



10. SEPTEMBER 2021


T1000

IMPORT VON TEXTINHALTEN IN EINE MEDIAWIKI-PLATTFORM

KURSIDEM RICO

AZO GMBH & CO. KG

Betreuer: Alexander Till



Inhaltsverzeichnis

| | |
|--------------------------------------------------|----|
| Abstract..... | 4 |
| Überblick über Tätigkeiten der Praxisphase | 5 |
| Semester 1..... | 5 |
| Semester 2..... | 7 |
| Einführung..... | 9 |
| Aufgabenstellung..... | 9 |
| Begriffsdefinition..... | 9 |
| Ziel des MediaWiki-Projektes | 13 |
| Arbeitsprozess..... | 14 |
| Nutzungsbeschreibung meines Programms | 14 |
| Aufsetzen der Arbeitsumgebung | 15 |
| Programmaufbau | 17 |
| MediaWiki API | 17 |
| Umsetzung..... | 19 |
| Programmcode..... | 22 |
| Anwenderschnittstelle (media_wiki.py) | 27 |
| Zusammenfassung und Ausblick..... | 28 |
| Literaturverzeichnis..... | 29 |
| Abbildungsverzeichnis | 29 |

Sperrvermerk

Der Inhalt dieser Arbeit darf an Dritte ohne Genehmigung der
Ausbildungsstätte nicht zugänglich gemacht werden.

Dieser Sperrvermerk gilt für unbegrenzte Dauer.

Datum: _____

Unterschrift: _____

Erklärung

Ich versichere hiermit, dass ich meine Projektarbeit zu Thema: „Import von Textinhalten in eine MediaWiki-Plattform“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Mosbach, 10.09.2021

Rico Kursidem

Abstract

Deutsch

Es ist üblich in großen Firmen, in denen eine große Menge an Informationen leicht zugänglich gemacht werden sollen, interne Wikis anzulegen. AZO arbeitet derzeit an einem Modul für das ACAS Projekt, welches diese Funktion erfüllen soll.

Das Ziel dieser Arbeit ist es herauszufinden ob ein automatisierter Import von Testdateien in Media Wiki möglich ist. Wenn der Import durchführbar ist soll auch ein Prototyp erstellt werden, welcher die Funktionalität beweisen soll.

Um die Funktionalität zu testen wurde die Media Wiki Umgebung auf einer virtuellen Maschine installiert und Recherche zum Import betrieben. Durch das Ansprechen der API ist es möglich direkt Textinhalte auf die Plattform hochzuladen. Mit weiteren Plugins können auch PDF-Dateien hochgeladen und angezeigt werden.

Der Prototyp, welcher die Funktionalität des Imports beweisen soll, wird in dieser Arbeit näher beschrieben. Das Programm kann durch eine Ordnerstruktur Textinhalte Kategorisieren und diese auf die Media Wiki Plattform hochladen. Außerdem können auch PDF-Dateien eingebettet werden.

Diese Arbeit beschreibt den Arbeitsvorgang des Programms und wie es zu bedienen ist. Außerdem wird der Entstehungsprozess des Programms erläutert.

English

It is common in large companies where a large amount of information is to be made easily accessible to create internal wikis. AZO is currently working on a module for the ACAS project, which should fulfill this function.

The aim of this thesis is to find out whether an automated import of test files into Media Wiki is possible. If the import is feasible, a prototype should also be created, which should prove the functionality.

To test the functionality, the Media Wiki environment was installed on a virtual machine and research was carried out on the import. By addressing the API, it is possible to upload text content directly to the platform. Other plugins can also be used to upload and display PDF files.

The prototype, which is supposed to prove the functionality of the import, is described in more detail in this work. The program can categorize text content through a folder structure and upload it to the Media Wiki platform. PDF files can also be embedded.

This thesis describes the working process of the program and how it is to be used. In addition, the development process of the program is explained.

Überblick über Tätigkeiten der Praxisphase

Semester 1

Meine Praxisphase bei AZO startete bereits im September, einen Monat vor Studienbeginn, mit einem Praktikum. In diesem Praktikum wurde ich durch die IT-Abteilung geführt und hatte erste Berührungspunkte mit genutzten Programmiersprachen und Programmierumgebungen. Da ich bereits Erfahrung in objektorientierter Programmierung mitbringen konnte wurden mir Übungsaufgaben gegeben, welche ich absolvieren sollte.

Zunächst traf ich während des Praktikums auf meine Mitauszubildenden in der IT. Zusammen mit ihnen bearbeitete ich ein Praktikumsprojekt am Raspberry Pi. Diese Übung bestand aus dem Löten und Programmieren mehrerer LED und Schaltern auf einer Platine, welche im Nachhinein mit Python programmiert werden sollten. Dieses Projekt führte uns grob in die Mikrokontrollertechniken und Python ein.

Während der ersten Praxisphase bekamen wir dann unsere Laptops und einen Kurs um uns im internen Netzwerk zurechtzufinden. Außerdem installierten wir Visual Studio, um mit C# anfangen zu können.

In den folgenden Wochen lernten ich die Grundlagen der C#-Programmierung in Visual Studio. Neben einem kleinen Taschenrechner und anderen einfachen Programmen erstellte ich auch das Spiel Snake. Da ich zuvor nur wenige Programme objektorientiert erstellt hatte, konnte ich mich bei diesem Programm damit ausprobieren und erkannte das genaue Planung sehr wichtig ist, um ein ordentliches Endresultat zu erreichen.

Ein weiteres Projekt dieser Zeit war ein Sudokulöser. Dieses Projekt zeigte mir wie wenig ich zu diesem Zeitpunkt über Programmstrukturen und Lösungsmöglichkeiten durch Algorithmen wusste. Ich versuchte das Problem zunächst iterativ anzugehen. Die einzige mir bekannte Lösungsmöglichkeit zu dieser Zeit. Nach etwa 4 Tagen Programmierarbeit nahm das Programm zunehmend an Komplexität zu und es traten Probleme auf, welche das Problem als unlösbar erschienen ließen. Ein Auszubildender im zweiten Lehrjahr riet mir dann den Begriff Backtracking nachzuforschen und mit dieser Methode an meine Problemstellung heranzugehen. Nachdem ich seinem Tipp nachgegangen war, hatte mein Programm an Komplexität schlagartig verloren und löste jede Art von Sudokus in sehr kleinem Zeitaufwand. Dieses Projekt lehrte mich bei Problemstellungen erfahrenere Kollegen zu befragen und dass ich noch viel zu lernen hatte.

Während der ersten Praxisphase hatte ich neben der C# Programmierung mich auch anderen Bereichen widmen dürfen. Ich hatte eine gesamte Woche einen Grundkurs zu Microsoft Office 365, in dem ich Applikationen wie Powerapps, Power Automate sowie Forms und Yammer kennenlernte. Hier erstellte ich neben einem eigenen Team bei Teams eine Yammer Seite, eine „Flappy Bird“ Powerapp und mehrere PowerFlows. Meine gelernten Fähigkeiten vereinte ich dann in der finalen Version der Flappy Bird Powerapp. Man konnte auf die Powerapp von meiner Teams-Website zugreifen und sie im Browser sowie in Teams nutzen. Von jedem Spieler wurden die zehn besten Versuche gespeichert, sowie ein Leaderboard erstellt. Das Leaderboard und die eigenen Bestleistungen konnte

man dann auf der Website einsehen. Als finales Feature konnte man in der App auch die Bestleistungen der Kollegen abfragen.

Bei AZO ist es Teil der Ausbildung einen Grundkurs in E-Technik und Mechanik zu belegen. Dieser Grundkurs dauerte 4 Wochen, wobei die Mechanik davon 2 Wochen einnahm. Im Mechanikkurs lernte ich grundlegende Bearbeitungsformen von Metallen, wie das Schneiden von Blechen und Gewindeschnitt. In diesen zwei Wochen erarbeitete ich mehrere Werkstücke wie einen Briefbeschwerer und ein Modellauto. Durch meine Kenntnisse in Mechanik aus der Realschule konnte ich mich sehr schnell in die Arbeitsabläufe einfinden und meine Werkstücke beenden.

In der dritten Woche des Grundkurses belegte ich und die anderen Auszubildenden der IT einen Grundkurs in E-Technik. Wir bekamen eine Reihe an Schaltungen, welche an einem Steckschrank realisiert werden mussten und im Anschluss dem Ausbilder vorgeführt werden mussten.



