Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Bachelorarbeit

Erstellung eines Userinterfaces zum Steuern von CI/CD Prozessen

Vorgelegt von:	Leander Gebhardt
Mat.Nr.:	565964
Datum:	Berlin, 31.10.2021
Fachbereich:	4
Studiengang:	Internationale Medieninformatik
Erstgutachter_in:	Prof. Dr. Barne Kleinen
Zweitgutachter_in:	Dip-Ing. Bahar Jawadi

Inhaltsverzeichnis

1. Einleitung und Motivation

Software Projekte in der heutigen Zeit sind sehr gut geplant und strukturiert. Häufig passiert es dennoch, das am Ende kaum mehr genug Zeit ist, um die Software richtig in der Endumgebung aufzusetzen. Deshalb will ich mit dieser Arbeit ein Nutzerinterface programmieren, welche diese Prozesse übersichtlich anzeigt. Dies soll durch ein ansprechendes Design und verlässliche Funktionen ermöglicht werden. Das Nutzerinterface soll den Entwicklern durch die Automatisierung mit CI/CD Zeit sparen und eine Übersicht geben.

Diese Arbeit wurde ausserdem in Zusammenarbeit mit der Firma Integr8 geschrieben, in welcher ich zum Zeitpunkt des Schreibens dieser Arbeit als Werkstudent angestellt bin. Das Programm, was im Laufe dieser Arbeit entsteht, werde ich außerdem an einem Chatbot Projekt der Firma Integr8 anwenden.

Mir ist durch den komplizierten Deployment Prozess des Projektes aufgefallen, dass es nützlich sein könnte, diese Abläufe zu automatisieren, um sich mehr auf die Entwicklung des Chatbots konzentrieren zu können. Da in der Firma in sehr wenigen Projekten Continuous Integration oder Continuous Delivery verwendet wird, will ich durch diese Arbeit etwas mehr Licht in das Thema CI / CD bringen. Außerdem soll das Tool auch für zukünftige Projekte genutzt werden.

1.1. Problemstellung

Die Hauptfrage soll sein: Wie weit lässt sich der Deployment - Prozess eines CI / CD Tools vereinfachen und grafisch darstellen? Da ich GitHub nutze, um die Action Runner auszuführen, inwieweit ist es möglich eine grafische Oberfläche zu bauen, mit der die Actions übersichtlicher werden. Gerade bei mehreren Projekten gibt es keinen zentralen Ort welcher eine Übersicht dieser Projekte in GitHub bietet.

Weiterhin werde ich in einem bereits vorhanden Projekt Continuous Integration und Continuous Deployment integrieren, und diesen Vorgang mit meinem Tool Visualisieren.

1.2. Lösungsansatz

In dieser Arbeit werde ich ein CI / CD Userinterface erstellen, mit dessen Hilfe man Projekte in GitHub in einer Umgebung deployen und GitHub Actions ausführen kann. Dies werde ich mithilfe von GitHub und GitHub Actions umsetzen, da nahezu alle Projekte der Firma Integr8 auf GitHub hinterlegt sind.

Mit dem Tool, welches im Rahmen dieser Arbeit entsteht, sollte man Programme in einer Heroku Cloud und S3 deployen können.

Nachdem die nötigen Actions in GitHub aufgesetzt sind, kann man nun das Deployment aus der App heraus steuern. Die Anwendung soll vor allem eine Übersicht über die letzten Deploys und Builds bieten.

2. Grundlagen

2.1. Marktanalyse

Da es bereits viele CI / CD Tools gibt habe ich mir in der Grundlagenforschung einige Tools angeschaut. Um erst einmal einen Überblick zu schaffen, habe ich einige Tools miteinander verglichen. (*siehe Tabelle 1*)

Tabelle 1: Überblickstabelle: Continuous Integration Tools auf einen Blick [10]

	Support für CD	Cloud Hosting	Preis für Premiumversion	Kostenlose Version	Besonderheit
Jenkins	✔	✔	-	✔	Sehr viele Plugins
Travis CI		✔	69 - 489 $ pro Monat	✔	Direkte Verbindung zu GitHub
Bamboo	✔	✔	10 – 110.000 $ einmalig	✔	
GitLab CI	✔	✔	4- 99 $ pro Monat	✔	Direkte Verbindung zu anderen Atlassian-Produkten
Circle CI	✔	✔	50 – 3.150 $ pro Monat	✔	Einfache Handhabung
CruiseControl			-	✔	Komplett kostenlos
Codeship	✔	✔	75 – 1.500 $ im Monat	✔	Pro- & Basisversion
Team City	✔		299 – 21.999 € einmalig	✔	Gated Commit

Ich habe mir zwei bereits vorhandene CI / CD Tools näher angeschaut Jenkins und TravisCI, da diese zwei der populärsten Tools sind [10]. Dabei habe ich besonders auf das UI geachtet, welche diese verwenden.

Jenkins, ein Fork der Software Hudson von Oracle (seit 2011 als Jenkins), ist ein Open - Source, webbasiertes Software-System für CI / CD von Software. 2007 war Hudson als eine bessere Alternative, zu dem damals bekannten Cruise Control, ebenfalls eine Open-Source CI / CD Anwendung, bekannt. [15]

Bei dem UI von Jenkins ist mir besonders die Build History und Stage View aufgefallen. In Jenkins veranschaulicht die Build History die letzten Builds mit ihren finalen Resultaten (erfolgreich, fehlgeschlagen) entweder entsprechend rot, wenn der Build fehlgeschlagen ist, oder grün, wenn der Build erfolgreich abgelaufen ist. Außerdem hat Jenkins eine „Wetteransicht“ jedes Projektes. Es werden alle Builds, eines Projektes zusammen betrachtet und je nach Prozentsatz der erfolgreichen Builds ein Wetter-Icon generiert. Je höher der Prozentsatz der erfolgreichen Builds, desto „besser das Wetter“.

Die Wetteransicht gibt also direkt eine grobe Übersicht, in welchem Zustand sich das Projekt befindet und wie die Vergangenheit der Builds aussieht.

Insgesamt ist das Userinterface von Jenkins sehr gut gelungen. Der einzige Punkt der hier zu kritisieren wäre, ist das UI Design, welches etwas veraltet wirkt.

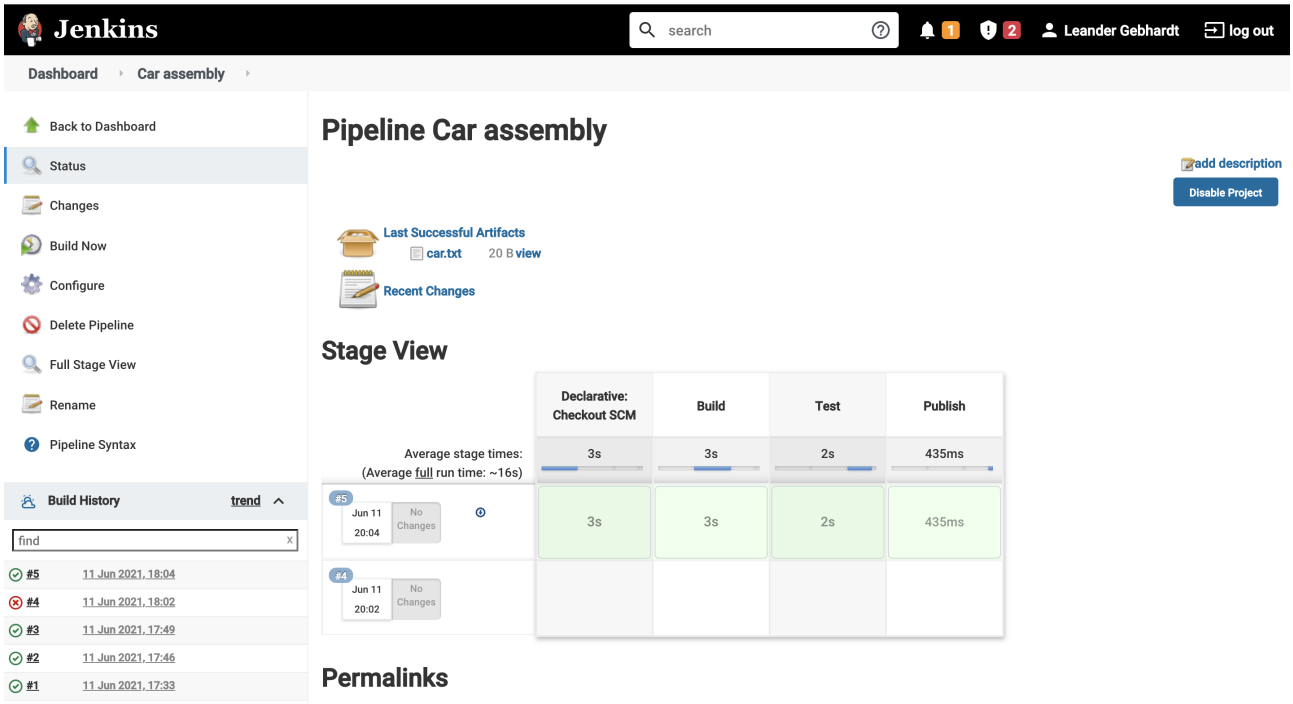

Illustration 1: Jenkins - Build Ansicht

Travis CI ist eine Open - Source – Software für Continuous Integration, welche 2011 in Berlin entwickelt wurde. Um Travis in einem Projekt auf GitHub zu integrieren, wird eine YAML-Datei (*.travis.yml*) mit Konfigurationsparametern im Root - Verzeichnis des Projektes hinterlegt. [16]

Bei Travis CI ist die direkte Anbindung an GitHub recht nützlich, da man dadurch den Import der Projekte extrem einfach hat. Anfangs kann zischen dem Import aller, oder nur ausgewählten Repositories gewählt werden.

Die Benutzeroberfläche ist auch hier sehr minimalistisch gehalten. Das Dashboard zeigt alle aktiven Repositories. Klickt man diese an, wird eine aktuelle Build Ansicht geladen.

