54935de87f42675c114721754": {
"Name": "gogs-compose_app_1",
"EndpointID": "3309a74372ac94794dcf1e6f7df21cda2a54c1b98a608e3a5b9922098831d894",
"MacAddress": "02:42:ac:17:00:03",
"IPv4Address": "172.23.0.3/16",
"IPv6Address": ""
},
"6d04174448af71fd4a5de8735f5ad46e52145f933a3c5fbaa3ea53052eccf168": {
"Name": "gogs-compose_postgres_1",
"EndpointID": "e48d315587aa881a2d77984cb856a446d1c5fb74f6fa2f95bb3be6671b90dba7",
"MacAddress": "02:42:ac:17:00:02",
"IPv4Address": "172.23.0.2/16",
"IPv6Address": ""
}
},
"Options": {},
"Labels": {}
```

```
}
]
```

4.2.3 Serverseite (Back-End)

Nun wird über die Serverseite gesprochen, die für die Kommunikation des Benutzers mit der Testumgebung verantwortlich ist.
Die Serverseite stellt viele Dienste wie folgt bereit:

Benutzererkennung In der westfälischen Hochschule Gelsenkirchen basiert die Speicherung der Hauptbenutzeridentitäten auf LDAP-Protokolls (Lightweight Directory Access Protocol) und gespeicherter LDAP-Datenbank. Dies hilft dabei, die Single Sign-On-Technologie (SSO) zu nutzen, die dadurch sich der Benutzer mit derselben ID und denselben Kennwörtern bei allen Anwendungen im selben Netzwerk anmelden kann. Um dies zu erreichen, sind 'spring-boot-starter-security' und 'spring-security-ldap' und 'unboundid-ldapsdk' in diesem Projekt verwendet.

Listing 4: pom.xml - security dependencies

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-ldap</artifactId>
</dependency>
<dependency>
<groupId>com.unboundid</groupId>
<artifactId>unboundid-ldapsdk</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.ldap</groupId>
<artifactId>spring-ldap-core</artifactId>
</dependency>
```

Nach der Konfiguration der Sicherheitsrichtlinie mit reinem Java unter Verwendung der obigen Abhängigkeiten muss der Benutzer seine Studmail-ID und sein Kennwort eingeben, um die Anwendung zu verwenden. Auf diese Weise kann die Anwendung den Benutzer leicht identifizieren und feststellen, ob es sich um einen normalen Student oder einen Administrator handelt.
Unter Verwendung der obigen Abhängigkeiten werden Sicherheitsrichtlinienkonfigurationen wie folgt geschrieben: Zunächst wird festgelegt, welche Dienste der Benutzer ohne Anmeldung bereitstellen kann und bei welchen Diensten er sich anmelden muss. Wenn der Benutzer seinen

Benutzernamen und sein Kennwort eingibt, wird überprüft, ob die Authentifikation-Daten gültig sind, indem sie mit den in der LDAP-Datenbank verfügbaren Daten verglichen werden.
Es wird nun davon ausgegangen, dass der Benutzer gültige Daten eingegeben hat. dh diese Daten sind identisch mit den in der LDAP-Datenbank gespeicherten Daten. Dann wird ein Token generiert und als Antwort auf die Authentifikation-Anfrage zurückgesendet. Dieses Token wird dann vom Frontend verwendet und mit jeder Anfrage gesendet.
Wird eine Anfrage geschickt, wird vom Backend dann überprüft, ob das Token gültig ist. Ist das der Fall, wird dann die Anfrage an den entsprechenden Controller weitergeleitet und damit die passende Funktion ausgeführt.

Erstellung eines neuen Aufgabenblatts Wie oben erklärt wurde, wird beim Erstellen eines neuen Aufgabenblatts ursprünglich ein neuer Testbereich erstellt, an dem die Studenten ihre Aufgabenlösungsdateien testen können. Mit anderen Worten, das Erstellen eines neuen Aufgabenblatts hängt vom Erstellen eines neuen Git-Repository ab, das denselben Namen wie das Aufgabenblatt hat z. B. (Aufgabenblatt-1) über die Gogs-Schnittstelle, die dem Administrator zur Verfügung steht und in der alle zum Testen erforderlichen Dateien gespeichert sind. Anschließend wird ein Jenkins Multibranch-Pipeline-Job erstellt, der auch denselben Namen wie das Aufgabenblatt hat.
Für die Erstellung von Multibranch-Pipeline-Job bietet Jenkins eine REST-API an, die viele Dienste bereitstellt, z. B. das Erstellen und Löschen von Job und viele andere Dinge. Um diese Jenkins-API zu verwenden, muss eine Maven-Abhängigkeit in pom.xml hinzugefügt werden.

