Konzeption und Implementierung einer generischen Crawler-Management-Plattform auf Basis bestehender Workflow-Orchestrierungstools

MASTER THESIS

Florian Oberndörfer

Eingereicht am 30. März 2026

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open Source Software

Betreuer:
Georg Schwarz, M. Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

Erlangen, 30. März 2026

Lizenz

_____

Erlangen, 30. März 2026

Abstract

Artificial Intelligence (AI) is one of the central technological topics of our time. The quality of AI models strongly depends on the quality and timeliness of the underlying data. Insufficient or outdated data directly leads to poorer results. Regular and reliable data provision is therefore essential for the sustainable use of AI systems.
The JValue project uses specialized web crawlers to provide structured data sets for AI systems. As the number and complexity of crawlers increase and their runtime lengthens, the complexity of crawler orchestration increases significantly. Existing workflow and orchestration frameworks offer functionalities for executing and orchestrating workflows, but no domain-specific functionalities such as crawler-specific metrics and analyses.
The aim of the work is to design a web-based crawler management system in the context of the JValue project. The system should be based on an existing orchestration framework and extend it with domain-specific functionalities without being closely coupled to the underlying orchestration framework. The system enables structured orchestration of distributed crawlers and provides domain-specific metrics and evaluations.
The results of this thesis show how a crawler management system can provide an up-to-date and reliable database, which is a key prerequisite for high-quality AI systems. Although the system is being developed for the JValue project, it is generic and can therefore also be used in other scenarios for the reliable provision of data by crawlers.

Zusammenfassung

Künstliche Intelligenz (KI) stellt eines der zentralen Technologiethemen der heutigen Zeit dar. Die Qualität von KI-Modellen hängt dabei jedoch stark von der Qualität und auch der Aktualität der zugrundeliegenden Daten ab. Unzureichende oder veraltete Daten führen unmittelbar zu schlechteren Ergebnissen. Eine regelmäßige und zuverlässige Datenbereitstellung ist daher von essenzieller Bedeutung für den nachhaltigen Einsatz von KI-Systemen.

Im Rahmen des JValue-Projekts werden spezialisierte Web-Crawler genutzt, um strukturierte Datensätze für KI-Systeme bereitzustellen. Mit steigender Anzahl, Komplexität sowie längerer Laufzeit der Crawler erhöht sich die Komplexität bei der Orchestrierung der Crawler maßgeblich. Bestehende Workflow- und Orchestrierungs-Frameworks bieten zwar Funktionalitäten zur Ausführung und Orchestrierung von Workflows, jedoch keinerlei domänenspezifische Funktionalitäten, wie crawler-spezifische Metriken und Datenanalysen.

Ziel dieser Arbeit ist die Konzeption und Entwicklung eines webbasierten Crawler-Management-Systems im Kontext des JValue-Projekts. Das System soll dabei auf einem bestehenden Orchestrierungs-Framework aufbauen und dieses um domänenspezifische Funktionalitäten erweitern, ohne dabei eng an das zugrundeliegende Orchestrierungs-Framework gekoppelt zu sein. Es ermöglicht eine strukturierte Orchestrierung verteilter Crawler und liefert dabei detaillierte domänenspezifische Metriken und Auswertungen.

Die Ergebnisse dieser Arbeit zeigen, wie durch ein Crawler-Management-System eine aktuelle und zuverlässige Datenbasis bereitgestellt werden kann, was eine zentrale Voraussetzung für qualitativ hochwertige KI-Systeme darstellt. Obwohl das System für das JValue-Projekt entwickelt wird, ist es generisch konzipiert und kann somit auch in anderen Szenarien zur zuverlässigen Bereitstellung von Daten durch Crawler eingesetzt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

A

AI ..... *Artificial Intelligence*
API ..... *Application Programming Interface*

C

CI/CD ..... *Continious Integration/Continous Delivery*
CRD..... *Custom Resource Definition*

D

DAG..... *Directed Acyclic Graph*

F

FTP..... *File Transfer Protocol*

H

HTTP ..... *Hypertext Transfer Protocol*

J

JWT ..... *Json Web Token*

K

KI ..... *Künstliche Intelligenz*

M

MCP ..... *Model Context Protocol*
MVP..... *Minimal Viable Product*

R

REST..... *Representational State Transfer*

U

UI ..... *User Interface*

X

XSS..... *Cross Site Scripting*

1 Einleitung

Künstliche Intelligenz entwickelte sich in den letzten Jahren zu einem der zentralen Technologiethemen. So sind Anwendungen auf Basis von maschinellem Lernen in beinahe allen Bereichen, darunter Medizin, Wirtschaft und Wissenschaft, präsent. Dabei wird gleichzeitig zunehmend deutlicher, dass der Erfolg solcher Systeme nicht allein von der Qualität der verwendeten Modelle und Algorithmen abhängt, sondern auch maßgeblich von der Qualität und Aktualität der zugrundeliegenden Daten beeinflusst wird.

Parmar et al. (2023) geht in seiner Arbeit noch einen Schritt weiter und argumentiert, dass die Qualität der zugrundeliegenden Daten eines KI-Systems einen größeren Einfluss auf dessen Qualität hat, als die Qualität des verwendeten Modells selbst.