Travis CI hebt die Konsole mit den Job - Logs und die config Datei, welche dazu benutzt wird das Build anzufertigen, stark hervor.

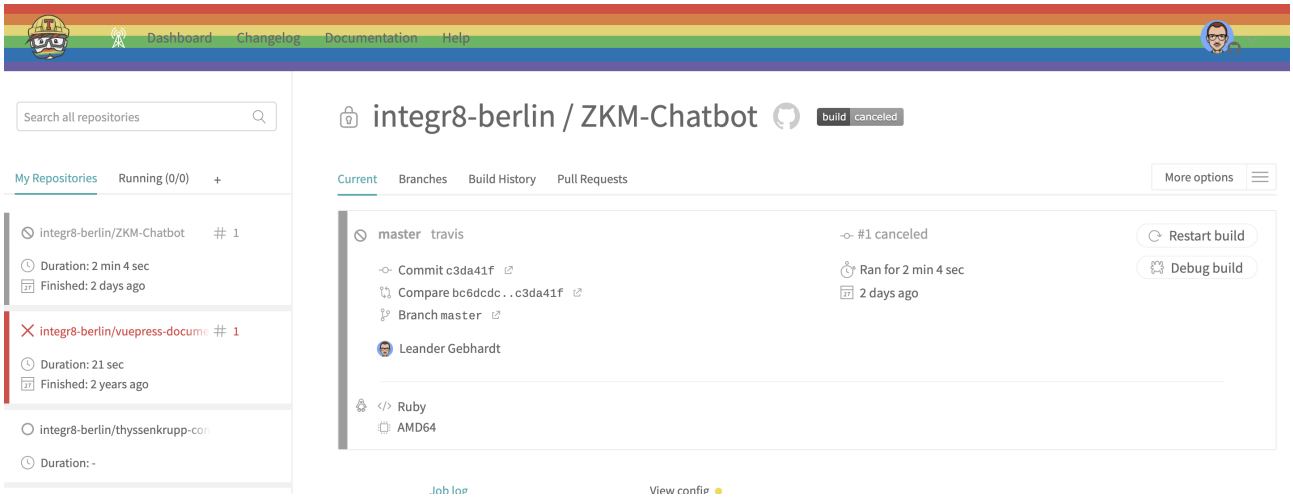

Illustration 2: Travis - CI Projektansicht

Hier fehlt, wie ich finde, eine klare Visualisierung. Wichtige Details, wie Ergebnis des Runs und andere Informationen werden, leicht zu finden, direkt im oberen Bereich dargestellt. Sobald aber nähere Informationen, wie der Run abgelaufen sind, muss die Konsole mit den Job Logs herangezogen werden. Das könnte am Anfang für manche etwas abschreckend wirken. Gibt dem Benutzer jedoch auch detaillierte Informationen zum Ablauf jedes Runs.

Beide Tools, Jenkins und und Travis CI haben etwa eine ähnliche Performance (Build Zeiten). [11]

2.2. Continuous Integration

Continuous Integration ist eine Software Entwicklungspraxis, bei der die Mitglieder eines Teams ihre Arbeit häufig integrieren, mindestens einmal täglich, was zu mehreren Integrationen pro Tag führt. Jede Integration wird von einem automatischen Build und darauf folgenden Tests verifiziert, um Integrationsfehler so schnell wie möglich festzustellen. Dies führt dazu das große Integrationsprobleme vermieden werden können, und Software schneller und sicherer entwickelt werden kann. [2]

2.3. Continuous Delivery / Continuous Deployment

Continuous Delivery ist eine Software Entwicklungstechnik die auf Continuous Integration aufbaut, welche die Auslieferung von Software vereinfachen soll. Dazu wird eine Deployment Pipeline benutzt, welche die Schritte definiert, bevor die Software in der Endumgebung / Development Umgebung oder Test Umgebung zum Einsatz kommt. Auch hier funktioniert die Initialisierung der Pipeline automatisch. [1]

Continuous Deployment geht sogar noch einen Schritt weiter und setzt die fertige Software automatisch in die Produktionsumgebung ein, sodass Benutzer der Software darauf zugreifen können, sobald neue Änderungen in das Source Control System übertragen wurden. Anders als bei Continuous Delivery, bei der noch eine letzte manuelle Überprüfung der Tests nötig ist. [13]

2.3 GitHub Actions

Um mit der CI / CD Anwendung richtig arbeiten zu können, ist ein gutes Verständnis von GitHub Actions unumgänglich. Um die Benutzung der Web Anwendung etwas einfacher zu gestalten, habe ich an entscheidenden Stellen kleine Anweisungen und Erklärungen eingebaut. Diese weisen die Nutzer darauf hin was genau in welcher GitHub Action getan werden muss, damit jede Funktion des Benutzerinterfaces genutzt werden kann.

An dieser Stelle möchte ich etwas ausführlicher auf GitHub Actions eingehen, da ich mich beim Entwerfen und der Implementierung der Anwendung viel damit auseinandergesetzt habe und Actions eine wichtige Grundlage des Userinterfaces sind.

Als erstes ist der Aufbau der Actions wichtig. Im Grunde bestehen GitHub Action Workflows aus einem oder mehreren Jobs. Diese werden von Events, welche direkt auf GitHub sein können oder von außerhalb kommen können, ausgelöst. Die Web Anwendung zum Beispiel löst außerhalb von GitHub bestimmte Actions über die GitHub API aus. Es gibt sehr viele Event – Typen, beispielsweise können Actions, bei einem Pull Request, einem Push, bei Änderungen in bestimmten Verzeichnissen oder nur wenn das Event von einer bestimmten Branche kommt, ausgelöst werden.

Innerhalb dieser Jobs sind Steps, in denen die einzelnen Actions definiert sind. Dieser Aufbau ist in Illustration 3 noch einmal grafisch dargestellt.

Wenn es mehrere Jobs in einem Workflow gibt können diese sequentiell oder parallel ausgeführt werden. Jeder der Jobs bekommt einen eigenen Runner, welcher die Actions in der vordefinierten Reihenfolge ausführt und schreibt die Ergebnisse dieser in einen Action Run - Log.

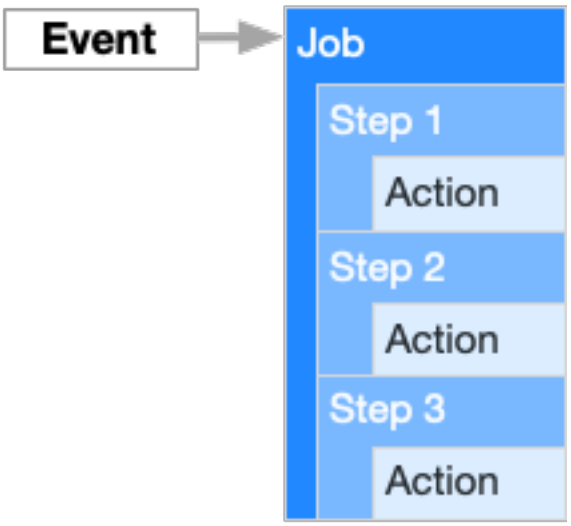

Illustration 3: GitHub Actions Aufbau [6]

Workflows werden auf GitHub in einem extra Ordner abgelegt (`.github/workflows`) und werden im `.YML` Format abgespeichert.

Hier ein kleines Beispiel mit einem Linter Workflow (Siehe Illustration 4) mit dem ich den Code für diese Arbeit linte.

Ein Linter führt statische Code-Analysen durch, welche helfen können, Syntax Fehler sehr schnell zu finden.

Dieser wird bei jedem push in das Repository ausgeführt. Workflows können auch andere, schon erstellte Workflows nutzen mit dem Ausdruck `uses:`. Es gibt bereits viele vorgefertigte Workflows, welche auf einem Marktplatz auf GitHub gesucht werden können.[1] In diesem Beispiel nutze ich den *checkout@v2* und *super-linter@v3* workflow.

Um Shell Befehle ausführen zu können kann das Stichwort `run:` verwendet werden. Im mittleren Step führe ich mehrere Shell Befehle nacheinander aus, indem ich das Pipe Symbol (|) verwende. Wie man in der letzten (Illustration: 4, Zeile: 24) sehen kann, können auch GitHub Geheimnisse (GitHub Secrets) in Workflows verwendet werden. Diese müssen vorher im Repository angelegt werden.

1 GitHub Marktplatz aufrufbar unter: https://github.com/marketplace

```yaml
name: Super-Linter

on: push

jobs:
        super-lint:
        name: Lint code base
        runs-on: ubuntu-latest
        steps:
                - name: Checkout code
                uses: actions/checkout@v2

                - name: Remove files before linting
                run: |
                        cd App/ba-project
                        rm -rf dist
                        rm -rf node_modules

                - name: Lint Code Base
                uses: github/super-linter@v3
                env:
                DEFAULT_BRANCH: main
                VALIDATE_ALL_CODEBASE: false
                GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Illustration 4: Beispielworkflow

Workflow Jobs können auch voneinander abhängen. Zum Beispiel wenn erst getestet werden soll, wenn der Build Job erfolgreich abgeschlossen ist. Dies kann dann folgendermaßen in GitHub Actions Syntax mit den Namen der Jobs ausgedrückt werden:

```
deploy:
    needs: [build, test]
```

Visualisierungen

GitHub bietet eine Graphen Visualisierung für jeden Job in einem Workflow an.

Diese ist nützlich wenn die Beziehungen zwischen den Jobs kompliziert sind. Wenn zum Beispiel mehrere Jobs von einem Anderen Job abhängig sind.

Allerdings können hier nicht die einzelnen Steps eingesehen werden, sondern es muss auf einen Job geklickt werden, welches den Konsolen-Output dieses Jobs anzeigt.

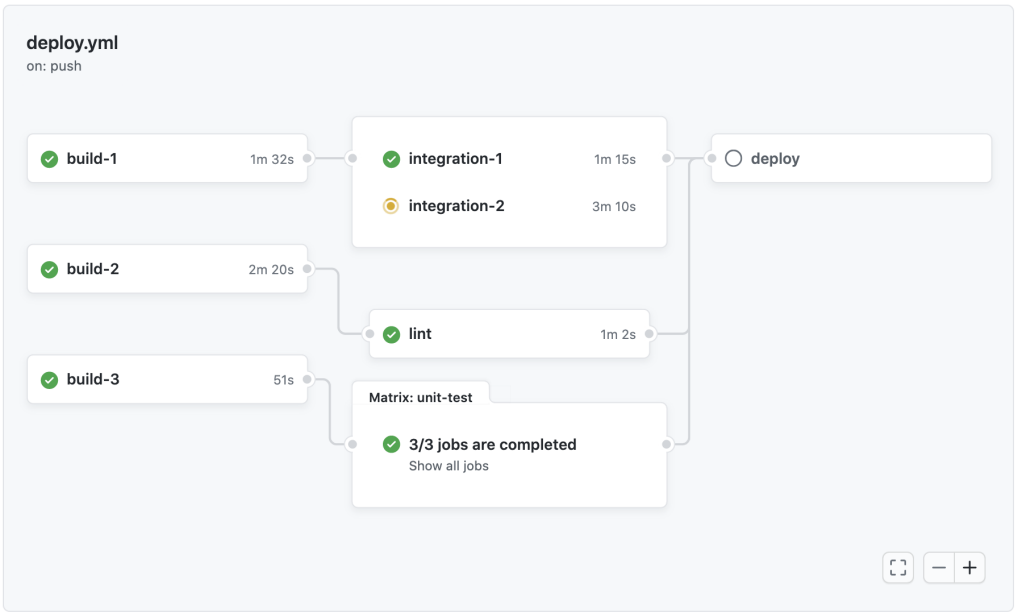

Illustration 5: Graphen Visualisierung GitHub Actions

https://docs.github.com/assets/images/help/images/workflow-graph.png (abgerufen 31.10.2021)

GitHub Runner

GitHub bietet die Option Runner auch lokal auf eigenen Servern zu hosten, da ich in dieser Arbeit lediglich die von GitHub gehosteten Runner benutze, werde ich die Hard- und Software-Spezifikationen dieser Runner kurz beschreiben.

GitHub hostet Linux- und Windows-Runner auf Standard_DS2_v2 Virtuellen Maschinen bei Microsoft Azure mit der darauf installierten GitHub Action Runner - Anwendung.

Außerdem stehen Virtuelle Maschinen mit vorinstalliertem macOS zur Verfügung, welche auf GitHub's eigener macOS Cloud gehostet werden.

Hardwarespezifikation für Windows und Linux virtuelle Maschinen (*Standard_DS2_v2*):

- 2-Kerne CPU

- 7 GB RAM Speicher

- 14 GB SSD Speicherplatz

Hardwarespezifikation für macOS virtuelle Maschinen:

- 3-Kerne CPU

- 14 GB RAM Speicher

- 14 GB SSD Speicherplatz

[14]

Außerdem sendet GitHub bei jedem Workflow Run eine Email an die Person, welche den Workflow als letztes bearbeitet oder reaktiviert hat.
GitHub Actions haben noch sehr viele anderen Features, diese Übersicht soll aber genügen für das, was damit in dieser Arbeit gemacht wird.

3. Entwurf

In den Folgenden Kaptiteln werde ich die Planung vor der Umsetzung des Projektes beschreiben.

3.1. Tech Stack

3.1.1 GitHub / GitHub Actions

Ich habe mich entschieden GitHub als Source Control System zu benutzen, da dieses hauptsächlich für Projekte in der Firma Integr8 benutzt wird. Außerdem bietet es die Möglichkeit, direkt in der Platform eingebaute Actions zu benutzen. Diese ermöglichen es, CI / CD workflows in ein Projekt zu implementieren.

3.1.2 Vue.js

Außerdem werde ich als Frontend Framework Vue.js 2.6.13 benutzen, welches ein modernes Javascript Framework ist, das in vielen aktuellen Benutzeroberflächen Anwendung findet. Dazu bietet sich dieses Framework an, da es sehr gut für die Implementierung von Drittsoftware geeignet ist. Es gibt viele Module, mit welchen sich die Funktionalität, je nach Bedarf erweitern lässt. Darüber hinaus hat es ein sehr gutes

reaktives Verhalten, welches gerade bei einer Übersicht, auf sich immer verändernde Inhalte, sehr wichtig ist.

Es bietet außerdem ein äußerst praktisches Development Tool, dass direkt in der Konsole des Browser ausgewählt werden kann, welches alle Variablen einer Vue Komponenten in Echtzeit anzeigt. Das ist äußerst praktisch zum debuggen einer Anwendung in der Entwicklung.

3.1.3 Express & Node.js

Node.js wurde 2009 veröffentlicht und verhalf damit JavaScript den Aufstieg zur Full-Stack Programmiersprache, welches vorher hauptsächlich für Client - seitige Anwendungen verwendet wurde.

Im Backend werde ich Express, welches zusammen mit Node.js verwendet wird, benutzen. Das erspart mir unterschiedliche Programmiersprachen im Back- und Frontend, da immer JavaScript verwendet wird.

3.1.4 MongoDB & Mongoose

Zum Speichern von Daten werde ich MongoDB Atlas verwenden, welches eine sehr praktische Cloud Funktion hat. Dies macht den Zugriff auf die Datenbank leichter, da dezentral von überall aus auf die Datenbank zugegriffen werden kann. Um mit der Datenbank zu kommunizieren, verwende ich Mongoose, welches das Modellieren von MongoDB - Objekten ermöglicht und vereinfacht, da mit vordefinierten Modellen gearbeitet wird. Das sichert die Integrität der Objekte.

3.1.5 Heroku

Heroku ist ein beliebter Cloud Provider, welcher kostenlos Anwendungen in der Cloud hostet.

Für die Produktionsumgebung habe ich mich für Heroku entschieden, weil es eine praktische Rollback Funktion anbietet. So kann bei missglückten Builds immer zu einer Version gesprungen werden, welche funktionsfähig ist. GitHub bietet auch eine Anbindung an die Heroku API. Außerdem hat Heroku ein CLI (Command Line Interface), welche das Deployment vereinfacht. [5]

3.1.6 Bootstrap Vue

BootstrapVue ist eine Implementation von Bootstrap v4 welche für Vue 1.6 verfügbar ist.

Bootstrap ist ein open – source Frontend Werkzeugkasten, mit dem sehr einfach responsive Webseiten erstellt werden können. Es beinhaltet mehr als 85 Komponenten, 45 plugins und über 1200 Icons, welche nach der Installation überall direkt eingebunden werden können. Dazu haben die Module teilweise eigene Interfaces (in Vue: directives), durch welche diese angesprochen und manipuliert werden können.

Bootstrap Vue ist eine sehr gute Basis, zum Entwickeln ansprechender Web Anwendungen. [12]

3.2. Anforderungen der Anwendung

Bevor ich anfange die eigentliche Web Anwendung zu schreiben, werde ich erst einmal die Anforderungen, die schlussendlich erfüllt werden sollen, definieren.

- Die erste Hauptanforderung ist die Übersichtlichkeit, also eine gute Übersicht über alle Projekte und ihren Status. Das heißt, es sollte erkennbar sein, ob ein Projekt deploybar ist, oder nicht und welche unterschiedlichen Environments für die Projekte existieren. Noch dazu sollte es eine Übersicht geben in welcher Umgebung ein bestimmtes Projekt wann deployed wurde.
- Eine optimierte Übersicht über GitHub Actions Workflows ist nötig.
- Das Deployment der Projekte kann entweder komplett automatisch funktionieren (CI / CD direkt automatisch ausgeführt von GitHub Action) oder aus der Web App gesteuert werden.
- Mit dem CI / CD Tool, welches in dieser Arbeit erstellt wird, sollte das ZKM Chatbot Projekt in einem Environment ausführbar sein.
- Die Benutzerdaten und Daten zu den unterschiedlichen Projekten werden in einer MongoDB Datenbank gespeichert.

3.3. Formulierung des MVPs

Bevor ich begann, die Anwendung zu programmieren, habe ich mir ein MVP (minimal viable product) gesetzt. Diese habe ich nach einer agilen Arbeitsvariante abgearbeitet und implementiert. Wöchentlich wurde der Backlog aufgefüllt und eine zeitliche Reihenfolge festgelegt.

Tabelle 2: MVP Übersicht

Projekte	Environment	Profil	Login	Registrierung	Deploy Methode
Projekt Übersicht (Listenansicht)	Environment Hinzufügen	Profil Übersicht	Möglichkeit zum Login	Möglichkeit zur Registrierung (Erstellung eines Profils)	Deploy Methode hinzufügen
Stage View (Pipelines)	Environment Umbenennen	Benutzer:in löschen	Möglichkeit zum Logout		Deploy Methode löschen
Workflow History	Environment Editieren	Passwort ändern			
Projekte Umbenennen	Environment Löschen				
Projekte Editieren					
Projekte Löschen					
In Environments deploybar					
Zurück zur Hauptansicht Button					

Um dies ohne Papier und überall verfügbar zu haben, verwendete ich das Management und Organisier – Tool *Notion*[2]. Dieses half mir dabei ein digitales und interagierbares Kanban Board zu erstellen und mit diesem zu arbeiten.

Diese hier genannten Funktionen sollten in dem fertigen Projekt funktionieren.

2 Verfügbar unter: https://www.notion.so/

4. Entwurf der Pipeline

4.1. Klassendiagramm

Im Folgenden habe ich ein Klassendiagramm erstellt, um den Zusammenhang der Daten, welche in der Benutzeroberfläche dargestellt werden noch einmal grafisch darzustellen. Im Klassendiagramm zusätzlich dargestellt sind die einzelnen Funktionen der Klassen, welche in der Benutzeroberfläche verfügbar sind.

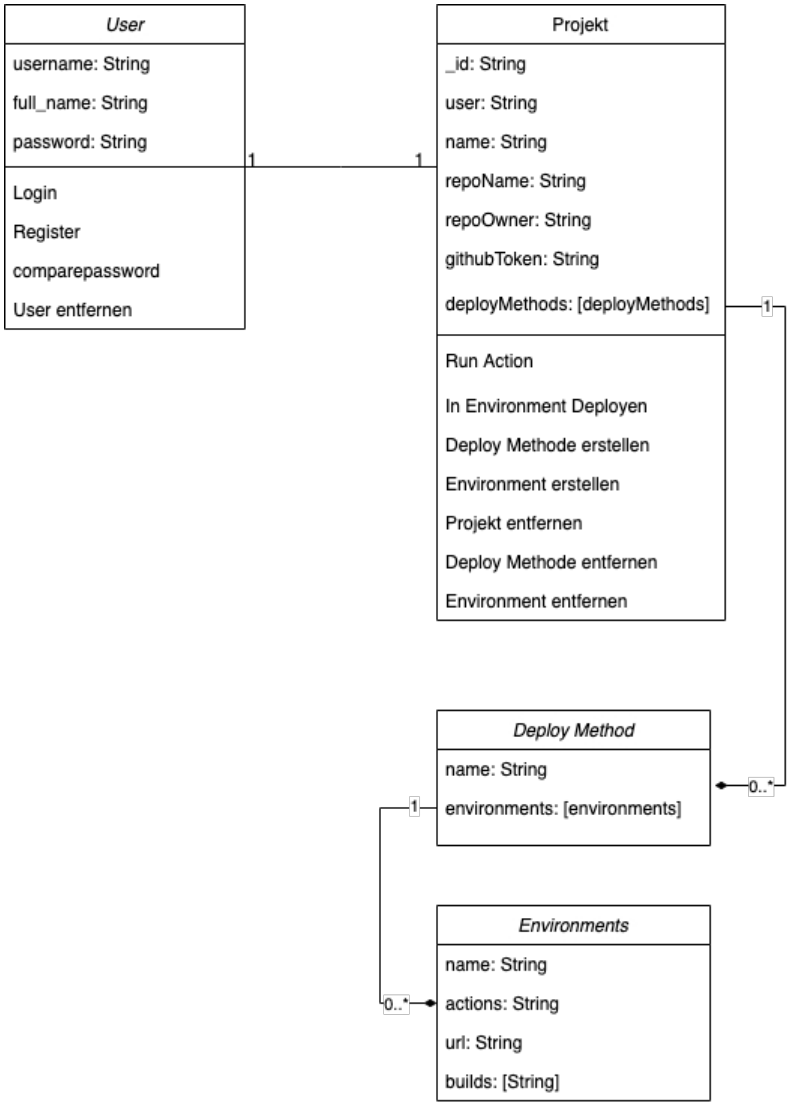

Illustration 6: Klassendiagramm [9]

4.2. Flowcharts

Um eine Beispielhafte Verwendung der Benutzeroberfläche grafisch darzustellen habe ich ein Flowchart erstellt. Dieses zeigt die einzelnen Schritte, die es braucht, um ein Projekt zu in einer Endumgebung zu deployen.

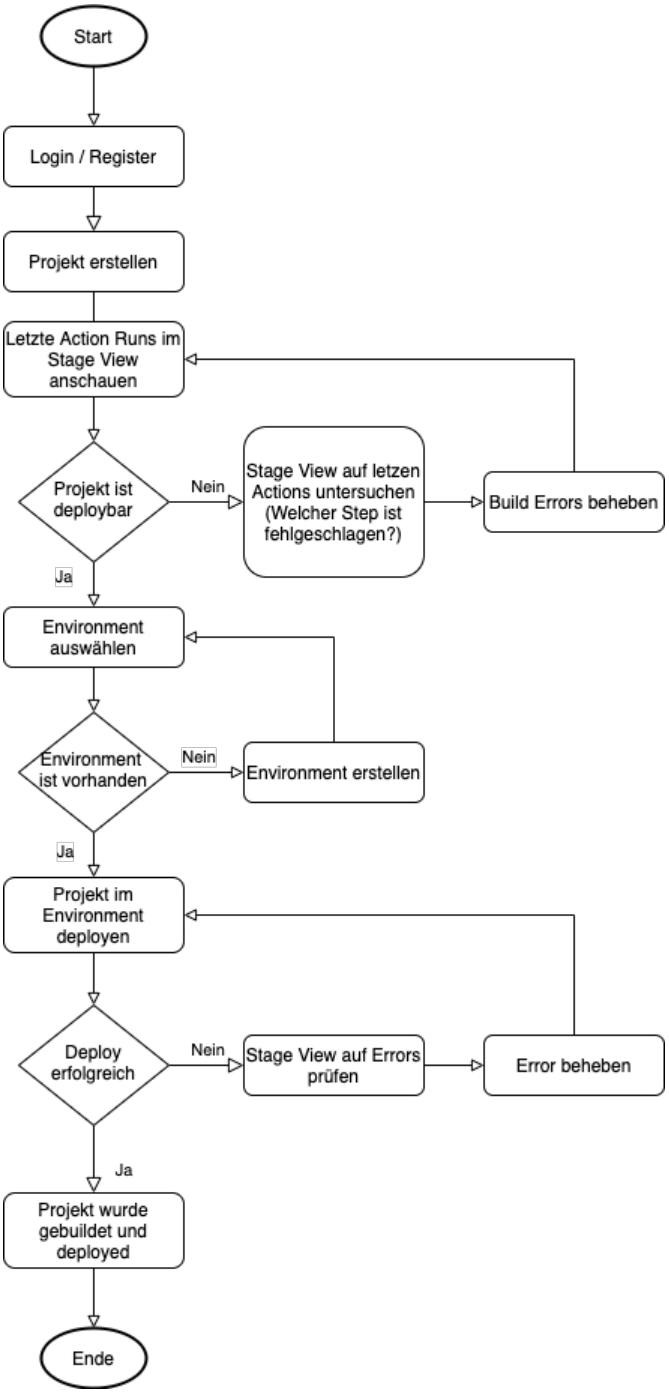

Illustration 7: Flowchart Projekt Deployment

5. Das Projekt indem CI / CD implementiert werden soll