Listing 5: pom.xml - Jenkins-Api

```
<dependency>
<groupId>com.offbytwo.jenkins</groupId>
<artifactId>jenkins-client</artifactId>
<version>0.3.7</version>
</dependency>
```

Zur Verdeutlichung wird erläutert, wie mit Jenkins-Api einen neuen Job erstellt werden kann. Um einen Job in Jenkins zu erstellen, müssen dem Job ein Name und eine Reihe von Einstellungen nach Bedarf zugewiesen werden.
Der Jobname wird als Zeichenfolge für die Jenkins-Api-Methode angegeben und die Einstellungen werden jedoch als XML-Datei dargestellt.

Listing 6: JenkinsServiceImp - createJob

```
@Override
public void createJob(String jobName, String jobXml) throws JenkinsException {
try{
JenkinsServer jenkinsServer = new JenkinsServer(new URI(jenkinsUrl), jenkinsUser,
    jenkinsPassword);
jenkinsServer.createJob(jobName, jobXml, true);
}catch (Exception e){
throw new JenkinsException(e);
```

```
}
}
```

C-Datei-Upload Nachdem der Benutzer sein eigenes Konto erstellt und sich bei diesem Konto angemeldet hat, wird die zu testende Datei hochgeladen. Um diesen Service anbieten zu können, habe ich implementiert, was auf der Abbildung 4 gezeigt wird.