So kann selbst ein perfekt trainiertes Modell mit einer theoretisch unendlich großen Menge an zur Verfügung stehenden, jedoch veralteten, Trainingsdaten, eine geringere Genauigkeit haben, als dasselbe Modell**,** trainiert mit einer begrenzten, aber dafür aktuellen Menge an Datensätzen (Valavi, Hestness, Ardalani, & Iansiti, 2021).

Veraltete, unvollständige oder inkonsistente Daten führen unmittelbar zu falschen und veralteten Ergebnissen der KI-Systeme. Insbesondere in Domänen mit sich schnell ändernden Daten, wie beispielsweise in der Wirtschaft, ist daher eine zuverlässige und regelmäßige Aktualisierung der Datensätze unerlässlich. Für die Erhebung dieser Daten aus dem Internet werden häufig sogenannte Web-Crawler eingesetzt, die es ermöglichen große Mengen an Daten automatisiert von verschiedenen Quellen zu sammeln und für KI-Systeme entsprechend aufzubereiten.

Mit zunehmender Anzahl und Komplexität solcher Crawler, sowie wachsenden Datenmengen steigen auch die Herausforderungen in Bezug auf die Steuerung und Überwachung der Crawler.

Sie müssen dabei regelmäßig und zuverlässig ausgeführt, überwacht und im Fehlerfall analysiert werden. Zudem ist es wichtig, dass Ausführungen zeitlich geplant, sowie Abhängigkeiten zwischen verschiedenen Verarbeitungsschritten berücksichtigt werden. Daher fordern insbesondere Szenarien mit vielen Crawlern eine strukturierte Orchestrierung.

Zur Orchestrierung von Workflows existieren bereits etablierte Orchestrierungsplattformen. Diese bieten zwar umfangreiche Funktionen in Bezug auf Planung, Ausführung und Überwachung komplexer Workflows an, sind dabei jedoch generisch konzipiert und verfügen daher über keinerlei domänenspezifische Funktionen für Web-Crawler. Es fehlen insbesondere spezifische Metriken und Visualisierungen für die Bewertung der Effektivität und Effizienz der Crawler.

Im Rahmen des JValue-Biomed-Projekts werden spezialisierte Web-Crawler entwickelt, um strukturierte Datensätze aus verschiedenen Quellen und Datenformaten zu sammeln und für KI-Systeme im biomedizinischen Bereich bereitzustellen**.**

Genau hier setzt das, im Rahmen dieser Arbeit zu entwickelnde, Crawler-Management-System an. Ziel ist es daher, eine webbasierte generische Crawler-Management-Plattform

auf Basis eines bestehenden Workflow-Orchestrierungstools zu konzipieren und zu entwickeln.
Das System soll dabei zwar auf einem bestehenden Orchestrierungs-Framework aufbauen und dieses um crawler-spezifische Funktionalitäten erweitern, jedoch möglichst lose an dieses gekoppelt sein, um ein Vendor-Lock-in zu vermeiden.
Um das zu entwickelnde Crawler-Management-System auch in anderen Anwendungsszenarien zur Orchestrierung und Analyse von Web-Crawlern einsetzen zu können, soll dieses explizit generisch entworfen werden.
Zu Beginn werden hierfür Anforderungen an ein zugrundeliegendes Orchestrierungs-Framework gestellt und anschließend auf Basis dieser verschiedene Frameworks verglichen.
Nach der Entscheidung für ein Orchestrierungs-Framework werden Anforderungen an das zu entwickelnde Crawler-Management-System gestellt. Auf dieser Grundlage erfolgt anschließend der Entwurf einer Systemarchitektur.
Es folgt ein Kapitel über das Frontend-Design der Anwendung, sowie ein Implementierungskapitel, in dem besondere Aspekte der Implementierung darrgestellt werden.
Abschließend werden die erreichten bzw. nicht erreichten Anforderungen, sowie die Auswahl des Orchestrierungs-Frameworks evaluiert, und zudem mögliche Weiterentwicklungsmöglichkeiten des Systems sowie des Gesamtprojekts vorgestellt.

2 Recherche

Für den Betrieb und die Steuerung der Crawler lassen sich zwei abgrenzbare Aufgabenbereiche definieren:

Zum einen die **Orchestrierung und Steuerung der Crawler-Ausführungen** und zum anderen die **Auswertung und Visualisierung von crawler-spezifischen Metriken und Statistiken**.

Ein Orchestrierungs-Framework adressiert hierbei ausschließlich den ersten Aufgabenbereich und stellt dabei grundlegende Funktionen zur Definition, zeitlichen Planung und Überwachung von Workflows bereit. Diese Funktionen sind generisch ausgelegt und vollständig unabhängig von der Art der auszuführenden Workflows. So sind Orchestrierungs-Frameworks nicht explizit auf Crawler ausgerichtet und bieten daher ausschließlich technische Informationen und Statistiken und keine spezifischen Statistiken oder Visualisierungen für Crawler.

Crawler stellen jedoch eine spezifische Klasse von Workflows dar, die spezielle Anforderungen an Statistiken und Auswertungen haben. Dazu zählen beispielsweise Informationen zur Anzahl entdeckter Datensätze, sowie der Anteil an neuen Datensätzen daraus. Diese Anforderungen stellen den zweiten Aufgabenbereich dar, welchen Orchestrierungs-Frameworks aufgrund ihres generischen Ansatzes nicht bieten können.