Das Projekt ist ein Chatbot, welcher Gäste eines Museums bei der Führung unterstützen soll. Der Chatbot kann sowohl komplette Führungen präsentieren, als auch auf Fragen der Besucher antworten. Dies macht der Chatbot mit Hilfe eines open source Frameworks, welche durch maschinelles lernen automatisch Text und Stimm-basierte Unterhaltungen generieren kann. Dieses Framework heißt *Rasa*.

Ich will im Folgenden den groben Aufbau des Projektes erläutern.

Es besteht aus einem Frontend, einem Rasa Server und einem Actions Server. Das Frontend ist für die Ausgabe der Unterhaltung zuständig und bietet die Möglichkeit mit verschiedenen Techniken, z.B. eines schiebbaren Bildes, Slidern, mit welchen Zahlen eingestellt werden können und einem QR Scanner, mit dem man Text-QR-Codes scannen kann, die Eingaben der Nutzer zu verarbeiten und an das Backend weiterzuleiten.

Die Hauptaufgabe des Rasa Servers besteht darin, die Absicht des Nutzers zu erkennen. Zum Beispiel wenn der Nutzer den Chatbot fragt: „Wie ist dein Name?“ sollte der Chatbot mit dem Namen, dem die Programmierer Ihm gegeben haben, antworten. Um diese Verbindung zwischen der Frage und der Antwort herzustellen, müssen mehrere Dateien vorhanden sein. Erst einmal brauch man eine *NLU* Datei, in welcher alle Fragemöglichkeiten mit mehreren Beispielen aufgelistet sind. Dank dieser Datei können auch Dinge, wie ein Datum in einem Satz, erkannt und abgespeichert werden.

Die zweite Datei, die sogenannte *Domain* Datei besteht aus allen möglichen Antworten des Chatbots. Diese Antworten können reiner Text sein oder aus Bildern oder extra erstellten Elementen bestehen, wie zum Beispiel ein Timer, einem Schieberegler oder einem Video.

In einer dritten Datei steht der Zusammenhang, d.h. welche Antworten sollen auf welche Fragen folgen. Rasa nennt diese Datei *Stories*.

Bevor der Rasa Server gestartet werden kann, muss eine Model mit Hilfe der oben genannten Dateien (*NLU*, *Domain*, *Stories*) trainiert werden. Dieser Vorgang kann unterschiedlich lange dauern, es kommt darauf an, wie viele unterschiedliche Konversationslinien es gibt.

Dies ist das grundlegende Konzept. Allerdings können auf diese Weise immer nur einfache Antworten gegeben werden, wie mit Text oder Bildern. Sobald aber eine etwas komplexere Aufgabe erfüllt werden soll, kommt der Rasa Action Server hinzu. Rasa erkennt, sobald eine sogenannte Aktion ausgeführt werden soll. Zum Beispiel, wenn der Nutzer nach den Öffnungszeiten fragt, soll eine API angesprochen werden, welche die aktuellen Öffnungszeiten zurückgibt.

Das Ansprechen der API würde in diesem Fall der Rasa Action Server übernehmen.

In Folge der Implementierung einer CI / CD Pipeline soll auch das Training automatisiert und der Action Server auf den Neusten Stand gebracht und anschließend gestartet werden. Github Actions ermöglicht es, dass nach jedem push auch automatisch trainiert und der Action Server aktualisiert werden kann. Ich möchte in der Arbeit diese Pipeline aufsetzen und mein Userinterface damit zur Anwendung kommen lassen.

5.1. Das ZKM

Das ZKM (Zentrum für Kunst und Medien) befindet sich in Karlsruhe, auf der Lorenzstraße 19. Es ist eine Kulturinstitution der raumbasierten Künste wie Malerei, Fotografie und Skulptur als auch der Zeitbasierten Künste wie Film, Video, Medienkunst, Tanz, Theater, und Performance. Die ursprüngliche Aufgabe des ZKMs ist es, klassische Künste ins digital Zeitalter fortzuschreiben.[3] Das ZKM hat Integr8 2020 damit beauftragt einen Chatbot, der sowohl beim Service auf der Website, als auch bei Führungen der Benutzer durch die Unterschiedlichen Werke unterstützt, zu entwickeln.

6. Implementierung

In den Folgenden Kapiteln werde ich die Umsetzung des beschriebenen Entwurfs näher beschreiben. In den Kapiteln wird es um die Erstellung des Userinterfaces zum Steuern von CI / CD Prozessen gehen.

3 https://zkm.de/de/das-zkm

6.1. Frontend (Vue App)

Beim Erstellen des Frontends war es mir besonders wichtig, dass es möglichst übersichtlich und einfach bleibt. Ich habe einen minimalistischen Ansatz gewählt. Bei diesem werden die Benutzer Schritt für Schritt bis zum Deployment geführt.

Ich habe darauf geachtet, dass einige einfach Regeln eingehalten werden. Überall, wo etwas angelegt werden kann, ist es möglich, dieses am selben Ort wieder zu entfernen, an dem man es erstellt hat.

Dafür war die Visualisierung der Daten in einem Projekt wichtig. Hier habe ich den Aufbau der Visualisierung in der Anwendung grafisch dargestellt.

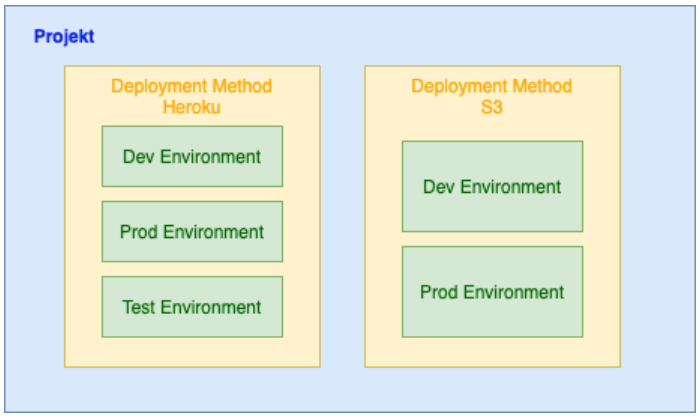

Illustration 8: Data Visualisierung

Nach diesem Aufbau wird die nutzende Person durch die Oberfläche geleitet. Beispielsweise kann in der Hauptansicht anfangs ein Projekt ausgewählt werden. Anschließend sind in der nächsten Ansicht die verschiedenen Deployment Methoden wählbar. So bauen die Ansichten aufeinander auf und sind untereinander abgeschlossen.

In Illustration 7 habe ich zwei unterschiedliche Deployment Methoden (hier in gelb dargestellt) veranschaulicht (Heroku & S3).

Hier ein Prototyp wie es ungefähr im fertigen Programm aussehen könnte.

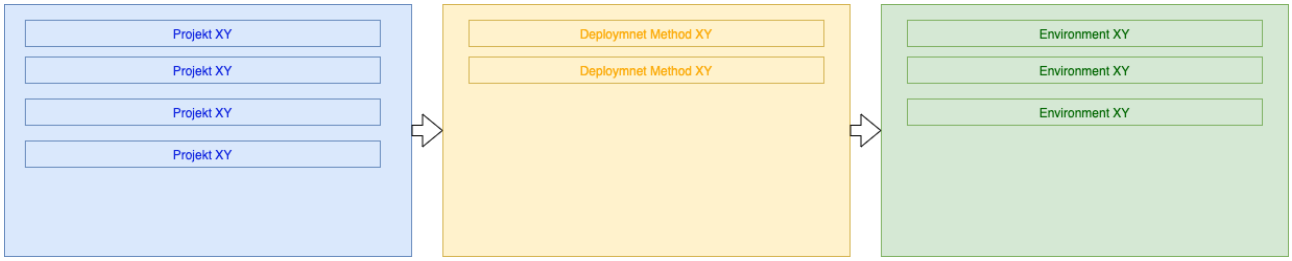

Illustration 9: Ansichten Prototyp

Um diese Ansichten in Vue umzusetzen, werden dafür vorerst Komponenten erstellt. In diesem Fall habe ich die Komponenten *Projekt*, *Deployment Methode* und *Environment* in Vue angelegt.

Kurzer Vue Exkurs:

Im Folgenden möchte ich zum Verständnis des Codes einen kurzen Vue Exkurs machen und einige Grundlagen erklären.

Eine Vue Datei besteht aus mehreren Abschnitten. Einem *Template* Abschnitt, welcher den HTML Teil der Komponenten darstellt.

Ein weiterer Abschnitt in einer Typischen Vue Komponente ist der *Script* Abschnitt, in welchem sich die Logik der Komponente (in javascript geschrieben) befindet.

Der letzte Teil ist der *Style* Abschnitt. In diesem wird der CSS Code der Komponente festgelegt.

Ich stellte relativ schnell fest, das noch eine entscheidende Ansicht fehlte. Es müsste eine Übersicht geben, die den Status des Projekts ersichtlich macht. Folgende Informationen sollen in der Übersicht dargestellt werden: Welche GitHub Actions gibt es im ausgewählten Projekt und dem verbundenen GitHub Repository? Wie waren die Ergebnisse der letzten Workflow Runs (erfolgreich, fehlgeschlagen). Welche einzelnen Jobs gibt es in diesen Actions?

Deshalb habe ich die Projekt Übersichts – Komponente erstellt, welche die oben genannten Dinge darstellt. Hier habe ich auch die Inspiration aus Jenkins zu Hilfe genommen und eine Art Build History erstellt, nur das ses in meinem Fall GitHub Actions sind und keine direkt in meinem Tool zusammengebauten Pipelines. Diese Ansicht fehlt

bei GitHub auch, da nur in der Konsole in Steps unterteilt wird. Jetzt findet eine Visualisierung der Steps statt, das schafft Übersicht.

Außerdem habe ich eine Node View eingeführt, um die einzelnen Steps der unterschiedlichen Actions besser darstellen zu können. Jede Node in dieser Ansicht repräsentiert einen Schritt in einer Action. Je nach dem wie diese laufen werden diese in Grün oder Rot dargestellt, entsprechend des Endresultates nach dem Durchlaufen der Action (erfolgreich, fehlgeschlagen), ähnlich wie bei den einzelnen Actions.

Diese Ansicht gibt eine gute Übersicht über den Verlauf und das Endergebnis des Action Runs. So kann das Projekt dementsprechend in einer erstellten Umgebung deployed, oder überwacht werden.

Falls bestimmte Actions nicht automatisch ausgelöst werden sollen, sondern manuell, kann das von der Benutzeroberfläche getan werden. Unter *„Run Actions“* werden alle Actions, die in diesem Repository existieren, aufgelistet und können von dort aus ausgeführt werden. (*Man beachte die Information zum richtigen Aufsetzen des Workflows*)

6.1.1. Arbeit mit der GitHubs Action API

Um aus der Anwendung heraus GitHub Actions zu starten, habe ich die GitHub API genutzt. Diese bietet die Möglichkeit, mittels eines HTTP Request, eine GitHub Action in einem Repository zu starten. Der Request Pfad sieht folgendermaßen aus.