Abbildung 1: Realisierung einer Ampelschaltung an einem Steckschrank

Am dritten Tag hatten wir die Aufgaben des Plans vollendet und durften wir uns der Planung einer Fußgängerampel widmen. Die Ampelschaltung wurde mit mehreren taktgesteuerten und normalen Schützen realisiert, welche wir selbst planen, aufbauen und testen mussten. Durch unser schnelles Voranschreiten am Schaltschrank und das Teamwork konnten wir die Aufgabe noch am selben Tag fertigstellen und durften uns an den letzten zwei Tagen der Woche den SPS Programmiergeräten widmen. Hierfür setzten wir eine virtuelle Maschine auf, welche die SPS beschreiben sollte. In der Simens IDE war es möglich in mehreren „Programmiersprachen“ das Steuergerät zu programmieren. Zunächst arbeiteten wir mit FUP (Funktionsplan), einer auf logischen Gattern aufgebauten Sprache. Durch meine Kenntnisse aus dem Informatikunterricht auf dem Technischen Gymnasium waren mir die grundlegenden Funktionen der Gatter bewusst, wobei es sehr spannend war Problemstellungen mit dieser Art der Programmierung zu lösen. Eine

weitere Programmiersprache ist AWL (Anwendungsplan). Hierbei wird hauptsächlich mit Operanden und Variablen gearbeitet, welche mich sehr an Mikrocontrollerprogrammierung erinnerten. Das Programmieren der SPS mit den verschiedenen Sprachen war sehr spannend und sehr hilfreich, um einen Einblick in eine gänzlich andere Tätigkeit bei AZO zu erhalten.

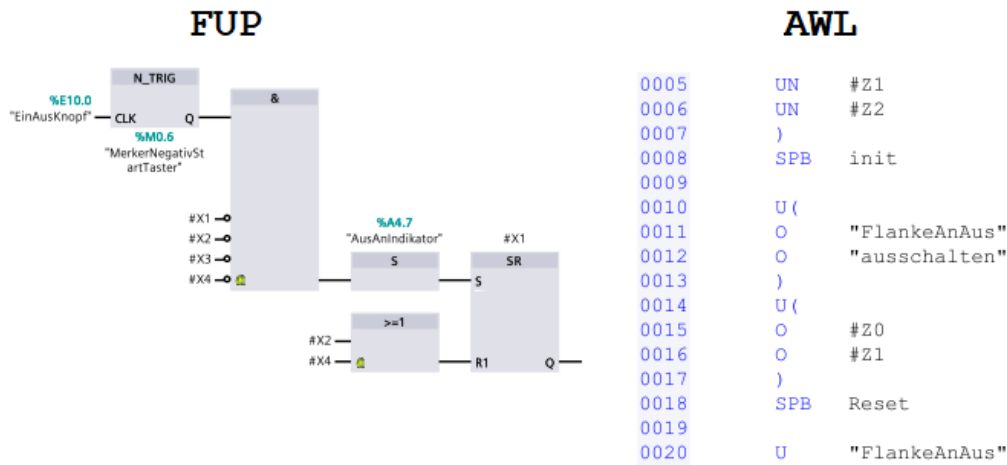


Abbildung 2: Beispiel für SPS Programmiersprachen

Ein weiteres Projekt der ersten Praxisphase bestand darin mit HTML, CSS, JavaScript und SQL eine funktionierende Webanwendung zu erstellen. Mein Ziel war es einen Terminkalender im Browser zu erstellen, in den man Termine frei einschreiben konnte. Diese Termine wurden dann als einzelne Karten wochenweise angezeigt und konnten auch im Nachhinein noch bearbeitet werden. Dieses Projekt gab mir Einblicke in Datenbankstrukturen und das Arbeiten mit JavaScript. Außerdem musste ich das erste Mal mit einem Raspberry Pi als Server arbeiten, da darauf meine SQL-Datenbank gespeichert war. Hier hatte ich auch meine ersten Berührungspunkte mit Python und APIs, da ich eine eigene API schreiben musste.

Semester 2

Im März 2021 gab es eine einmonatige Praxisphase zwischen dem ersten und zweiten Theoriesemester. In diesem Zeitraum war ich in der IT Security eingesetzt und lernte die Netzwerkstruktur bei AZO kennen sowie die Sicherheitsstrukturen. In dieser Zeit gab es mehrere Sicherheitsprobleme mit Windows weshalb unternehmensweit sichergestellt werden musste, dass alle Windowsversionen auf dem neusten Stand waren. Um dieser Aufgabe nachzugehen stellte die IT-Abteilung ein Team zusammen, welches bei Mitarbeitern von AZO anriefen und deren Computer updaten sollten. Meine Aufgabe war es dabei mich mit TeamViewer auf ausgewählte Geräte im ganzen Unternehmen zu verbinden und das Windowsupdate aus dem Softwarecenter zu installieren.

Nachdem ich aus der zweiten Theoriephase im Juli zurück zu AZO kam, hatte ich mehrere Projekte in Vorlesungen bekommen, welche ich daraufhin in meiner Arbeitszeit bearbeiten durfte. Mein Java Projekt in Programmieren II war dabei das umfangreichste. Die Aufgabenstellung war es ein Spiel zu programmieren, bei welchem man die Rolle eines Lagerristen übernahm. Das Wichtigste an diesem Projekt war es objektorientiert die

Aufgabe zu vollenden und eine sinnvolle Programmstruktur zu entwerfen. Das Planen, Umsetzen und Testen dieses Programms dauerte etwa eine Woche und lehrte mich vor allem wie sehr man ein Programm planen sollte bevor man es umsetzt, um einen sauberen Code und leichte Wartbarkeit zu garantieren. Das zweite Projekt aus den Vorlesungen bestand darin eine Website mithilfe von HTML, CSS und PHP zu erstellen. Das Thema für die Website war dabei frei wählbar und auch die Gestaltung war ohne Vorgaben. Es mussten lediglich alle in der Vorlesung erwähnten HTML Tags sinnvoll verwendet werden. Außerdem mussten zwei PHP-Anwendungen in die Seite eingebunden werden. Durch gute Absprache im Team und Nutzung von GitHub konnten wir uns sehr gut innerhalb des Teams abstimmen und so das Projekt zügig vollenden.

Dem Rest der zweiten Praxisphase meines Studiums widmete ich dem Hauptthema dieser Arbeit. Der Integration von existierenden Textinhalten in den bei AZO genutzten MediaWiki-Dienst.

Einführung

Mein Programm nutzt mehrere Systeme und Anwendungen, um Textinhalte auf die MediaWiki Plattform zu importieren. Im Folgenden werde ich auf MediaWiki, Docker und das ACAS Projekt näher eingehen, um das Umfeld meines Programms zu erläutern. ACAS ist das Projekt bei AZO, in welchem die MediaWiki Plattform integriert ist und somit auch mein Programm. Docker ist die Softwareplattform, welche zur Verwaltung von Anwendungen auf den Servern von ACAS genutzt wird.

In dem Programm wird noch weitere Drittanbietersoftware genutzt. Pandoc ist ein Docker Image, welches zur Konvertierung der Textformatierung in das MediaWiki-Format genutzt wird. Außerdem wird eine MediaWiki Erweiterung implementiert, welche es ermöglicht PDFs auf Wiki Seiten einzubetten. Diese Erweiterung nennt sich PDFEmbed und wird dem MediaWiki Image zugefügt.

Das Programm besteht aus einer .sh Anwendung, welche das Dateihandling und die Kommunikation mit Pandoc übernimmt. Die Verbindung zum MediaWiki Container bearbeitet ein Pythonscript, welches von der .sh Anwendung aufgerufen wird.

Aufgabenstellung

Im internen Firmenprojekt ACAS soll ein Wiki angelegt werden. Dieses soll mit der Opensource Software MediaWiki realisiert werden. Um Seiten auf diesem Wiki zu erstellen müssen Textdateien in ein MediaWiki-Format gebracht werden und manuell eingefügt werden.

Das Ziel meines Projektes war es den Import von vorhandenen Textinhalten, wie Worddateien, Markdown, txt-Dateien und PDFs, automatisiert und einfach zu realisieren. Zunächst sollte ich recherchieren ob und mit welchen Mitteln ein Import möglich ist. Daraufhin sollte ein Prototyp die Funktionalität eines Imports beweisen.