Aus diesem Grund ist die Entwicklung einer separaten Anwendung zur Anzeige und Auswertung crawler-spezifischer Informationen notwendig.

Die folgende Evaluation von Crawler-Orchestrierungs-Frameworks verfolgt somit nicht das Ziel ein Orchestrierungs-Framework zu finden, welches beide Aufgabenbereiche abdeckt. Viel mehr liegt der Fokus ausschließlich auf der Bewertung der Frameworks hinsichtlich der Eignung zur Orchestrierung von Crawler-Ausführungen.

Die Orchestrierung von Webcrawlern sowie deren einzelne Verarbeitungsschritte, stellt eine technisch anspruchsvolle und umfangreiche Aufgabe dar, da Aspekte wie Zuverlässigkeit, Skalierbarkeit und Wartbarkeit zu berücksichtigen sind.

Es existieren jedoch bereits zahlreiche Tools und Frameworks, die zentrale Aspekte dieser Orchestrierung übernehmen, und dadurch den notwendigen Entwicklungsaufwand drastisch reduzieren.

Ziel dieses Kapitels ist die systematische Definition von Anforderungen an ein Orchestrierungs-Framework für das **JValue-Biomed-Projekt**, die Vorauswahl von infrage kommenden Frameworks, sowie die finale Gegenüberstellung und Auswahl eines Orchestrierungs-Frameworks.

2.1 Anforderungen an die Crawler-Orchestrierung

Die Anforderungen werden anhand ihrer Relevanz für das Projekt in vier Prioritätsklassen (A-D) unterteilt, um eine strukturierte Bewertung infrage kommender Frameworks zu ermöglichen.

- **Priorität A (Muss-Kriterien):**
  Anforderungen dieser Priorität sind zwingend erforderlich. Frameworks, die eines oder mehrere dieser Kriterien nicht erfüllen, werden ausgeschlossen.
- **Priorität B (Soll-Kriterien):**
  Diese Anforderungen sind von hoher Bedeutung. Ihr Fehlen führt nicht zwangsläufig zum Ausschluss, wirkt sich jedoch stark negativ auf die Bewertung des Frameworks aus.
- **Priorität C (Kann-Kriterien):**
  Anforderungen dieser Priorität stellen wünschenswerte Funktionen dar. Ihre Erfüllung wirkt sich stark positiv auf die Bewertung des Frameworks aus.
- **Priorität D (Optionale Kriterien):**
  Diese Anforderungen stellen weiterführende Funktionen dar, die den Funktionsumfang und die Nutzbarkeit erweitern, jedoch nur von geringer Relevanz sind.

Anforderungen Priorität A (Muss-Kriterien):

A1: Das Framework muss unter einer **Open-Source-Lizenz** veröffentlicht sein, welche der Open-Source-Definition (Open Source Initiative, 2007) entspricht, sodass keinerlei Lizenzkosten entstehen und zudem eine langfristige Nutzung gewährleistet ist.

A2: Das Framework muss Teil einer **etablierten Software-Foundation** oder eines **langfristig gepflegten Projekts** sein, um Weiterentwicklung und Stabilität zu sichern.

A3: Das Framework muss kompatibel mit **Kubernetes** sein, um in die bestehende Kubernetes-Umgebung deployt werden zu können.

A4: Das Orchestrierungs-Framework muss über Möglichkeiten zur **Authentifizierung** verfügen, um unautorisierten Zugriff zu verhindern.

A5: Das Framework muss die Definition **wiederverwendbarer Workflow-Schablonen** unterstützen, um manuellen Konfigurationsaufwand zu minimieren.

A6: Es muss eine Möglichkeit zur **zeitgesteuerten, regelmäßigen Ausführung** von Workflow-Schablonen (im Folgenden auch Schedules genannt) geben, um Workflow-Schablonen automatisiert ausführen lassen zu können.

A7: Unterschiedliche Parametrisierungen für dieselbe Workflow-Schablone, auch in Kombination mit verschiedenen Schedules, müssen unterstützt werden, um doppelten Konfigurationsaufwand zu verhindern

A8: Das System muss hohe **Zuverlässigkeit und Stabilität** durch Replikation oder einen High Availability Modus bieten.

A9: Die **Versionierung von Workflow-Schablonen** zur Nachverfolgung der Schablonen muss unterstützt werden.
A10: Änderungen an Workflow-Schablonen dürfen **kein erneutes Deployment** des Systems erfordern, um den manuellen Aufwand und das erhöhte Risiko durch häufiges Deployment gering zu halten.
A11: Die Entwicklung der Crawler muss **weitgehend unabhängig vom Orchestrierungs-Framework** erfolgen können, abgesehen von grundlegenden strukturellen Vorgaben und benötigten Schnittstellen.
A12: Disaster Recovery Mechanismen für den Neustart nach Systemabstürzen mit Wiederherstellung des Zustands ohne manuelles Eingreifen müssen im System integriert sein.