```
POST https://api.github.com/repos/${owner}/${repo}/dispatches
```

Im Request Body muss ein Event Typ mitgeliefert werden und ein Basic Authorization Header mit dem GitHub Benutzername und Access Token als Passwort sind notwendig. Dieser ist der Name des Typen welcher in der `workflow.yml` Datei auf GitHub definiert sein muss. Näheres zu GitHub Actions und wie diese aufgebaut sind im Kapitel 2.3 *GitHub Actions*. [4]

6.1.2. Pipeline-Darstellung Implementierung

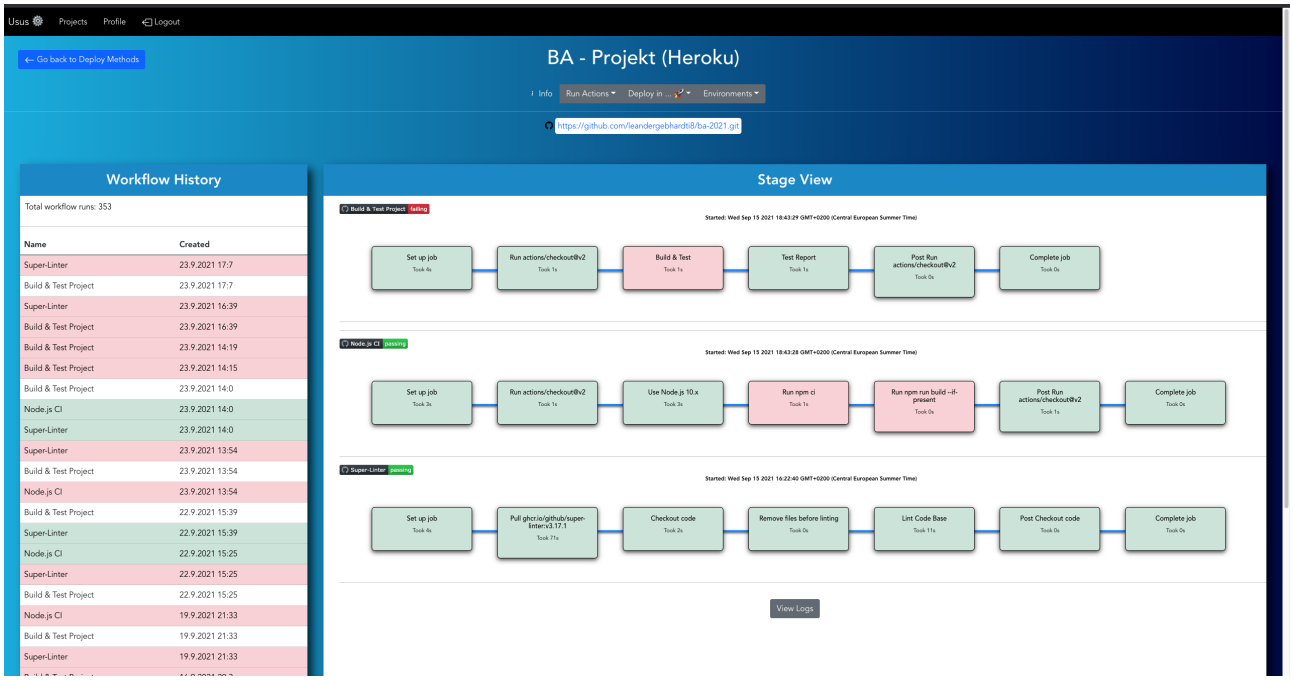

Illustration 10: Projekt - Ansicht

Eine weitere Herausforderung war die Gestaltung einer Pipeline Ansicht.
Auf der linken Seite der Projekt – Ansicht, ist die Workflow History zu sehen. Diese zeigt alle Action Workflows an, welche im GitHub Repository ausgeführt wurden. Entsprechend des Endergebnisses, werden diese grün (erfolgreich), oder rot (fehlgeschlagen) eingefärbt. Neben der Workflow History ist in der Etappenansicht (Stage View) eine Pipeline, Visualisierung der einzelnen Workflow – Jobs des aktuellsten Runs dargestellt. Die einzelnen Bestandteile der Pipeline machen die Schritte in einem Workflow Job aus.

Besitzt ein Workflow mehrere Jobs, sind diese auch einzeln in ihren Schritten unterteilt aufgelistet.

Dabei gibt es auch ein bestimmtes Farbschema, welches sowohl bei den einzelnen Steps, als auch bei den Jobs zutrifft. Bei Jobs ist allerdings ein beschriftetes Symbol neben dem Job zu sehen, wobei bei einem Schritt die Hintergrundfarbe den Status des Abschlusses bestimmt.

- **Grün**: Step/Job war erfolgreich

- **Rot**: Step/Job ist fehlgeschlagen

- **Grau**: Step/Job wurde übersprungen

- **Weiß**: Step wurden noch nicht ausgeführt

6.1.3. Error Handhabung

In der Benutzeroberfläche gibt es viele Anwendungsmöglichkeiten für Error Handhabung. Die wichtigsten davon sind, die Fälle, in denen es um das Anzeigen von Informationen, welche von der GitHub API kommen geht.

Die GitHub API benötigt zur Authentifizierung und zur Zuordnung immer den Namen des Repositories, den Repository - Inhaber und dessen persönlichen Access Token, der zuvor erstellt werden muss. Sobald die Benutzer eines dieser Informationen falsch angeben, wird es zu Fehlern kommen und das Projekt mit den dazugehörigen Actions wird nicht richtig angezeigt werden können.

Dafür müssen für die jedes Szenario eine Fehlerbenachrichtigung angezeigt werden, um die Herkunft des Fehlers möglichst präzise einzukreisen.

In der Testphase der Anwendung sind zwei Fehler-Szenarien besonders oft aufgetreten. Das erste Szenario, das vorher bereits erwähnt wurde, ist die fehlerhafte Eingabe der GitHub Informationen.

Das zweite Fehler-Szenario ist das Ablaufen des GitHub Tokens. Da die GitHub Access Tokens mit einem „Verfallsdatum“ versehen werden können, kann es demnach zu Problemen kommen, an die Informationen von GitHub zu kommen wenn der zur Verfügung gestellte Token nicht mehr gültig ist.

6.1.4. Einzelne Actions manuell aus dem Tool starten

In der Projektansicht des Tools, gibt es eine Toolbar mit unterschiedlichen Dropdown Menüs. Das erste Menü, bietet eine Auswahl von allen Workflows, welche in dem GitHub Repository existieren. Beim Auswählen einer dieser Workflows wird dieses, soweit es richtig in GitHub aufgesetzt ist, gestartet. Direkt neben dem Menü befindet sich ein Informationsfeld, welches das richtige Aufsetzen des Workflows in GitHub beschreibt.

Dies kann nützlich sein, wenn zum Beispiel manuell ein Test wiederholt werden soll, oder um spezielle Workflows manuell auszuführen, welche nicht automatisiert ablaufen sollen.

6.1.5. Ein Projekt in einem Environment deployen

Ebenfalls in der Toolbar der Projektansicht, ist das Menü zum deployen eines Projektes in eine Umgebung zu finden (2. Dropdown Menü von links). Dieses Menü bietet die Möglichkeit ein Environment auszuwählen, welches vorher mit dem Tool in diesem Projekt erstellt werden muss. Ist das Environment angelegt, ist es in diesem Menü wählbar. In einer Deploy Methode können mehrere Environments erstellt werden. So kann z.B. eine häufig verwendete Auswahl an unterschiedlichen Environments angelegt werden, in welche je nach Development Phase deployed werden kann.

Als Beispiel wären diese: Eine Development Umgebung, Test Umgebung und eine Produktions Umgebung. Ist ein neues Feature in der Development Umgebung deployed und getestet worden, kann das Projekt von hier aus in die Test Umgebung deployed werden und von Testern manuellen Tests unterzogen werden.

6.2. Frontend Ansichten

In diesem Kapitel möchte ich jede Ansicht in dem Projekt kurz beschreiben und deren Nutzen erläutern.

6.2.1. Register / Login

Das Erste, was die Benutzer in der Webanwendung tun ist, sich zu registrieren. Dies kann in der Registrier - Ansicht gemacht werden, mit der Angabe eines Benutzernamens, des vollen Namens und das Anlegen eines Passworts. Dieses Passwort wird noch einmal abgefragt, um sicher zu gehen, dass die Benutzer sich nicht bei der ersten Eingabe des Passworts vertippt haben. Stimmen diese beiden Passwörter überein so kann das Konto erstellt werden. Nachdem sich die Benutzer erfolgreich registriert haben, werden sie sofort angemeldet.

Wenn bereits ein Benutzeraccount erstellt wurde, kann sich damit in der Login Ansicht angemeldet werden.

6.2.2. Home

Die nächste Ansicht, zu welcher man direkt nach der Anmeldung gelangt, ist die Home Ansicht. Hier wird eine Übersicht dargestellt über alle Projekte der Benutzer, der in der Vergangenheit bei Deployments erstellen Builds.

Dazu sind alle zum Deployment wichtigen Informationen, wie z.B. der Zeitpunk des Builds, in welcher Umgebung deployed wurde (Dev, Test, Prod), der Name der Deploy Methode (S3, Heroku) und der Name des Projektes, hier dargestellt.

Es gibt in dieser Ansicht die Möglichkeit direkt zu der Projektübersicht zu gelangen, ohne die Navigation im Header der Seite zu verwenden. So kommt man schneller zur nächsten hilfreichen Ansicht.

6.2.3. Projektübersicht

Die Projektübersicht ist eine Listenansicht aller Projekte, die die nutzende Person angelegt hat. Diese repräsentieren ein GitHub Repository mit integrierten GitHub Actions.

Oberhalb der Listenansicht ist ein Suchfeld, mit welchem Projekte gefiltert werden können. Dies erleichtert das Finden eines bestimmten Projektes, wenn bereits viele Projekte angelegt wurden.

In dieser Ansicht können neue Projekte über die Schaltfläche oberhalb der Listenansicht angelegt werden. Dazu werden Informationen zum GitHub Repository und ein GitHub Access Token der nutzenden Person benötigt.

Außerdem können hier bereits erstellte Projekte umbenannt, editiert, oder gelöscht werden.

Durch die Schaltfläche links oben kann auch zur Home Ansicht gewechselt werden.

6.2.4. Deploy Methoden

Nachdem ein Projekt ausgewählt wurde, kann nun festgelegt werden in welcher Umgebung das Projekt später aufgesetzt werden soll (Heroku, S3).

Ist eine Deploy Methode ausgewählt, kann diese in der Listenansicht aufgesucht werden.

Hier können auch beide Deploy Methoden ausgewählt werden, wenn das Projekt sowohl in Heroku, als auch in der Amazon S3 Cloud aufgesetzt werden soll.

Dies dient zur Einordnung. Hier werden keine GitHub Actions automatisch ausgeführt. Diese müssen später noch erstellt werden, falls diese nicht schon im Repository existieren.

Oben links gibt es wieder eine Schaltfläche, welche die Benutzer zurück zur Projektübersicht führt.

6.2.5. Projekt Ansicht

Diese Ansicht ist die zentrale Komponente in der Webanwendung, denn hier werden die wichtigsten Information angezeigt.

Als erstes wird der Name des Projekts eingeblendet. Darunter ist eine Toolbar mit mehreren Dropdown Menüs. Im ersten Dropdown Menü können GitHub Action Workflows manuell ausgeführt werden. Im zweiten Menü kann das Projekt in eine zuvor angelegte Umgebung deployed werden. Im dritten und letzten Menü, können Environments ausgewählt und neue erstellt werden.

Unter der Toolbar ist der Link zum zugehörigen GitHub Repository angegeben.

Der übrige Teil des Benutzerinterfaces ist in zwei Teile aufgeteilt. Auf der linken Seite die Workflow History, welche anzeigt wie oft insgesamt workflows in diesem Repository ausgeführt wurden. Die letzten 30 Workflow Runs werden darunter in einer Listenansicht angezeigt. Dazu sind die Workflow Namen eingeblendet und wann diese gestartet wurden.

Auf der rechten Seite, in der „Stage View“ werden die einzelnen workflows im Detail angezeigt. Jeder Workflow wird erst in Jobs unterteilt. Bei jedem Job wird eine Pipeline – Ansicht mit den einzelnen Steps dargestellt. Nähres zu dieser Ansicht habe ich bereits im

Kapitel **6.1.2. Pipeline-Darstellung Implementierung** beschrieben und möchte deshalb hier nicht weiter darauf eingehen.

6.2.6. Environment Ansicht

In der Toolbar in der Projekt Ansicht, kann im rechten Dropdown Menü ein Environment ausgewählt werden. Dies bringt die Benutzer zu der Environment Ansicht.

In dieser Ansicht kann die Umgebung editiert, umbenannt oder gelöscht werden. Darunter ist eine Deployment History dargestellt. Hier ist der Zeitpunkt des letzten Deploys und eine Liste von Zeitstempeln aller Deploys.

6.2.7. Profil Ansicht

Sobald die Benutzer angemeldet sind ist in der Navigationsbar am oberen Bildschirmrand der Webanwendung die Profil Ansicht wählbar. Diese zeigt alle Benutzerinformationen an. Hier kann das Passwort geändert werden oder das Benutzerkonto gelöscht werden.

6.3. Backend

Um mit dem Backend einfacher kommunizieren zu können, habe ich eine REST API eingerichtet. Um diese ausführlich zu dokumentieren habe ich mich dazu entschieden, mithilfe von API Dokumentationstool Swagger [7] , eine Dokumentation in der Web Anwendung anzuzeigen. Diese kann unter dem folgenden Pfad aufgerufen werden (`/docs`).

Das Backend kommuniziert mit der Datenbank, mit einem CRUD Interface, welches serverseitig implementiert werden musste.

6.3.1. User Management

Die wichtigsten Bestandteile des Backends sind die mongoose Models und die Definition der API. Dabei werden Projekte, wie auch Benutzer verwaltet.

Ein Benutzer besteht aus dem folgenden Mongoose Schema.