Der erste Aufgabenteil war das Einrichten der Umgebung. Hierfür musste eine Ubuntu VM aus Vorlagen der Abteilung Technique aufgesetzt werden. Außerdem musste Docker installiert und mit weiteren Einrichtungsskripten aufgesetzt werden.

Ein besonders wichtiger Anwendungsfall, welcher umgesetzt werden sollte, war der Import von PDF-Dateien. Viele Dokumente und Pläne liegen im Moment als PDF vor und sollen direkt in das Wiki eingebunden werden können.

Eine Seite auf der MediaWiki-Plattform kann durch Kategorisierung besser auffindbar gemacht werden. Die Kategorisierung soll durch eine Ordnerstruktur realisiert werden, welche auch zu implementieren war.

Begriffsdefinition

ACAS

ACAS ist eine Automatisierungsplattform, mit der die Automatisierung und Funktionalität von Anlagen sichergestellt werden soll. Sie ist die zentrale Kommunikationsschnittstelle zwischen Kunde und Anlage. In dem Projekt ACAS befinden sich alle Funktionen, die damit zu tun haben, wie der Kunde auf die Anlage Einfluss nehmen und wie er die Informationen der Anlage auslesen kann.

ACAS hat das Ziel die Anlage (also die Ventile, Motoren, usw.) zu automatisieren. Die Plattform ist dafür verantwortlich, dass bei gewünschter Funktion einer Anlage auch die nötigen Signale an diese gesendet werden, damit die Anlage automatisiert funktionsfähig wird. Soll eine Anlage also zwei Zutaten mischen so ist die Funktion, das Mischen der grobe Überbegriff für die Aktion, welche die Aktorik in der Anlage durchführen muss. Dieser Befehl wird dann von ACAS übersetzt in Signale an beispielsweise einen Motor, wann wie schnell und wie oft dieser sich bewegen muss.

Eine weitere Aufgabe von ACAS ist es die Anlage zu visualisieren. Die Visualisierung der Anlage umfasst alles was mit der Darstellung von Daten oder Abläufen in der Anlage zu tun hat. Es versucht dem Kunden, welche die Anlage auslesen und daraufhin bedienen muss, möglichst deutlich zu vermitteln was die Anlage momentan tut. Eine Visualisierung kann beispielsweise eine Grafik der Anlage sein, in der sichtbar wird welche Anlagenteile zusammengeschlossen sind oder welche Motoren gerade laufen und welche nicht. Visualisierung besteht aber auch darin Sensoren auszulesen und die Daten überschaubar zu repräsentieren. Im Bereich der Visualisierung ist es wichtig alle Daten so komplex wie nötig und so überschaubar wie möglich darzustellen. Die Plattform soll die Gegebenheiten der Anlage (Behälter, Sensoren, Aktoren, Verrohrung) dem Kunden so transparent und flexibel wie möglich zur Verfügung stellen.

ACAS soll auch Störungen und fehlerhafte Abläufe in der Anlage erkennen und ein Alarmsystem integrieren, welches Fehlfunktionen verhindern und Schäden vermeiden soll.

Auch Bedien- und Signalisierungselemente gehören zu ACAS. Also alle Bauteile, welche es ermöglichen Einfluss auf den Arbeitsprozess in der Anlage zu nehmen. Hierzu gehören Taster, Signale/Leuchten, SPS-Panels und andere Endgeräte. Ein mobiles Gerät, welches beispielsweise auf Browseranwendungen zugreift, die mit der Anlage kommunizieren gehört auch dazu. In dem Bereich der Bedienbarkeit ist es wichtig die Anlage individuell auf den Anwendungsfall zu optimieren. Beispielsweise muss sichergestellt werden, dass ein Taster, der an der Anlage verbaut wird, ggf. mit Handschuhen bedienbar ist. Hierbei muss auch darauf geachtet werden, dass die Anlage komfortabel und effektiv bedient werden kann.

ACAS wird im Zuge eines internen Projektes aufgebaut. Es ist eine Plattform, welche von Grund auf bei AZO entwickelt wird. Das Ziel von ACAS ist es die gesamte Expertise und das Wissen von AZO zur Automatisierung von Anlagen bestmöglich so umzusetzen, dass ein optimales System entsteht. Schnittstellen zwischen Systemen unterschiedlicher Hersteller sollen so minimiert werden, um den Anlagenbetrieb möglichst reibungslos zu realisieren. Durch die Entwicklung bei AZO kann sichergestellt werden, dass ACAS bestmöglich mit AZO Anlagen funktioniert und perfekt mit der Fertigungstechnik abgestimmt ist.

ACAS soll aber nicht nur bei AZO genutzt werden, sondern soll auch bei anderen Herstellern die beste Wahl zur Automatisierung von Anlagen sein.

Die MediaWiki Anwendung ist ein Teil des ACAS Projektes. Das Wiki soll Informationen über das Betreiben von Anlagen, Fehlerbehebung, Wartung und weiteres beinhalten. Es

soll dazu beitragen, dass der Kunde reibungslos mit Anlagen, die ACAS nutzen, arbeiten kann.

MediaWiki

MediaWiki ist eine Verwaltungssystem für Wiki Inhalte. Es wurde primär entwickelt, um Wikipedia zu verwalten. MediaWiki ist darauf ausgelegt vielen Nutzern gleichzeitig Zugriff auf gespeicherte Informationen zu ermöglichen und Daten geordnet in Kategorien abzuspeichern.

MediaWiki ist frei nutzbar und ermöglicht Nutzern, die Zugriff haben, auf Informationen über den Browser zuzugreifen und sie selbst hochzuladen. Die normale Formatierung des Textes erfolgt über ein MediaWiki-Format. Inhalte können sowohl abschnittsweise als auch auf der ganzen Seite ersetzt und geändert werden.

Eine weitere Eigenschaft der MediaWiki Plattform ist das Archiv, welches über alle Seiten angelegt wird. Wird etwas geändert so wird die Änderung, der Zeitpunkt und der Nutzer, welcher die Änderung vorgenommen hat, dokumentiert und alle Versionen sind im Nachhinein einsehbar und wiederherstellbar.

Eine MediaWiki Seite folgt immer einem ähnlichen Aufbau. Die Seite hat einen einmaligen Titel. Der Inhalt selbst ist daraufhin in Abschnitte mit eigenen Überschriften aufgeteilt. Bilder und Tabellen können ebenfalls eingebunden werden. Die Bilddateien sind in einem Datenspeicher abgespeichert, wobei sie eine einmalige Bezeichnung benötigen. Die Bilddateien werden auf der MediaWiki Seite dann mit ihrer Bezeichnung importiert und angezeigt.

Um mit der MediaWiki Plattform zu kommunizieren und einen Datenaustausch zu erzeugen muss man die MediaWiki API nutzen. Diese ermöglicht eine Vielzahl von Möglichkeiten mit der Plattform zu interagieren.

Docker

Docker ist eine Softwareplattform, die es ermöglicht viele Anwendungen parallel auf einem Hostsystem zu betreiben. Dazu nutzt Docker ein Containersystem. Jede Anwendung läuft dabei in seinem eigenen Container, welcher alle notwendigen Eigenschaften besitzt, um die Anwendung auszuführen, beispielsweise benötigte Bibliotheken. Ein Docker Container wird immer von einem Image erzeugt, welches in einer Docker File gestartet wird.

Durch die Nutzung einer containerorientierten Verwaltung der Anwendungen sind diese universell einsetzbar. Das bedeutet das die Funktion der Anwendung nicht vom Betriebssystem oder dem Ablageort auf dem Computer abhängt. Das macht Dockercontainer einfach zu programmieren, einfach einzurichten und sehr leicht wartbar, da die Voraussetzungen, um einen Container zu starten nur Docker selbst und das dazugehörige Image sind.

Container funktionieren wie kleine, isolierte Computer oder VMs, welche ihre eigenen Ressourcen haben. Dadurch können sie sehr flexibel gestartet, gestoppt oder neugestaltet werden, ohne dabei andere Prozesse zu behindern oder das Hostsystem zu beeinflussen.