Anforderungen Priorität B (Soll-Kriterien):
B1: Das Orchestrierungs-Framework soll eine **grafische Benutzeroberfläche** mit folgenden Funktionen anbieten:

a) Manuelles Starten von Workflows
b) Definition zeitgesteuerter Ausführungen (Schedules)
c) Konfiguration unterschiedlicher Parameter pro Schedule und Workflow-Schablone
d) Anzeige aller Workflow-Ausführungen pro Schedule

B2: Das Framework soll eine, für **nicht-technische Nutzer**, **einfach verständliche und bedienbare Oberfläche** anbieten.
B3: Das System soll ein **Nutzermanagement** mit mindestens einer Differenzierung zwischen **Lese- und Schreibrechten** anbieten, um die Rechte bestimmter Nutzer einschränken zu können.
B4: Konfigurierbare **Retry-Mechanismen** zur Verbesserung der Fehler-Resilienz sollen im Framework definierbar sein.
B5: Es soll eine **Statistik** zur Anzahl **erfolgreicher und fehlgeschlagener Ausführungen pro Schedule** angeboten werden, um Schedules mit erhöhter Fehlerrate identifizieren zu können.
B6: Durch den Betrieb des Frameworks soll unabhängig von der Ausführung der Crawler ein möglichst geringer zusätzlicher **Performance-Overhead** entstehen (bewertet anhand der Anzahl der Systemkomponenten).

Anforderungen Priorität C (Kann-Kriterien):
C1: Eine Möglichkeit zur **Priorisierung von Workflow-Ausführungen** pro Schedule und Parametrisierung kann angeboten werden, um die Ausführung wichtiger Schedules sicherzustellen.
C2: Eine einfache Anzeige von **Crawler-Logs** für die Erstanalyse kann angeboten werden.
C3: Eine Möglichkeit zur **Alarmierung** von Nutzern bei fehlgeschlagenen Workflow-Ausführungen kann unterstützt werden.

C4: Die **Integration von eigenen Design-Komponenten** für Crawler-Statistiken kann unterstützt werden.
C5: Die Möglichkeit zur **Pausierung** von Workflow-Ausführungen zwischen einzelnen Verarbeitungsschritten kann angeboten werden.
C6: Eine **feingranulare Rechteverwaltung** kann bereitgestellt werden, um Nutzer auf die Funktionen einzuschränken, für die sie befugt sind.
C7: Ereignisbasierte Alarmierungen (z.B. bei überfüllter Warteschlange), um Nutzer vor potenziellen Fehlern zu warnen, können möglich sein.
C8: Eine Anzeige der **genutzten Ressourcen** von Workflow-Ausführungen zur Erstanalyse kann angeboten werden.

Anforderungen Priorität D (Optionale Kriterien):
D1: Eine **Visualisierung der Ressourcenauslastung** der, für Crawler definierten, Infrastruktur wäre nützlich, um mögliche Ressourcenknappheit frühzeitig zu erkennen.
D2: Die Unterstützung von **Plugins** würde den möglichen Funktionsumfang erweitern.

2.2 Evaluation der Orchestrierungs-Frameworks

Anhand der zuvor definierten Anforderungen werden im Folgenden die bekanntesten und für den gegebenen Kontext relevantesten Orchestrierungs-Frameworks verglichen.
Apache Airflow ist eine etablierte Open-Source-Plattform zur Entwicklung, Planung und Überwachung von batch-orientierten Workflow-Pipelines. Workflows werden als gerichtete azyklische Graphen (DAG) in Python definiert. (Apache Airflow, 2025 - a)
Prefect ist ein modernes Workflow-Orchestrierungs-Framework, das speziell für Python-native Orchestrierung entwickelt wurde. Workflows werden in Python definiert und zur Laufzeit dynamisch ausgeführt, was Kontrollflussentscheidungen zur Laufzeit ermöglicht. (Prefect, o. D. - b)
Argo Workflows ist ein container-natives Workflow-Framework für Kubernetes. Workflows werden deklarativ im YAML-Format und als Custom Resource Definitions (CRD) direkt im Kubernetes Cluster ausgeführt, wobei jeder Schritt eines Workflows typischerweise als separater Container in einem eigenen Pod läuft. (Argo Workflows, o. D.)
Für die Evaluation wurden die Dokumentationen der jeweiligen Frameworks sowie Begleitartikel genutzt. Zusätzlich folgte ein manueller Test alle drei Frameworks, um diejenigen Anforderungen zu evaluieren, welche nicht zweifelsfrei anhand der Dokumentation geklärt werden konnten.

Für die Testung wurden die folgenden Versionen verwendet:

- Apache Airflow: 3.1.0 [1], Dokumentation: (Apache Airflow, 2025 - a)
- Prefect: 3.5.0 [2], Dokumentation: (Prefect, o. D. - b)
- Argo Workflows: 3.7.3 [3], Dokumentation: (Argo Workflows, o. D.)

Anforderungen Priorität A

A1: Open-Source-Lizenz

Apache Airflow ist unter der Apache License 2.0 veröffentlicht und erfüllt damit das Kriterium einer kostenfreien Open-Source-Lizenz. (Apache Airflow, 2025 - b)

Prefect ist sowohl in einer Open-Source-Variante (Apache License 2.0) als auch als kommerzielle Cloud-Anwendung verfügbar. Da dieses Kriterium explizit eine Open-Source-Lizenz fordert, wird im Rahmen dieser Evaluation ausschließlich die Open-Source-Variante von Prefect betrachtet. (Prefect, 2022)

Argo Workflows ist ebenfalls unter der Apache License 2.0 veröffentlicht und erfüllt damit das Kriterium. (Argo Workflows, 2018)

Alle drei Frameworks erfüllen die Anforderung A1.