```
1       /* User Schema */
2       const userSchema = new mongoose.Schema({
3               full_name: {
4                       type: String,
5                       trim: true,
6               },
7               username: {
8                       type: String,
9                       unique: true,
10                      required: true,
11              },
12              password: {
13                      type: String,
14                      unique: false,
15                      require: true,
16              },
17      });
```

Illustration 11: Benutzer Mongoose Schema

6.3.2. User Authentifizierung

Um das Passwort in der Datenbank zu speichern, wird dieses vorher erst einmal mit einer Hash-Funktion umgewandelt. In meinem Fall habe ich mich für bcrypt[4] entschieden, welches eine weit verbreitete Hashfunktion ist, die noch dazu gut anpassbar ist. Mit einem einstellbaren Kostenfaktor kann der Arbeitsaufwand der beim hashen des Passwortes nötig ist [8] definiert werden. So wird bei jedem Login das gehashte Passwort mit dem vom Benutzer eingegebenen Passwort verglichen. Das hashen hat den Vorteil das in der Datenbank keine Klartext-Passwörter abgespeichert werden, und deshalb auch nicht in die Hände Außenstehender fallen können.

6.3.3. Project Management

Jedes Projekt wird auch nach einem bestimmten „Bauplan“ erstellt, nämlich dem hier dargestellten Object Struktur (Schema).

Jedes Projekt wird mit einem user gespeichert, welcher den/die Benutzer:in darstellt, welche/r das Projekt abgespeichert hat. So werden allen Nutzern nur die Projekte angezeigt, welche sie auch erstellt haben.

4 Verfügbar auf: https://www.npmjs.com/package/bcrypt