Container werden immer aus vorgefertigten Images erstellt, welche ganz einfach von Docker Hub heruntergeladen werden. Auf Docker Hub werden Images Opensource basierend geteilt und veröffentlicht. Dadurch, dass die Container keine Voraussetzungen auf dem Hostsystem benötigen, kann ein solches Image einfach heruntergeladen werden und daraufhin kann von diesem Image ein Container erstellt werden.

Docker hat viele Vorteile im Gegensatz zu herkömmlichen Systemen mit VMs. Beim Nutzen von VMs müssen diesen statische Ressourcenmengen zugeordnet werden. Es muss ein komplettes Betriebssystem emuliert werden. Außerdem muss jede VM über einen Hypervisor, einen Übersetzer, mit dem Hostsystem kommunizieren. Docker umgeht diesen Hypervisor indem es direkt mit dem Hostsystem kommunizieren kann. Dadurch können deutlich mehr Applikationen über Dockercontainer als über herkömmlich VMs gestartet werden. Container können auch als Host für eigene Betriebssysteme genutzt werden.

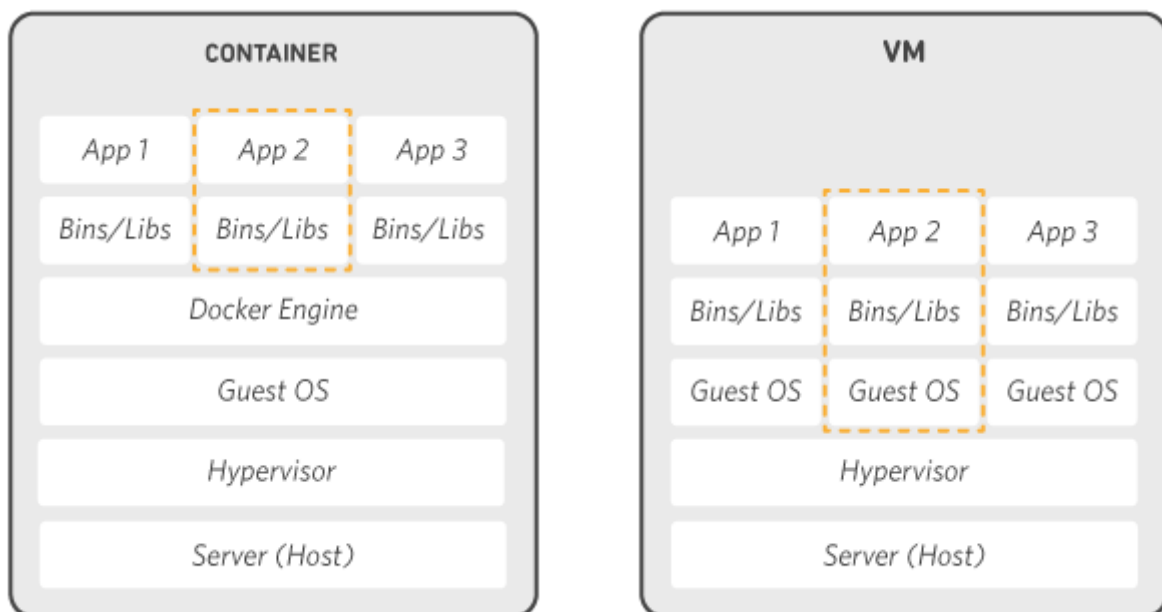


Abbildung 3: VM und Docker im Vergleich

Docker benötigt auch weniger Speicherplatz, da es ein Speichersystem nutzt, welches es ermöglicht, dass Container, welche auf dieselbe Datei zugreifen müssen, nur einen Verweis auf diese Datei speichern und keine Kopie. Dadurch können oft genutzte Daten zentral gespeichert werden und dann mehrfach genutzt werden.

Container werden durch Images erstellt. Diese dienen als Bauplan für die Container. Images können einfach von Docker Hub heruntergeladen werden. Docker Hub ist ein Onlineportal auf welchem Opensource Images zum Herunterladen bereitstehen. Dadurch dass Container kaum Voraussetzungen auf dem Hostsystem benötigen können Container aus Images von Docker Hub einfach ausgeführt und genutzt werden. Das macht es besonders komfortabel neue Systeme einzurichten. Images können auch mit einer eigenen Docker Datei erstellt werden.

Ziel des MediaWiki-Projektes

Das MediaWiki Projekt bei AZO soll sich zu einer zentralen Datenbank zur Informationsspeicherung zu allen Themen bei AZO entwickeln. Primär ist dabei im Focus Kunden zu ermöglichen Fehlercodes oder andere Probleme im Wiki nachschlagbar zu machen, um eventuelle Selbstreparaturen zu ermöglichen. Außerdem sollen mit dem Wiki Informationen zur reibungsfreien Bedienung von Anlagen zu Verfügung gestellt werden

Ein Kunde soll bei Auftreten eines Fehlers beispielsweise den Fehlercode im Wiki suchen können und so eventuell einen Guide bekommen, wie er den Fehler beheben kann. Das Media Wiki soll dabei auf Deutsch und Englisch angeboten werden, um so auch dem internationalen Schwesterunternehmen von AZO zu ermöglichen Inhalte hochzuladen und abzurufen.

Neben den beiden „KundenWiki“ soll auch ein internes azoWiki aufgebaut werden. Hier werden Informationen gespeichert, welche häufig von internen Abteilungen abgerufen werden müssen. Spezielle Anwendungsfälle sind:

- die Speicherung von Anlageplänen der Konstruktionsabteilung
- die Speicherung von Elektroplänen der MCS-Abteilung
- die Speicherung von Plänen eingesetzter Sensorik und Aktorik

Alle oben genannten Pläne liegen häufig in PDF-Form vor. Deshalb war es besonders wichtig herauszufinden wie und ob man PDF-Dateien in die MediaWiki Plattform integrieren kann.

Arbeitsprozess

Nutzungsbeschreibung meines Programms

Das Programm ist derzeit noch eine .sh Anwendung, welche neben der Ordnerstruktur für die Daten und Skripte liegen muss. In späteren Version ist es vorstellbar diese Anwendung als Dockercontainer zu starten und so die Anwendung komplett in die Dockerumgebung des Servers zu integrieren.

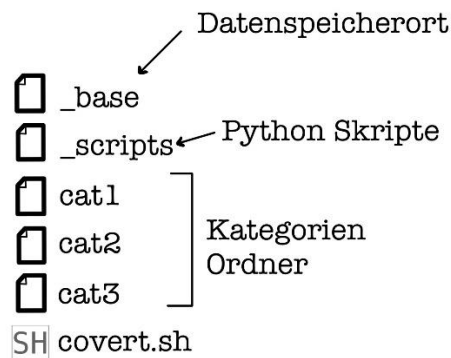


Abbildung 4: Ordnerstruktur

Die Ordnerstruktur besteht aus einem „_base“ Ordner in welchem die Daten abgelegt werden müssen, welche bei Ausführen der „convert.sh“ konvertiert und auf MediaWiki hochgeladen werden sollen. In „_scripts“ liegen alle Pythonskripte und Bibliotheken, welche benötigt werden. Dieser Ordner kann von Nutzern der Anwendung ignoriert werden. Jeder weitere Ordner stellt eine Kategorie dar, in welche die MediaWiki-Seite eingebunden werden soll. Es können beliebig neue Kategorien hinzugefügt werden. Der Name des Ordners ist der Name der Kategorie, welche an die Textdatei angehängt wird.

Wird die Textdatei in „_base“ abgelegt, wird die Bezeichnung der Datei als Titel genutzt. Die Datei „Test.docx“ würde den Titel „Test“ zugeordnet bekommen. Unter diesem Titel kann auf diese Seite verlinkt oder sie direkt gesucht und aufgerufen werden. Existiert bereits eine Seite mit diesem Titel wird diese überschrieben und der bisherige Inhalt wird als Archiveintrag abgespeichert. Sollte die Seite bereits vorhanden sein und es werden keine Änderungen vorgenommen so wird keine Änderung vorgenommen und auch kein Archiveintrag erstellt.