A2: Etablierte Software-Foundation oder langfristig gepflegtes Projekt

Apache Airflow ist Teil der Apache Software Foundation (Apache Software Foundation, kein Datum), welche langfristige Wartung sowie eine breite Community sicherstellt. Zudem wird Airflow von zahlreichen großen Unternehmen produktiv eingesetzt. (Chaurasia, 2026)

Prefect ist kein Projekt einer Software Foundation, sondern wird primär von einem kommerziellen Unternehmen entwickelt. Zwar ist der Quellcode Open-Source gestellt und namhafte Konzerne und Institutionen, wie NASA, Meta und Cisco (Prefect, o. D. - a) verwenden Prefect, jedoch wird hierbei nicht kenntlich gemacht, ob diese die Open-Source-Version verwenden. Daher ist die langfristige Wartung nicht abgesichert und hängt maßgeblich von den strategischen Entscheidungen des Unternehmens ab.

Argo Workflows ist ein Projekt der Cloud Native Computing Foundation, zu der unter anderem auch Kubernetes und Helm gehören. Die CNCF sichert die langfristige Wartung. (Cloud Native Computing Foundation, o. D.)

Airflow und Argo erfüllen die Anforderung A2 uneingeschränkt, Prefect nicht.

A3: Kubernetes Kompatibilität

Apache Airflow bietet ein offizielles Helm Chart für Kubernetes an und unterstützt spezifische Ausführungsmechanismen über Kubernetes Executors und Operators.

Prefect ist mit Kubernetes kompatibel und bietet dafür spezifische Worker-Modelle, sowie ebenfalls ein offizielles Helm Chart an.

---

[1] https://airflow.apache.org/docs/apache-airflow/stable/release_notes.html#airflow-3-1-0-2025-09-25
[2] https://docs.prefect.io/v3/release-notes/oss/version-3-5
[3] https://github.com/argoproj/argo-workflows/releases/tag/v3.7.3

Argo Workflows ist explizit für Kubernetes entwickelt und setzt daher Kubernetes zwingend voraus und bietet hierfür auch ein offizielles Helm Chart an.
Alle drei Frameworks erfüllen die Anforderung A3.

A4: Authentifizierung
Apache Airflow bietet ein umfangreiches Benutzer- und Rechtemanagement mit verschiedenen Authentifizierungsmöglichkeiten.
Prefect bietet eine einfache Authentifizierung über Basic Auth (Username und Passwort) an, welche sowohl für das User Interface (UI) als auch für das Application Programming Interface (API) genutzt wird. Jedoch kann hierbei nur ein Nutzer über Umgebungsvariablen beim Start gesetzt werden.
Argo Workflows nutzt das Kubernetes Rollen- und Rechtemodell (RBAC). Sowohl API als auch UI können über, auf Kubernetes basierende, Authentifizierung abgesichert werden.
Alle drei Frameworks erfüllen die Anforderung A4.

A5: Wiederverwendbare Workflow-Schablonen
Apache Airflow definiert Workflow-Schablonen als DAGs in Form von Python Skripten. Diese können parametrisiert und in anderen DAGs wiederverwendet werden.
Prefect definiert Workflows als Python-Funktionen, die explizit als wiederverwendbar und parametrisierbar konzipiert sind.
Argo Workflows bietet sogenannte Workflow Templates, welche parametrisiert und referenziert werden können.
Alle drei Frameworks erfüllen die Anforderung A5.

A6: Zeitgesteuerte Ausführung von Workflow-Schablonen
Apache Airflow unterstützt zeitgesteuerte Ausführungen über Schedules, die jedoch ausschließlich im DAG-Code definiert werden können.
Prefect bietet die Möglichkeit Schedules auf Workflow-Schablonen Ebene komfortabel über das UI zu konfigurieren.
Argo Workflows bietet sogenannte Cron Workflows, welche zeitgesteuerte Ausführungen definieren und dabei Workflow Templates referenzieren können.
Alle drei Frameworks erfüllen die Anforderung A6.

A7: Unterschiedliche Parametrisierungen pro Workflow-Schablone und Schedule
In der getesteten Version 3.1.0 ermöglicht **Apache Airflow** pro DAG lediglich eine Schedule und ein Parameterset. Für mehrere verschiedene Schedules bzw. unterschiedliche Parametrisierungen wird somit jeweils ein eigenes DAG pro Schedule und Parameterset benötigt.
Prefect erlaubt die Definition unterschiedlicher Parametersets je Schedule.

Argo Workflows erlaubt das Referenzieren von Workflow Templates in Cron Workflows, wodurch unterschiedliche Parameter je Schedule definiert werden können.
Prefect und Argo erfüllen die Anforderung A7 vollständig, Airflow nur eingeschränkt.

A8: Hohe Zuverlässigkeit und Stabilität
Alle drei Frameworks bieten hohe Zuverlässigkeit und Stabilität durch die Möglichkeit zur Replikation aller Komponenten und erfüllen damit die Anforderung A8 vollständig.