```
1       /* Environment */
2       const environmentSchema = new mongoose.Schema({
3               name: String,
4               action: String,
5               url: String,
6               builds: [String],
7       });
8
9       /* deployMethod */
10      const deployMethod = new mongoose.Schema({
11              name: String,
12              environments: [environmentSchema],
13      });
14
15      /* Project */
16      const projectSchema = new mongoose.Schema({
17              projectId: {
18                      type: String,
19                      index: true
20              },
21              user: {
22                      type: String,
23                      unique: true,
24              },
25              name: String,
26              repoName: String,
27              repoOwner: String,
28              githubToken: String,
29              deployMethods: [deployMethod],
30      });
```

Illustration 12: Mongoose Schema für Projekte

6.4. UX / UI Design

UX Design definiert einen großen Teil (mit Inbegriffen UI Design) des gesamten Design Prozesses einer, in meinem Fall, Web Anwendung.

Beim UX Design habe ich besonders darauf geachtet, das alle Elemente, mit denen interagiert werden kann, möglichst schnell zu finden und klar angeordnet sind. Das habe ich durch eine immer gleiche Positionierung der Elemente erreicht. So lernen die Benutzer des Interfaces einmal die Funktion eines Elementes und können diese auf anderen anwenden, und wissen was passieren wird, wenn diese betätigt werden.

Das Hauptziel des Designs ist es die Übersichten möglichst simpel zu halten und nur die gerade nötigen und wichtigsten Information zu zeigen.

UI Design durch die Firma Integr8: Design-Team[5]

5 Hanna Biarozka h.biarozka@integr8.com

6.5. CI / CD Implementierung in das Projekt

Wie in Kapitel 5. bereits beschrieben, handelt es sich bei diesem Projekt um einen Rasa Chatbot. Mit Hilfe von GitHub Actions soll nun CI / CD in das Projekt implementiert werden.

Den Groben Entwurf der Deploy Pipeline habe ich bereits in Kapitel 5 beschrieben, deswegen werde ich hier nur eine Skizze einführen, welche die Pipeline Visualisiert.

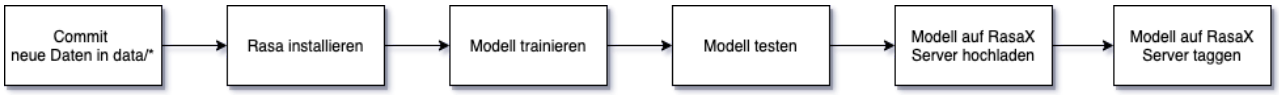

Illustration 13: CI Pipeline

Der erste Schritt der Pipeline ist das Prüfen, ob sich neue Daten in irgendeinem Unterverzeichnis von `data` nach einem Commit befinden. Wie in *Kapitel 5* beschrieben ist es nur notwendig ein neues Modell zu trainieren, wenn sich in diesem Unterverzeichnis Daten verändert haben. Ist dies der Fall, wird anschließend Rasa (und alle anderen Pakete die gebraucht werden, *siehe requirements.txt*) in der virtuellen Umgebung installiert und ein Modell trainiert. Daraufhin wird das trainierte Modell getestet, und falls diese Tests alle erfolgreich waren, soll das Modell auf dem RasaX Server hochgeladen werden.

Diese Tests werden in `conversation_tests.yml` definiert. Hier habe ich einige Werke beschrieben, welche beim testen des Chatbots durchgegangen werden.

Als letztes soll das soeben hochgeladene Modell noch als Produktionsmodell deklariert werden, damit der RasaX Server das Modell, für den in Produktion laufenden Chatbot, benutzt.

RasaX ist eine zusätzliches Werkzeug, zu der auf dem Server laufenden Rasa Open Source Anwendung. Es hat ein hübsches UI, um den Rasa Assistenten besser konfigurieren, und die letzten Unterhaltungen des Chatbots überwachen und analysieren zu können. (Siehe RasaX UI Screenshot Illustration: 13)

Den letzten Schritt habe ich getestet. Er ist aber nicht Teil des finalen Workflows, da das ZKM Karlsruhe selber Modelle auf den RasaX Server hochladen will. Mit dem letzten

Schritt würde ich diese Modelle immer überschreiben, deswegen ist dieser Schritt außen vor.

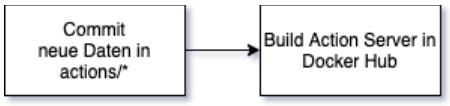

Illustration 14: Action Server Pipeline

Mein Ziel war CI und CD in zwei unterschiedliche Workflows aufzuteilen. Da es sich aber anbietet, das Modell im gleichen Workflow in dem es trainiert wird auch hochzuladen, ist hier die Unterscheidung zwischen CI und CD nicht mehr ganz so einfach.

Im ersten Workflow (*ich habe ihn CI genannt*) werden alle Aufgaben die das Modell betreffen, erledigt. Im zweiten Teil soll jetzt der Action Server deployed werden. Dazu habe ich einen neuen Workflow erstellt. Dieser Workflow wird immer dann aktiviert, wenn Änderungen in das `actions` Verzeichnis gepusht werden.

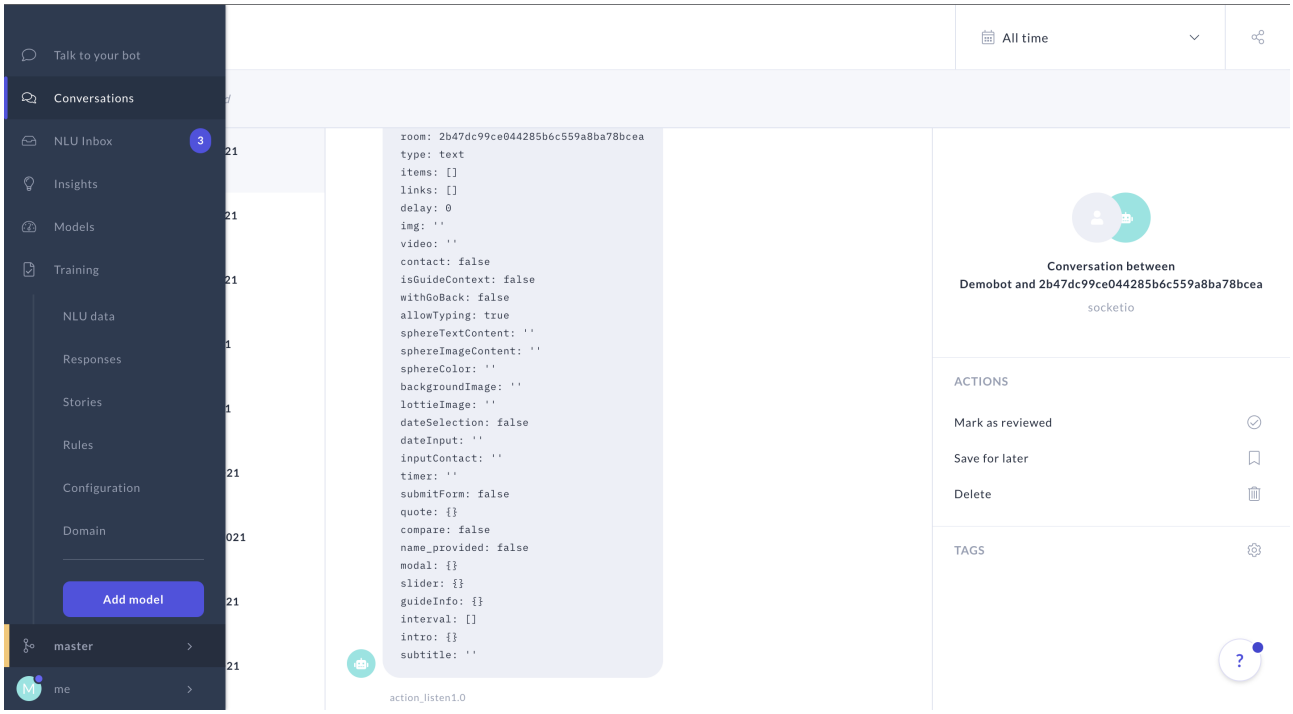

Illustration 15: RasaX Server UI Screenshot