Um Kategorien zu dem Textdokument hinzuzufügen muss aus einem der Kategorieordner auf die Hauptdatei verwiesen werden. Dies wird auf Linux mit dem „ln“-Befehl erreicht. Hierzu muss man erst in den Kategorieordner navigieren und dann folgenden Befehl eingeben.

```
ln -s ../_base/<Zielfile>
```

Wenn in allen gewünschten Kategorieordnern die Zielfile verlinkt ist, so kann die `convert.sh` ausgeführt werden und der Textinhalt wird ausgelesen und hochgeladen.

Verwendbare Datentypen sind Worddokumente, CSV Dateien, Markdown und .txt Dateien, die bereits im MediaWiki-Format sind. Außerdem können auch PDF-Dateien so eingebunden werden.

Aufsetzen der Arbeitsumgebung

Der erste Schritt in der Bearbeitung meines Projektes war es die Arbeitsumgebung für mich zugänglich zu machen. Ziel war es eine Ubuntu VM auf meinem Laptop aufzusetzen und auf dieser einen Teil des ACAS-Serversystems zu starten um mit der Implementierung meines Programms anfangen zu können.

Zu Beginn musste eine normale Ubuntu Server VM mit einer .iso Datei aufgesetzt werden. Um die ACAS Umgebung einzurichten waren mehrere Schritte nötig.

Um das System auf Acas vorzubereiten mussten mehrere Installationsskripte auf dem Server ausgeführt werden. Diese Skripte sind bei uns in der SVN abgespeichert. Die SVN ist das interne Versionsverwaltungssystem bei AZO. Der Ordner musste in das Home Verzeichnis der Ubuntu Maschine kopiert werden und die „installation.sh“ musste ausgeführt werden. Dieses installiert unter anderem Docker und setzt wichtige Umgebungsvariablen. Nach der Ausführung des Skriptes war die VM bereit die ACAS Umgebung zu Installieren.

Für die Installation von ACAS musste ich erneut auf das SVN zugreifen. Ich benötigte ein ACAS Template welches ich in den Ordner „bin\Ubuntu“ kopieren musste. Da der Server direkt auf meinem Laptop lief, war es notwendig Ressourcen zu sparen und meinen Laptop etwas zu entlasten. Deshalb musste in der up.sh einige Zeilen auskommentiert werden, um möglichst wenige Anwendungen in der VM tatsächlich zu starten. Es war sicherzustellen, dass in der Testumgebung nur alles Nötige für die Funktionalität von MediaWiki vorhanden war. Durch das Ausführen der up.sh wurde das ACAS Verzeichnis auf meinem Server lauffähig gemacht. Nachdem das up.sh Skript ausgeführt worden war, war meine ACAS Installation durchgeführt und meine Testumgebung aufgesetzt.

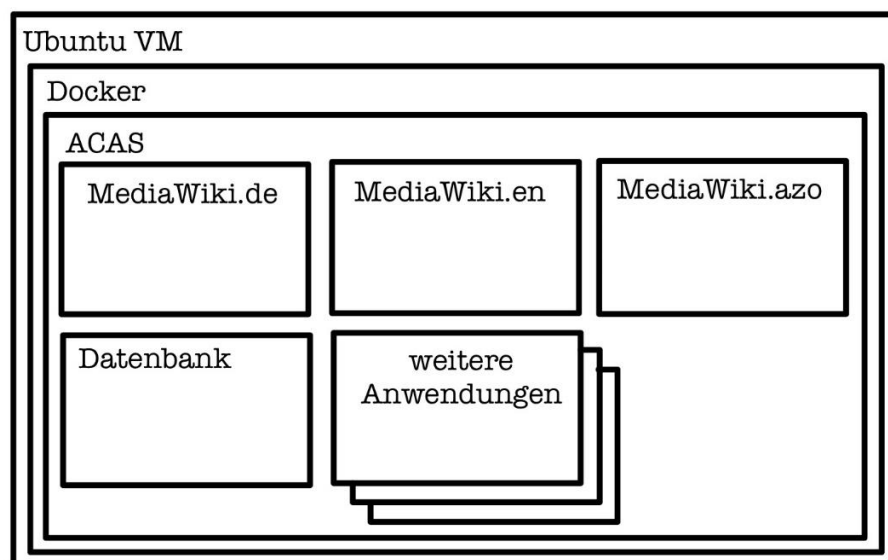


Abbildung 5: Serverumgebung der ACAS Plattform

Das Aufsetzen der Arbeitsumgebung stellte sich für mich als großes Hindernis heraus, da ich kaum Kenntnisse zu den meisten Themen hatte. Ich arbeitete das erste Mal mit einem Ubuntu Server und hatte auch noch kaum Erfahrung mit VMs gesammelt. Außerdem hatte

ich bis zu diesem Zeitpunkt bei AZO noch nie mit ACAS arbeiten müssen, da ich zuvor in anderen Bereichen, wie der IT-Sicherheit oder dem Support eingesetzt war. Auch mit SVN zu arbeiten fiel mir zu Beginn sehr schwer, weshalb ich oft um Hilfe bitten musste. Nach etwa einer Woche konnte ich grundlegend mit SVN umgehen und hatte auch bereits erste Erfahrungen mit Docker sammeln können.

Programmaufbau

MediaWiki API

MediaWiki stellt zur Kommunikation mit der Plattform eine API zu Verfügung. Eine genaue Dokumentation dieser ist auf der Plattform unter /api.php (auf dem Azoserver: <https://de.wiki.server.azo/api.php>) zu finden.

Die media_wiki.py Schnittstelle nutzt drei Befehle der API. Verbindung und Verifizierung auf den Server, Hochladen von Textdateien und Hochladen von Bild bzw. PDF-Dateien. Media_wiki.py nutzt dafür die „requests“ Bibliothek.

```
import requests
```

Um im Programm sich die Daten der Verbindung zum MediaWiki-Server zu speichern wird eine Session angelegt.

```
session = requests.Session()
```

um eine Get-Request an die MediaWiki API zu senden wird die Funktion `session.get` genutzt. Dieser Funktion müssen die `mediawiki_url` und die `params` übergeben werden. Außerdem wird `verify=False` gesetzt. Dies ist aus Sicherheitsgründen nicht empfehlenswert und das Programm wirft auch mehrere Warnungen aus. Allerdings ist mein Programm nur eine Demonstration der Funktionalität und die Verifizierung mit den Zertifikaten hatte nicht funktioniert, weshalb ich mich entschieden habe diesen Weg zu gehen.

Um nun Get-Requests an die API zu senden müssen nur noch die `params` gesetzt werden. Dies wird beispielsweise in diesem Code gesetzt.

```
params_1 = {  
    "action": "query",  
    "meta": "tokens",  
    "type": "login",  
    "format": "json"  
}
```

In `params_1` sind nun mehrere Werte gesetzt. Es soll die Aktion „Query“ ausgeführt werden (`"action": "query"`) um einen `login_token` zu erhalten (`"meta": "tokens"`, `"type": "login"`). Die Antwort der API soll in Form eines `"json"` Strings erfolgen.

Nachdem der `login_token` erhalten wurde wird eine `login` Anfrage an den Server gesendet.

```
params_2 = {  
    "action": "login",  
    "lgname": login_Info[0],  
    "lgpassword": login_Info[1],  
    "format": "json",  
    "lgtoken": login_token  
}
```

Dafür werden die Anmeldeinformationen des Bots (login_Info) und der login_token übergeben.

Als dritten Schritt des Verbindens zu MediaWiki wird ein csrf_token abgefragt.

```
params_3 = {  
    "action": "query",  
    "meta": "tokens",  
    "format": "json"  
}
```

Die connect Funktion gibt dann die session und den csrf_token zurück. Diese werden in folgenden Schritten genutzt, um Daten an MediaWiki zu senden.

Um Seiten auf MediaWiki zu beschreiben bietet die MediaWiki API eine "edit" Action an. Diese Post-Request müssen der Seitentitel (pSiteTitle), der Textinhalt (pText) und der csrf_token übergeben werden.

```
PARAMS = {  
    "action": "edit",  
    "format": "json",  
    "title": pSiteTitle,  
    "text": pText,  
    "token": csrf_token  
}
```

Um Dateien auf das Wiki zu laden gibt es die "upload" Aktion. Dabei müssen in den Parametern der Dateiname (filename) und csrf_token festgelegt werden.

```
PARAMS = {  
    "action": "upload",  
    "filename": filename,  
    "format": "json",  
    "token": csrf_token,  
}
```

Außerdem muss bei der Post-Request die Datei selbst mitgegeben werden. Dies erfolgt durch diesen Code:

```
file = {'file':(file_safe, open("_base/" + file_safe, 'rb'), 'multipart/form-data')}  
response = session.post(mediawiki_url, files=file, data=PARAMS)
```

Bei dem Hochladen von Dateien können mehrere Fehler auftreten, welche ich durch eine rekursive Funktion abgefangen habe. Sollte es bereits eine Datei geben, welche denselben Dateinamen hat, wird die Funktion erneut aufgerufen mit einem neuen, zufällig erstellten Dateinamen. Existiert die Datei bereits in der MediaWiki Datenbank wird der Dateiname der bereits bestehenden Datei zurückgegeben.