A9: Versionierung von Workflow-Schablonen
Bei **Apache Airflow** können DAGs in einem Git Repository verwaltet und über Continious Integration und Continious Delivery (CI/CD) Prozesse oder Synchronisationsmechanismen in die Airflow-Umgebung deployt werden.
Auch bei **Prefect** wird der Workflow-Code typischerweise in Git Repositories verwaltet.
Argo Workflows hingegen speichert Workflow Templates als Kubernetes-CRDs im Cluster. Eine Git-basierte Versionierung ist jedoch möglich, indem die CRDs in einem Git-Repository verwaltet und per CI/CD nach Kubernetes deployt werden, wo Argo sie schließlich erkennt. Die Funktion Workflow Templates über das UI zu ändern ist dadurch jedoch nutzlos, da diese Änderungen durch die nächste Synchronisation mit dem Repository überschrieben werden.
Alle drei Frameworks erfüllen die Anforderung A9.

A10: Kein erneutes Deployment des Systems bei Änderungen an Workflow-Schablonen
Apache Airflow erkennt neue DAGs automatisch, sobald diese im dafür konfigurierten Verzeichnis liegen.
Prefect registriert neue Workflows über sogenannte Deployments, ohne dass ein erneutes Deployment der Anwendung notwendig ist.
Argo Workflows erkennt neue Workflow Templates automatisch, sobald diese im Kubernetes Cluster als CRDs verfügbar sind.
Alle drei Frameworks erfüllen die Anforderung A10.

A11: Entwicklung der Crawler unabhängig vom Orchestrierungs-Framework
Bei allen drei Frameworks liegt der Fokus auf der Orchestrierung. Die eigentlichen Crawler werden weder im Framework implementiert noch ausgeführt, sondern lediglich als externe Prozesse bzw. Container orchestriert.
Alle drei Frameworks erfüllen die Anforderung A11.

A12: Disaster Recovery

Apache Airflow speichert den gesamten Systemzustand in einer Metadaten Datenbank, über welche der komplette Systemzustand nach einem Absturz automatisch rekonstruiert wird. (Astronomer, o. D. - b)

Prefect speichert ebenfalls den gesamten Systemzustand in einer zentralen Datenbank, anhand welcher nach Absturz automatisch der gesamte Systemzustand rekonstruiert wird.

Argo Workflows speichert den kompletten Zustand im Kubernetes etcd, anhand dessen nach Absturz der komplette Systemzustand rekonstruiert wird.

Alle drei Frameworks erfüllen die Anforderung A12 vollständig.

Anforderungen Priorität B

B1: Grafische Benutzeroberfläche für Scheduling und Parametrisierung

B1a: Manuelles Starten von Workflows

Alle drei Frameworks ermöglichen in der getesteten Version das manuelle Starten von Workflows über eine grafische Oberfläche und erfüllen somit die Anforderung B1a vollständig.

B1b: Definition zeitgesteuerter Ausführungen (Schedules)

Apache Airflow ermöglicht in der getesteten Version die Definition von Schedules ausschließlich im DAG-Code. Eine Konfiguration von Schedules über die grafische Oberfläche ist nicht möglich.

Schedules sind zwar theoretisch über globale Airflow-Variablen definierbar, jedoch wird die Nutzung von Airflow-Variablen zur Steuerung von DAGs in der offiziellen Dokumentation von Airflow explizit als Bad Practice bezeichnet, weil dies zu schlechter Wartbarkeit und geringer Transparenz führt, da die Airflow-Variablen lediglich global und somit für alle DAGs definiert werden können.

Prefect bietet in der getesteten Version die Möglichkeit, Schedules für Workflow-Schablonen komfortabel über die Benutzeroberfläche zu erstellen und zu verwalten.

Argo Workflows bietet die Möglichkeit zur Erstellung und Verwaltung von Cron Workflows (Schedules) komfortabel über die grafische Benutzeroberfläche.

Damit erfüllen Prefect und Argo die Anforderung B1b uneingeschränkt, Airflow hingegen nur sehr eingeschränkt.

B1c: Konfiguration unterschiedlicher Parameter pro Schedule und Workflow-Schablone

Apache Airflow ermöglicht die Übergabe von Parametern über die Benutzeroberfläche ausschließlich bei manuellen DAG-Runs. Für zeitgesteuerte Ausführungen existiert in der getesteten Version keine Möglichkeit unterschiedliche Parameter je Schedule zu definieren. Workarounds mittels der globalen Airflow-Variablen in Verbindung mit codebasierten

Auswertungen wären auch hierbei wieder möglich, gelten jedoch, wie bereits beschrieben, als Bad Practice.
Prefect erlaubt in der getesteten Version die Definition eines eigenen Parametersatzes pro Schedule über die Benutzeroberfläche.
Auch **Argo Workflows** ermöglicht es in der getesteten Version pro Cron Workflow (Schedule) einen eigenen Parametersatz für das referenzierte Workflow Template über die Benutzeroberfläche zu definieren.
Prefect und Argo erfüllen die Anforderung B1c vollständig, Airflow hingegen nur sehr eingeschränkt.