```yaml
name: CI

# Controls when the workflow will run
on:
  push:
    branches: [ main ]
    paths:
      - 'data/**'

 jobs:
  build-model:
    name: Build, test, and upload model
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python 3.7
      uses: actions/setup-python@v1
      with:
        python-version: 3.7

    - name: Install dependencies
      run: |
        python -m pip install --upgrade "pip"
        pip install -r requirements.txt
        python -m spacy download de_core_news_md
    - name: Train model
      run: |
        rasa train
    - name: Run Tests
      run: |
        rasa test core --stories tests/conversation_tests.yml
    - name: Upload Model to RasaX
      run: |
        model_name=`ls models/* | head -n 1`
        curl -k -F "model=@"$model_name"" "http://blender.zkm.de/api/projects/default/models?
                api_token=9c069c93ca3b9e760a8eea08f437ad8d5b077743"
      # curl -X PUT "http://blender.zkm.de/api/projects/default/models/"$model_name"/tags/production"
```

Illustration 16: CI Workflow

```yaml
name: Action Server
# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the main branch
  push:
    branches: [ main ]
    paths:
      - 'actions/**'
  repository_dispatch:
     types: [action-deploy]
 # A workflow run is made up of one or more jobs that can run sequentially or in parallel
 jobs:
   # This workflow contains a single job called "build"
   build:
     # The type of runner that the job will run on
     runs-on: ubuntu-latest
     # Steps represent a sequence of tasks that will be executed as part of the job
     steps:
       # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
       - uses: actions/checkout@v2
       # Runs a single command using the runners shell
       - id: action_server
         name: Build an action server
         uses: RasaHQ/rasa-action-server-gha@main
         with:
           docker_image_name: 'eersada/docker_hub_repo'
           docker_image_tag: 'action-zkm-02-test'
           docker_registry_login: ${{ secrets.DOCKER_HUB_LOGIN }}
           docker_registry_password: ${{ secrets.DOCKER_HUB_PASSWORD }}
           dockerfile: 'Dockerfile_action'
```

Illustration 17: Action Server Workflow

Nachdem der beschriebene CI/CD Prozess erfolgreich war, kann der Chatbot aufgerufen werden und mit den neu hinzugefügten Features benutzt werden.

7. Fazit

Einer meiner größten Hürden war es die Reaktivität aller Komponenten im gesamten Projekt aufrecht zu erhalten. Das heißt, wenn an einer Stelle Daten verändert wurden, mussten diese gleichzeitig überall im Programm aktualisiert werden. Vue liefert direkt einige Dinge, wie Vuex mit, mit welchem zentral alle Daten der einzelnen Komponenten in vordefinierten Abläufen verändert werden können. Dies half mir dabei, einige Probleme zu lösen, die ich vor der Implementierung mit Vuex hatte. [3]

7.1. Verbesserungsansätze

Eine möglich Verbesserung wäre es, die Webanwendung komplett responsive zu mache, damit z.B. auch vom Smartphone darauf zugegriffen werden kann.

Das Design von manchen Ansichten könnte noch an einigen Stellen verbessert werden.

7.2. Erweiterungsmöglichkeiten & Zukunftsaussichten

Ich hatte während und nach der Erstellung des Tools viele Ideen, wie man das Nutzerinterface noch erweitern könnte. Einige davon werde ich hier beschreiben.

In Zukunft könnten noch mehr Optionen hinzugefügt werden, die es in der Benutzeroberfläche ersichtlich machen, welche Deploy Methode einem Projekt zugeordnet ist. Bis jetzt existieren die beiden Optionen das Projekt in Heroku und in der Amazon Cloud S3 zu deployen, es könnten aber noch viele mehr hinzugefügt werden, wie z.B. Microsoft Azure, einem festen Server und viele andere.

Eine Variante zur Erweiterung des Projektes wäre es, fertige Action Code Snippets in das Repository zu pushen (wenn dies mit der GitHub API möglich ist), ohne das man selber die Action in GitHub anlegen muss. Man kann diese nach dem automatischen push in das Repository direkt von der Benutzeroberfläche ausführen.

Eine weitere Idee für eine Erweiterung des Tools wäre einen Rollout über das Tool auszuführen. Das soll heißen, es ist möglich, ein Projekt in mehreren Environments gleichzeitig zu builden, testen und schlussendlich deployen zu können. Ähnlich wie ein

Release, welcher eine fertige Software auf allen definierten Plattformen zugänglich macht. Auch wenn die Environments unterschiedliche deploy Methoden haben.

Damit bestehen gute Voraussetzungen für die zukünftige Anwendbarkeit des Systems.

7.3. Zusammenfassung

Das Thema der Arbeit ist die Erstellung eines Userinterfaces zum Steuern von CI/CD Prozessen. Im Ersten Teil der Arbeit habe ich eine Marktforschung durchgeführt, in der ich mehrere CI/CD Tools analysiert und miteinander verglichen habe. Anschließend werden GitHub Actions, näher beschrieben, welche ich in meinem Tool hauptsächlich nutze.

Der Nächste Schritt ist der Entwurf. Hier stelle ich alle, in dieser Arbeit benutzten Tools und Frameworks kurz vor und beschreibe die Konzeption des Projektes. Daraufhin wird der Entwurf der Pipeline vorgestellt.

Das Projekt indem Ci/CD implementiert werden soll wird kurz vorgestellt, und die Planung der Implementierung umrissen.

In der Implementierung beschreibe ich wie ich beim Erstellen des Benutzerinterfaces vorgegangen bin. Hier spreche ich auch über jede einzelne Ansicht in dem fertigen Benutzerinterface und gehe näher auf die Funktionen und deren Nutzen ein.

Nachdem das Benutzerinterface implementiert wurde, folgt die Implementierung von CI/CD in dem vorher beschriebenen Chatbot Projekt. Dies dient zum Test des Benutzerinterfaces.

Zum Schluss wird ein Fazit gezogen und es werden Verbesserungsansätze genannt.

Um die Arbeit abzuschließen werden einige Erweiterungsmöglichkeiten aufgezählt und die Zukunftsaussichten beschrieben.

8. Literaturverzeichnis

[1] Continuous Delivery, Jez Humble, David Farley, A Martin Fowler Book, Addison Wesley; Illustrated Edition (27. Juli 2010) ISBN 978-0321601919

[2] Continuous Integration, Paul M. Duvall mit Steve Matyas, Andrew Glover, Addison Wesley; Illustrated Edition (2013) ISBN 978-0321336385

[3] Authentication in Vue.js https://www.smashingmagazine.com/2020/10/authentication-in-vue-js/ (abgerufen am 11.08.2021)

[4] GitHub Docs Rest API Actions https://docs.github.com/en/rest/reference/actions (abgerufen am 02.06.202)

[5] Heroku Deploying with Git https://devcenter.heroku.com/articles/git (abgerufen am 13.06.2021)

[6] GitHub Actions https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions (abgerufen am 29.09.2021)

[7] Swagger https://swagger.io/ (abgerufen am 29.09.2021)

[8] Bcrypt https://en.wikipedia.org/wiki/Bcrypt (abgerufen am 29.09.2021)

[9] UML Klassendiagramme https://www.informatik.uni-leipzig.de/~stjaenicke/mup1/s2.pdf (abgerufen am 01.10.2021)

[10] Die besten CI-Tools im Überblick https://www.ionos.at/digitalguide/websites/web-entwicklung/continuous-integration-tools/ (abgerufen am 21.10.2021)

[11] Comparison of Different CI/CD Tools Integrated with Cloud Platform https://ieeexplore.ieee.org/abstract/document/8776985 (abgerufen am 21.10.2021)

[12] Bootstrap Vue

https://bootstrap-vue.org/ (abgerufen am 21.10.2021)

[13] Unterschied zwischen CI, CD und CD?

https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment (abgerufen am 27.10.2021)

[14] Über GitHub gehostete Runners

https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners (abgerufen am 27.10.2021)

[15] Jenkins (Software)

https://en.wikipedia.org/wiki/Jenkins_(software) (abgerufen am 31.10.2021)

[16] Travis CI

https://en.wikipedia.org/wiki/Travis_CI (abgerufen am 31.10.2021)

Glossar

CI.....Continuous Integration
CD.....Continuous Delivery
S3.....Amazon Simple Storage Service
UI.....User Interface
UX.....User Experience
ZKM.....Zentrum Für Kunst und Medien Karlsruhe
deployen.....bereitstellen
Fork.....Entwicklungszweig

9. Source Code

Usus – CI/CD Benutzerinterface - https://github.com/leandergebhardti8/ba-2021

ZKM Chatbot - https://github.com/leandergebhardti8/ZKM-Chatbot-for-BA

10. Links

Usus Anwendung - https://usus-ba-leander.herokuapp.com/

ZKM Chatbot Data - https://blender.zkm.de/chatbot

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und keine andere als die angegeben Literatur benutzt habe. Alle von anderen Autoren wörtlich übernommen Stellen wie auch die sich an die Gedankengänge andere Autoren eng anlehnenden Ausführungen meiner Arbeit sind besonders gekennzeichnet. Diese Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, 31.10.2021	
Ort, Datum	Unterschrift des Studenten