Umsetzung

Überblick

Die Aufgabe des Programms ist die Konvertierung und die Anzeige der Daten auf MediaWiki. Dafür muss das Programm die Daten zunächst einlesen und durch Ermitteln des Dateityps entscheiden wie es weiter verfahren muss.

Das Programm arbeitet dafür mit mehreren Skripten. Convert.sh kümmert sich um die Konvertierung der Daten in das MediaWiki-Format, führt die Python Skripte aus welche über die media_wiki.py mit der offiziellen MediaWiki API kommunizieren.

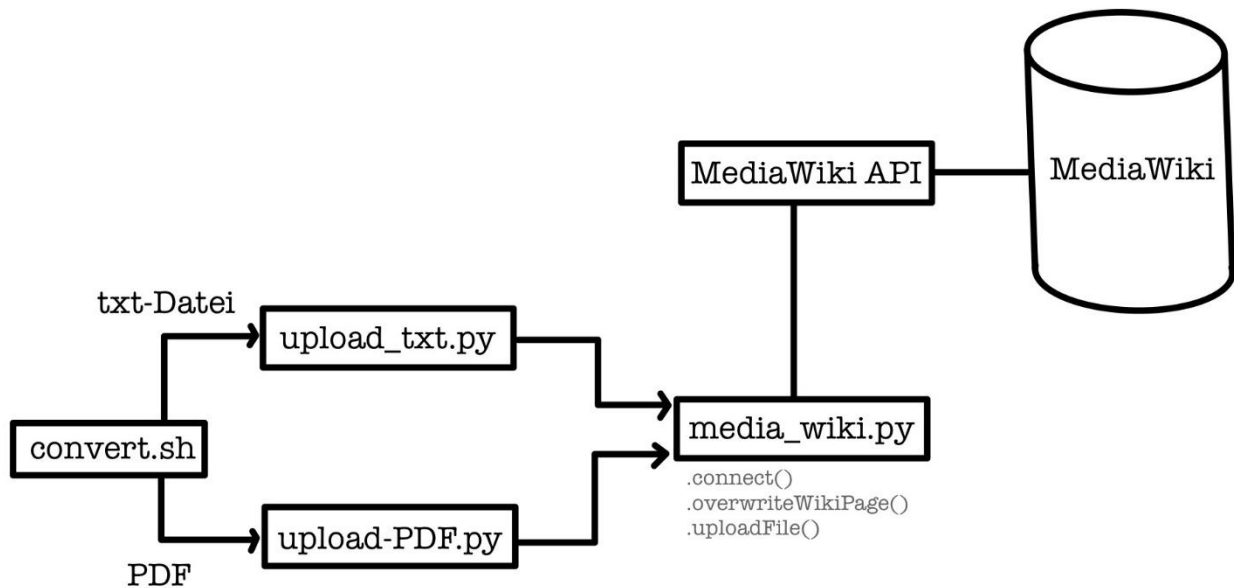


Abbildung 6: Programmaufbau

Einlesen und konvertieren

Zunächst loopt die convert.sh durch alle Dateien im „_base“ Ordner. Für jede Datei dort wird der Titel, also der Dateiname, der Dateityp und die Dateiendung gespeichert. Um die Kategorien zu ermitteln, sucht das Programm die Kategorie Ordner nach Verlinkungen auf die Hauptdatei ab. Es wird ein Kategorienstring erstellt, der alle Kategorien abspeichert und später an die Datei angehängt werden muss.

Es gibt drei Anwendungsfälle, welche auftreten können. Der erste Fall ist die Konvertierung einer Textdatei. In diese Kategorie fallen alle Formen von Textinhalten wie Markdown, txt und auch docx. Im Skript selbst werden Word-Dateien extra behandelt da sie bei Nachfrage des Dateityps nicht „text/plain“ zurückgeben. Diese Dateien müssen zunächst in das richtige Format konvertiert werden. Dafür nutzt mein Programm Pandoc.

Pandoc ist eine kostenfreie Anwendung, um Text in andere Formate zu konvertieren. Beispielsweise kann Pandoc aus Word-Dateien den Text auslesen und die Formatierung einer MediaWiki Seite zurückgeben. Dieses Feature mache ich mir zu Nutze, um meine eingelesenen Dateien in das MediaWiki-Format zu konvertieren.

Der Befehl zur Konvertierung von Textdateien mit einem Pandoc Container:

```
sudo docker run --rm --volume "`pwd`::/data" --user `id -u`:`id -g` pandoc/core -  
t mediawiki -i $FILE -o ${FILENAME%.*}.txt
```

Hier wird ein Dockercontainer aus einem Pandoc Image erstellt. Dieser konvertiert dann den Input („-i“) in das Zielformat („-t“) und speichert die konvertierte Version danach in den Output („-o“) ab. Durch die Mitgabe des „--rm“ Flags wird der Container wieder entfernt, sobald er die Konvertierung durchgeführt hat. Dies ist notwendig, um Ressourcen und Speicherplatz nicht zu überlasten, da bei jeder Konvertierung ein neuer Container gestartet wird.

Da Pandoc keine CSV Dateien auslesen und konvertieren kann müssen diese separat bearbeitet werden. Diese Konvertierung erfolgt später im Python Skript durch die Funktionen `tableLayout(text)` und `convertCSV(csv_file)`. Um zu erkennen welche Textdateien CSV Dateien sind, wird der CSV Datei die Kategorie „CSV“ angehängt.

Nachdem der Inhalt der Datei in das MediaWiki-Format gebracht und zwischengespeichert wurde, wird das Python Skript aufgerufen.

```
python3 _script/upload_txt.py ${FILENAME%.*}.txt $CATEGORIE_STRING
```

Diesem Skript wird lediglich der Kategorienstring (\$CATEGORIE_STRING) mitgegeben.

Wird eine PDF-Datei eingelesen, muss, bevor die Verlinkung auf die PDF auf der MediaWiki-Seite geschrieben wird, zunächst die Datei in den Datenspeicher des Wikis abgespeichert werden.

```
python3 _script/upload_PDF.py $FILENAME $CATEGORIE_STRING
```

Dafür wird ein anderes Skript ausgeführt, welches den Dateinamen der PDF und den Kategorienstring (\$CATEGORIE_STRING) übergeben bekommt.

Hochladen der Textinhalte

Um das Hochladen der Textinhalte zu bearbeiten werden zwei unterschiedliche Skripte und eine selbstgeschriebene Anwenderschnittstelle verwendet. Unter dem Kapitel „Anwenderschnittstelle (media_wiki.py)“ kann eine schnelle Funktionsbeschreibung der einzelnen Funktionen der Anwenderschnittstelle nachgeschaut werden.

Das erste Skript (upload_txt.py) wird verwendet um Textdokumente, welche bereits von der .sh in das MediaWiki-Format konvertiert wurden, auf die MediaWiki-Seite zu schreiben.

Die „upload_txt.py“ braucht einen Übergabewert, um ausgeführt zu werden. Außerdem benötigt es die Anmeldedaten des Bots, welcher ein Konto auf der MediaWiki-Plattform hat. Diesen Bot kann man anfordern, indem man auf MediaWiki zur Spezial:BotPasswords einen neuen Bot erstellt. Dieser wird dann als Bot erkannt und ein Skript kann sich mit seinen Anmeldedaten einloggen. In diesem Programm sind die benötigten Anmeldedaten in einem anderen Pythonprogramm abgespeichert.

Zunächst werden alle wichtigen Daten in Variablen gespeichert und die Sprache des Wikis wird auf „de“ gesetzt. Die Kategorien werden ausgelesen und in einen String gespeichert, welcher vor die MediaWiki-Datei gespeichert werden muss.