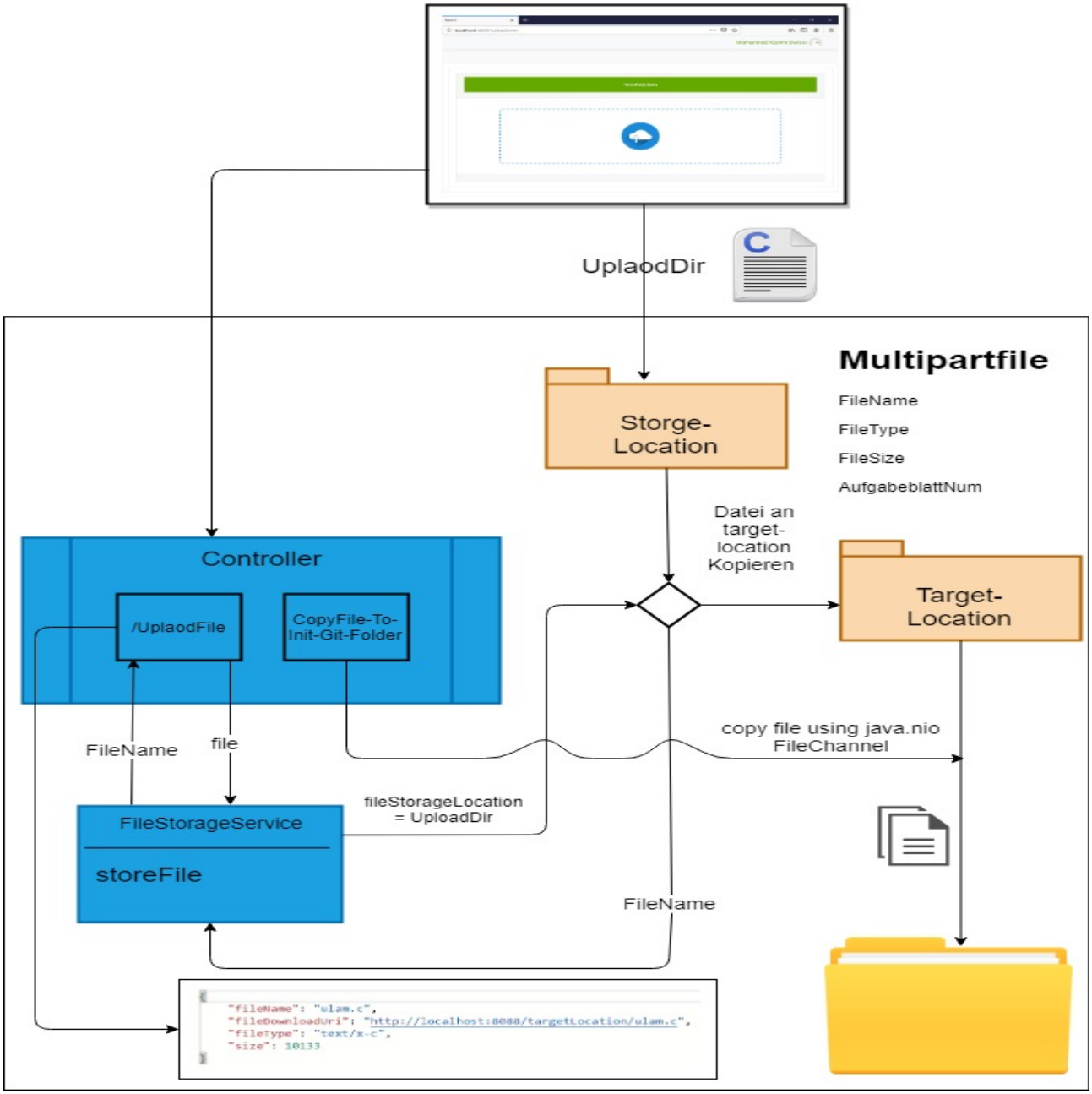

Abbildung 4: Back-End UplaodFile Architektur

Das FrontEnd lädt eine C-Datei an einen temporären Speicherort im Speicher hoch. Darüber hinaus sendet das FrontEnd auch damit einige Informationen wie FileName und Aufgabenblatt-

Nummer, daher empfängt das BackEnd die Datei in Form Multipartfile. Deshalb habe ich eine FileStorageProperties-Klasse erstellt, die als @ConfigurationProperties für die hochgeladene Datei festgelegt wird. Diese Klasse enthält als Attribute das Upload-Verzeichnis von der C-Datei und diese wird danach als fileStorageLocation Path missbraucht.

```
@Autowired
public FileStorageService(FileStorageProperties fileStorageProperties) {
this.fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
.toAbsolutePath().normalize();
}
```

Eine Post-Anfrage wird in dem Controller durch die Methode Uplaodfile generiert. Letzterere verwendet die in FileStorageService implementierte storeFile-Methode, um die Datei als Multipartfile von fileStorageLocation nach targetLocation zu kopieren oder die vorhandene Datei mit demselben Namen zu ersetzen, dafür habe ich verwenden die abstrakte Copy-Methode von java.nio.file.Files.

```
 public String storeFile(MultipartFile file) {
// get File Name
String fileName = StringUtils.cleanPath(file.getOriginalFilename());

try {
// Ueberpruefen, ob der Dateiname ungueltige Zeichen enthaelt
if(fileName.contains("..")) {
throw new FileStorageException("Sorry! Filename contains invalid path sequence " +
    fileName);
}

// Datei an den targetlocation kopieren (Vorhandene Datei mit demselben Namen
    ersetzen)
Path targetLocation = this.fileStorageLocation.resolve(fileName);
Files.copy(file.getInputStream(), targetLocation,
    StandardCopyOption.REPLACE_EXISTING);

return fileName;
} catch (IOException ex) {
throw new FileStorageException("Could not store file " + fileName + ". Please try
    again!", ex);
}
}
```

Die UploadFile-Methode gibt ein UploadFileResponse-Objekt als Antwort zurück. Dieses enthält Informationen über die hochgeladene Datei, wie Dateiname und Aufgabenblattnummer.
Nachdem die zu testende C-Datei mit ihren Informationen als Multipartfile in TargetLocation verfügbar geworden ist, besteht der nächste Schritt darin, diese Datei an ein Initialisiertes Verzeichnis als Git-Repository kopieren.

Abbildungsverzeichnis