B1d: Anzeige aller Ausführungen pro Schedule
Apache Airflow bietet eine Detailseite pro DAG, die alle Runs auflistet. Da pro DAG eine feste Schedule definiert wird, entspricht dies der Anzeige aller Runs pro Schedule.
Prefect bietet eine Liste aller Ausführungen mit diversen Filtermöglichkeiten an.
Argo Workflows bietet in der getesteten Version eine Detailseite der Cron Workflows an, auf der alle Ausführungen des entsprechenden Cron Workflows und somit der Schedule angezeigt werden.
Alle drei Frameworks erfüllen die Anforderung B1d uneingeschränkt.

B2: Für nicht-technische Nutzer leicht bedienbare Oberfläche
Apache Airflow bietet in der getesteten Version eine funktional sehr umfangreiche Benutzeroberfläche, deren Bedienung jedoch aufgrund der Verwendung technischer Konzepte ein technisches Grundverständnis voraussetzt.
Prefect bietet in der getesteten Version eine moderne, übersichtliche und technisch stark abstrahierte Benutzeroberfläche, welche für die gängigen Funktionen ohne tiefgehende technische Kenntnisse sehr gut verständlich ist.
Argo Workflows Benutzeroberfläche in der getesteten Version ist für nicht-technische Nutzer schwer verständlich, da die meisten zentralen Bestandteile auf YAML-Definitionen basieren, was eine sinnvolle Bedienung der Oberfläche für nicht-technische Nutzer nahezu unmöglich macht.
Damit erfüllt Prefect die Anforderung B2 am besten, Apache Airflow eingeschränkt und Argo Workflows nicht.

B3: Nutzermanagement mit Differenzierung zwischen Lese- und Schreibrechten
Apache Airflow bietet eine integrierte Nutzerverwaltung mit vordefinierten Rollen und erfüllt dieses Kriterium damit vollständig.
Prefect ermöglicht es lediglich einen einzelnen Admin-Benutzer mit vollständigen Rechten anzulegen und bietet damit keine Möglichkeit zur Differenzierung zwischen Lese- und Schreibrechten.

Argo Workflows nutzt das Kubernetes Rollen- und Rechtemodell (RBAC) und bietet daher umfassende feingranulare Rechteverwaltung an.
Argo Workflows und Apache Airflow erfüllen die Anforderung B3 uneingeschränkt, Prefect hingegen nicht.

B4: Konfigurierbare Retry-Mechanismen
Alle drei Frameworks bieten umfassende Retry-Mechanismen auf Task- bzw. Step-Ebene und erfüllen damit die Anforderung B4 vollständig. (Astronomer, o. D. - a)

B5: Statistik zur Anzahl erfolgreicher und fehlgeschlagener Ausführungen pro Schedule
Apache Airflow bietet in der getesteten Version ausführliche Statistiken pro DAG. Da jeder DAG nur eine Schedule haben kann, entsprechen diese Statistiken den Statistiken pro Schedule.
Prefect bietet in der getesteten Version ebenfalls diese Funktionalität an.
Argo Workflows zeigt in der getesteten Version die Anzahl erfolgreicher und fehlgeschlagener Ausführungen pro Cron Workflow (Schedule) an.
Alle drei Frameworks erfüllen die Anforderung B5.

B6: Geringer Performance Overhead
Diese Anforderung wird vereinfacht anhand der Anzahl an laufenden Systemkomponenten evaluiert.
Apache Airflow besteht, je nach Konfiguration, aus vier bis sechs Systemkomponenten, welche konstant laufen. Dies führt dazu, dass selbst ohne aktive Workflow Runs bereits erhebliche Ressourcen belegt werden.
Prefect wiederum besteht in der getesteten Version aus einem API-Server, mindestens einem Worker, sowie einer Datenbank.
Argo Workflows benötigt lediglich einen Server und einen Workflow-Controller. Wenn Informationen zu abgeschlossenen Runs archiviert werden sollen, wird zudem noch eine Datenbank benötigt.
Argo Workflows erfüllt die Anforderung B6 vollständig, Prefect nur eingeschränkt, Apache Airflow nicht.

Anforderungen Priorität C

C1: Priorisierung von Workflow-Ausführungen pro Schedule und Parametrisierung
Apache Airflow unterstützt die Priorisierung von DAGs und Tasks über Prioritätsgewichte und bietet zudem die Möglichkeit explizite Prioritätsregeln zu definieren.
In **Prefect** lassen sich verschiedene Work Queues erstellen, denen verschiedene Prioritäten zugeordnet werden können. Pro Schedule wird anschließend definiert, in welcher Work

Queue ein Workflow ausgeführt werden soll, und somit auch, welche Priorität die Ausführung erhält.
Argo Workflows bietet zwei verschiedene Ebenen für die Priorisierung. Zum einen werden für die Priorisierung die Priority-Klassen von Kubernetes genutzt. Für jedes Workflow Template oder jeden Cron Workflow (Schedule) kann eine Kubernetes Priority-Klasse angegeben werden, welche das Scheduling der zugehörigen Pods direkt in Kubernetes beeinflusst (Kubernetes, o. D.). Zum anderen kann noch eine interne Prioritätszahl vergeben werden, welche beeinflusst, in welcher Reihenfolge Workflows zur Ausführung an Kubernetes gegeben werden, sollte ein Limit für die parallele Ausführung von Workflows definiert sein.
Alle drei Frameworks erfüllen die Anforderung C1 ohne Einschränkungen.