Mit der Funktion `media_wiki.connect` wird eine Verbindung mit dem Wiki hergestellt und eine Anmeldung mit den Bot-Anmeldedaten durchgeführt. Die dabei zurückgegebene `session_item` Liste wird danach benutzt, um sich für die Änderungen zu verifizieren.

Da die Verbindung nun steht und die Anmeldung erfolgreich war, kann mit `media_wiki.overwriteWikiPage` fortgefahren werden. Dieser Funktion werden die Session Items, die MediaWiki URL sowie der Titel der Seite und ein String mit dem Inhalt der Textdatei, welcher auf die Seite geschrieben werden soll, übergeben.

```
category + media_wiki.tableLayout(readFile(filename, category))
```

Der Inhalt setzt sich aus den Kategorien und dem gelesenen Inhalt aus der Textdatei zusammen. In `readFile(filename, category)` wird die Textdatei ausgelesen und nötige Formatierungen durchgeführt.

```
def readFile(filename, category):
    if "CSV" in category:
        print("CSV -----")
    - "!!!!")
    return media_wiki.convertCSV(open(filename, "r"))
    else:
        return open(filename, "r").read()
```

In `readFile` wird die Datei geöffnet und als String (`.read()`) zurückgegeben. Hat die Datei das Keyword „CSV“ angehängen, so wird zusätzlich zum Auslesen der Datei noch die gesamte CSV mit `media_wiki.convertCSV` konvertiert und ins MediaWiki-Format gebracht.

Ist die Funktion ausgeführt und der Text auf die MediaWiki Seite geschrieben beendet sich das Programm.

Der zweite Anwendungsfall ist das Darstellen einer PDF. Hierfür wird ein anderes Skript von der Bash aufgerufen, die „`upload_PDF.py`“. Diese muss neben dem Hochladen der Verlinkung auf die PDF-Datei auch die PDF selbst in den Speicher der MediaWiki Plattform speichern. Um diese Aufgabe zu erfüllen wird neben `media_wiki.connect` und `media_wiki.overwriteWikiPage` eine weitere Funktion genutzt.

Die Funktion `media_wiki.uploadFile` lädt die Datei, welche unter dem Dateinamen `filename` abgespeichert wurde, auf die MediaWiki Plattform hoch. Ist die Datei bereits dort gespeichert wird die Version auf dem Server genutzt und der Dateiname dieser Datei zurückgegeben. Existiert bereits eine Datei im Speicher des Wikis mit derselben Bezeichnung wird die Datei umbenannt und der neue Dateiname zurückgegeben. Trifft keiner der beiden anderen Fälle ein wird der normale Dateiname zurückgegeben. Dieser Rückgabewert ist also in jedem Fall der Dateiname der PDF im Wiki Speicher und damit auch die Datei, auf die auf der Wiki Seite verlinkt werden muss.

Die Einbindung der PDF auf die Wiki Seite erfolgt mit einer Extension namens „`PDFEmbed`“.

Der Syntax, um eine PDF einzubetten ist:

```
<pdf>Datei:Dateiname</pdf>
```

Dieser Sting wird durch diese Anweisung mit den gewünschten Kategorien und dem Dateinamen erstellt und übergeben.

```
category + "<pdf>Datei:" + filename + "</pdf>"
```

Ist die Datei eingebunden und im Speicher des Wikis abgelegt, beendet sich das Python Skript.

Abschluss

Sind die Python Skripte mit dem Hochladen des Inhalts fertig, löscht die .sh alle temporär erstellten Dateien.

```
rm ${FILENAME%.*}.txt
```

Es ist denkbar an dieser Stelle auch alle Dateien aus dem „_base“ Ordner und die Verlinkungen in den Kategorieordnern zu löschen. Diese Funktion muss eingebunden werden, wenn das Programm eingesetzt wird und automatisiert in zeitlichen Abständen ausgeführt werden soll.

Programmcode

convert.sh

```
#!/bin/bash
```

```
#noch unabhängig von Speicherort von convert.sh machen
```

```
FILES=$(find _base -type f)
```

```
FOLDERS=$(ls -d */)
```

```
for FILE in ${FILES}; do
```

```
    #Ermittle den Dateinamen
```

```
    FILENAME=${FILE##*/}
```

```
    echo FILENAME: $FILENAME
```

```
    #Ermittle die Kategorien welche hinzugefügt werden sollen
```

```
    CATAGORIE_STRING=" "
```

```
    echo -n "CATEGORIE: "
```

```
    for CATEGORIE in ${FOLDERS}; do
```

```
        if [ $CATEGORIE != "_base/" ] && [ $CATEGORIE != "_script/" ]; then
```

```
            if [ -f $CATEGORIE$FILENAME ]; then
```

```
                echo -n " - ${CATEGORIE%/*}"
```

```
                CATAGORIE_STRING+="[[Category:${CATEGORIE%/*}]]";
```

```
            fi
```

```
        fi
```

```
    done
```

```
    echo $CATAGORIE_STRING
```

```
    #Ermittle den Dateityp
```

```

EXTENSIONTYPE=$(file --mime-type _base/$FILENAME)
echo EXTENSIONTYPE: $EXTENSIONTYPE
#Entscheide was mit der Datei passieren soll -> pythonscripts aufrufen

#pdf muss Upload und Einbindung erfolgen
#plaintext muss pandoc zur konvertierung genutzt und dann Eingebunden erfolgen
#docx dateien muss pandoc zur konvertierung genutzt und dann Eingebunden erfolg
en
#
-
> Für Zukünftige Versionen Bilder hinzufügen durch Upload und Tabellen fixen
#csv muss manuell convertirt werden und dann Eingebunden werden(Problem: CSV ha
t Dateityp plain)

if [ "$EXTENSIONTYPE" = "_base/$FILENAME: application/pdf" ]; then
    echo -n "loading to MediaWiki ... "
    python3 _script/upload_PDF.py $FILENAME $CATAGORIE_STRING
    echo -e "\033[32mdone\033[0m"
elif [ "$EXTENSIONTYPE" = "_base/$FILENAME: text/plain" ]; then
    sudo docker run --rm --volume "`pwd`::/data" --user `id -u`:`id -
g` pandoc/core -t mediawiki -i $FILE -o ${FILENAME%.*}.txt
    if [ ${FILENAME##*.} = "csv" ]; then
        CATAGORIE_STRING+="[[Category:CSV]];"
    fi
    python3 _script/upload_txt.py ${FILENAME%.*}.txt $CATAGORIE_STRING
    rm ${FILENAME%.*}.txt
    echo -e "\033[32mdone\033[0m"
elif [ "$EXTENSIONTYPE" = "_base/$FILENAME: application/vnd.openxmlformats-
officedocument.wordprocessingml.document" ]; then
    sudo docker run --rm --volume "`pwd`::/data" --user `id -u`:`id -
g` pandoc/core -t mediawiki -i $FILE -o ${FILENAME%.*}.txt
    python3 _script/upload_txt.py ${FILENAME%.*}.txt $CATAGORIE_STRING
    rm ${FILENAME%.*}.txt
    echo -e "\033[32mdone\033[0m"
else
    echo -e "\033[31mNicht unterstützter Datentyp\033[0m"
fi
echo -----
done

```

upload_txt.py

```

import sys
import media_wiki
import bot_info

def main(args = None):
    print(args)
    filename = args[1]
    try:
        category = args[2].replace(";", "\n")
    except:

```



```

        category = ""
        title = filename.split(".")[0]
        language = "de"

        #connectToDatabase
        session_item = media_wiki.connect(media_wiki.getURL(language), bot_info.getInfo
    ())
        media_wiki.overwriteWikiPage(session_item[0], session_item[1], media_wiki.getUR
L(language), title, category + media_wiki.tableLayout(readFile(filename, category))
    )

def readFile(filename, category):
    if "CSV" in category:
        print("CSV -----
- !!!!")
        return media_wiki.convertCSV(open(filename, "r"))
    else:
        return open(filename, "r").read()

if __name__ == '__main__':
    #This script needs a argument to run
    # 1 --- 'categories in MediaWiki format'
    main(sys.argv)

```

upload_pdf.py

```

import sys
import media_wiki
import bot_info

def main(args = None):
    print(args)
    filename = args[1]
    try:
        category = args[2].replace(";", "\n")
    except:
        category = ""
    title = filename.split(".")[0]
    language = "de"

    #connectToDatabase
    session_item = media_wiki.connect(media_wiki.getURL(language), bot_info.getInfo
    ())
    filename = media_wiki.uploadFile(
        session_item[0], session_item[1], media_wiki.getURL(language), filename)
    #write to Page
    media_wiki.overwriteWikiPage(
        session_item[0], session_item[1], media_wiki.getURL(language), title,
        category + "<pdf>Datei:" + filename + "</pdf>")

if __name__ == '__main__':

```

```

#This script needs a argument to run
# 1 --- 'Filename of File to read'
# 2 --- 'categories in MediaWiki format'
main(sys.argv)

```

media_wiki.py

```

import requests
import random
import string

def getURL(language):
    return "https://" + language + ".wiki.server.azo/api.php"

def connect(mediawiki_url, login_Info):
    session = requests.Session()
    params_1 = {
        "action": "query",
        "meta": "tokens",
        "type": "login",
        "format": "json"
    }

    response = session.get(url=mediawiki_url, params=params_1, verify=False)
    print(response)
    if response.status_code != 200:
        return None
    data = response.json()
    print(data)
    login_token = data["query"]["tokens"]["logintoken"]

    # Step 2: Send a POST request to login. Use of main account for login is not
    # supported. Obtain credentials via Special:BotPasswords
    # (https://www.mediawiki.org/wiki/Special:BotPasswords) for lgname & lgpassword
    try:
        params_2 = {
            "action": "login",
            "lgname": login_Info[0],
            "lgpassword": login_Info[1],
            "format": "json",
            "lgtoken": login_token
        }

        response = session.post(mediawiki_url, data=params_2)
    except Exception as e:
        print(str(e) + " --- 1")
        return None
    try:
        params_3 = {
            "action": "query",
            "meta": "tokens",

```

```

        "format": "json"
    }

    response = session.get(url=mediawiki_url, params=params_3)
    DATA = response.json()

    csrf_token = DATA["query"]["tokens"]["csrftoken"]
except Exception as e:
    print(str(e) + "    ---    2")
    return None

return [session, csrf_token]

def overwriteWikiPage(session, csrf_token, mediawiki_url, pSiteTitle, pText):
    PARAMS = {
        "action": "edit",
        "format": "json",
        "title": pSiteTitle,
        "text": pText,
        "token": csrf_token
    }
    response = session.post(mediawiki_url, data=PARAMS)
    print(response.json())

def uploadFile(session, csrf_token, mediawiki_url, filename):
    return uploadFileRec(session, csrf_token, mediawiki_url, filename, filename)

def uploadFileRec(session, csrf_token, mediawiki_url, filename, file_safe):
    PARAMS = {
        "action": "upload",
        "filename": filename,
        "format": "json",
        "token": csrf_token,
    }
    file = {'file':(file_safe, open("_base/" + file_safe, 'rb'), 'multipart/form-
data')}
    print("file    " + str(file))
    response = session.post(mediawiki_url, files=file, data=PARAMS)
    print(response.json())

    if response.json()["upload"]["result"] != "Success":
        print("    --- " + str(response.json()["upload"]["warnings"]))
        if "duplicate-archive" in str(response.json()["upload"]["warnings"]):
            return str(response.json()["upload"]["warnings"]["exists"])
        elif "duplicate" in str(response.json()["upload"]["warnings"]):
            #This File is already uploadet to the Server,
            #maybe under an other name. It will be used
            return str(response.json()["upload"]["warnings"]["dublicate"])
        elif "exists" in str(response.json()["upload"]["warnings"]):
            #This Filename is already used -> generating a new one

```

```

        return uploadFileRec(session, csrf_token, mediawiki_url, filenameGenerator(), file_safe)

    return filename

def filenameGenerator(size=6, chars=string.ascii_uppercase + string.digits):
    return ''.join(random.choice(chars) for _ in range(size)) + ".pdf"

def tableLayout(text):
    layout_string = '''{| class="wikitable"\n'''
    return text.replace("{|", layout_string)

def convertCSV(csv_file):
    text = ""
    lines = csv_file.readlines()
    print(lines)
    for line in lines:
        line = line.replace(";", "\n|")
        text += line
    return text

```

Anwenderschnittstelle (media_wiki.py)

```

def getURL(language):
    Anhand des Übergabeparameters language wird die URL des MediaWiki Servers
    übergeben. language muss dabei „en“, „de“ oder „az“ sein.

def connect(mediawiki_url, login_Info):
    meldet sich mit der mitgegebenen login_Info bei der MediaWiki-Plattform über die
    URL in mediawiki_url an. Gibt einen SessionKey und einen CSRF-Token zurück.

def overwriteWikiPage(session, csrf_token, mediawiki_url, pSiteTitle, pText):
    verbindet sich mit der MediaWiki Seite über die mediawiki_url und schreibt auf die
    Seite mit dem Titel in pSiteTitle den String pText.

def uploadFile(session, csrf_token, mediawiki_url, filename):
    lädt die Datei am Speicherort filename in die Datenbank des Wikis mit der URL
    mediawiki_url. Existiert diese Datei Bereits so wird der Name dieser
    zurückgegeben. Ist der Dateiname bereits vergeben so wird ein zufälliger String als
    Dateiname verwendet und zurückgegeben. Im Default-Case gibt die Funktion den
    normalen filename zurück.

def tableLayout(text):
    der text wird auf Tabellen im MediaWiki-Format abgesucht und an diese Tabellen
    eine Klasse angehängt. Der neue String wird zurückgegeben.

def convertCSV(csv_file):
    eine CSV-Datei csv_file wird in eine MediaWiki Tabelle konvertiert. Die MediaWiki
    Tabelle wird zurückgegeben.

```

Zusammenfassung und Ausblick

ACAS ist eine sehr große Software, die alles umfassen soll, was bei AZO zwischen Kunde und Anlage steht. MediaWiki ist ein noch nicht fertig implementierter Teil von ACAS und braucht noch etwas Arbeit. Mein Programm ist ein Prototyp, welcher mit der Zeit noch erweitert und verbessert werden wird. Es hat gezeigt, dass der Import von Textdateien und PDFs möglich ist und liefert eine erste Lösung für diese Aufgabe. Die Ordnerstruktur, die zur Kategorisierung genutzt wird, ist implementiert und ein erster Ansatz, um die MediaWiki Plattform nutzbar zu machen.

Es gibt noch einiges an diesem Projekt was zu überarbeiten ist. Die Verifikation auf dem Server sollte so gebaut werden, dass das Programm keine Warnungen ausgibt und die Anmeldeinformationen sollten verschlüsselt und sicher abgespeichert werden.

Mein Programm ist in vielen Bereichen noch erweiterbar. Beispielsweise ist denkbar die Anwendung in einem Dockercontainer zu starten und sie so in die Serverumgebung von ACAS zu integrieren. Außerdem könnte eine graphische Oberfläche zur Anwendung erstellt werden, welche es Nutzern ermöglicht eigene Dateien zu importieren. Diese könnte integriert im Wiki als Spezialseite aufrufbar sein, was es ermöglichen könnte Textinhalte von Kunden zu importieren.

Literaturverzeichnis

aws.amazon.com. (31. 08 2021). Von *aws.amazon.com/de/docker/*:
<https://aws.amazon.com/de/docker/> abgerufen

Red Hat. (30. 08 2021). Von *redhat.com*:

https://www.redhat.com/de/topics/containers/what-is-docker?sc_cid=7013a000002wLw0AAE&gclid=EAlaIQobChMI8sGQqq_Y8gIVje3tCh0ODQMFEAAAYASAAEgIlgSvD_BwE&gclsrc=aw.ds abgerufen

Abbildungsverzeichnis

| | |
|----------------------------------------------------------------------------|----|
| Abbildung 1: Realisierung einer Ampelschaltung an einem Steckschrank | 6 |
| Abbildung 2: Beispiel für SPS Programmiersprachen..... | 7 |
| Abbildung 3: VM und Docker im Vergleich..... | 12 |
| Abbildung 4: Ordnerstruktur..... | 14 |
| Abbildung 5: Serverumgebung der ACAS Plattform | 15 |
| Abbildung 6: Programmaufbau | 19 |