C2: Anzeige von Crawler-Logs
Alle drei Frameworks bieten einfache Log-Anzeigen und erfüllen damit die Anforderung C2.4.

C3: Alarmierungen bei fehlgeschlagenen Workflow-Ausführungen
Apache Airflow bietet Callback-Mechanismen zur Benachrichtigung bei bestimmten ausführungsbezogenen Ereignissen, z.B. über E-Mail oder Slack.
Prefect unterstützt ebenfalls Benachrichtigungen über verschiedene Kanäle und bietet dabei flexible Konfigurationsmöglichkeiten.
Argo Workflows ermöglicht die Definition von Exit Handlern in Workflow Templates, welche nach Abschluss eines Runs ausgeführt werden und dabei beliebige Aktionen, wie auch Benachrichtigungen auslösen können. Allerdings wird dafür stets ein eigener Pod gestartet.
Alle drei Frameworks erfüllen die Anforderung C3 vollständig.

C4: Integration von eigenen Design-Komponenten
Apache Airflow bietet seit Version 3.1 die Möglichkeit, die Benutzeroberfläche um eigene React-basierte Komponenten zu erweitern. Dieses Feature ist jedoch als experimentell gekennzeichnet, wodurch keine Stabilität garantiert ist.
Prefect und **Argo Workflows** bieten keine Möglichkeit zur Integration eigener UI-Komponenten.
Apache Airflow erfüllt die Anforderung C4 mit großen Einschränkungen, Prefect und Argo Workflows nicht.

C5: Pausieren von Workflow-Ausführungen zwischen einzelnen Schritten

Apache Airflow bietet in der getesteten Version keine Möglichkeit Workflow-Ausführungen zu pausieren.

Bei **Prefect** und **Argo Workflows** ist das Pausieren zwischen einzelnen Schritten des Workflows in der getesteten Version problemlos möglich.

Prefect und Argo Workflows erfüllen die Anforderung C5 uneingeschränkt, Apache Airflow nicht.

C6: Feingranulare Rechteverwaltung

Apache Airflow bietet ein umfassendes Rollenkonzept, das um eigene Rollen mit feingranularen Rechten erweitert werden kann

Prefect bietet keinerlei Rechteverwaltung.

Argo Workflows nutzt das Kubernetes Rollen- und Rechtemodell (RBAC) und bietet daher umfassende feingranulare Rechteverwaltung.

Apache Airflow und Prefect erfüllen die Anforderung C6 damit uneingeschränkt, Prefect nicht.

C7: Ereignisbasierte Alarmierungen

Apache Airflow bietet Alarmierungen lediglich auf Task- bzw. DAG-Ebene. (Rangoola, 2023)

Prefect bietet die Möglichkeit sogenannte Automations zu definieren. Hierfür können diverse Trigger und Actions definiert werden, wie beispielsweise auch Alarmierungen.

Argo Workflows selbst bietet keine ereignisbasierten Alarmierungen. Diese Funktionalität kann jedoch über Argo Events[5] erreicht werden, welches eng mit Argo Workflows zusammenarbeitet.

Prefect erfüllt die Anforderung C7 vollständig, Argo Workflows bietet eine Erweiterung aus dem Argo-Ökosystem dafür an, während Apache Airflow diese Anforderung nicht erfüllt.

C8: Anzeige genutzter Ressourcen einer Workflow Ausführung

Apache Airflow und Prefect liefern in den getesteten Versionen keinerlei Informationen zu genutzten Ressourcen von Workflow-Ausführungen.

Argo Workflows zeigt pro Workflow Ausführung eine Schätzung der genutzten Ressourcen anhand von angefragten Ressourcen und Limits an.

Argo Workflows erfüllt die Anforderung C8 eingeschränkt, Apache Airflow und Prefect nicht.

[5] https://argoproj.github.io/events/

Anforderungen Priorität D

D1: Visualisierung der Ressourcenauslastung

Keines der betrachteten Frameworks bietet in den getesteten Versionen eine Visualisierung der Cluster Ressourcenauslastung an.

D2: Plugins

Apache Airflow bietet ein umfassendes Plugin-System und ein großes Plugin-Ökosystem.
Prefect bietet die Integration von Plugins lediglich als experimentelles Feature an.
Argo Workflows bietet als Plugins bzw. Erweiterungen lediglich die Anwendungen aus dem eigenen Argo-Ökosystem an, sowie die Möglichkeit zur Integration von Executor Plugins, welche bei der Ausführung angewandt werden.
Lediglich Apache Airflow erfüllt die Anforderung D2 uneingeschränkt, Prefect und Argo Workflows nur eingeschränkt.

2.3 Auswertung der Orchestrierungs-Frameworks

Anforderung	Apache Airflow	Prefect	Argo Workflows
A1 (Open Source)	Ja	Ja	Ja
A2 (Wartung)	Ja	Nein	Ja
A3 (Kubernetes)	Ja	Ja	Ja
A4 (Authentifizierung)	Ja	Ja	Ja
A5 (Workflow-Schablonen)	Ja	Ja	Ja
A6 (Scheduling)	Ja	Ja	Ja
A7 (Parametrisierung)	Eingeschränkt	Ja	Ja
A9 (Zuverlässigkeit)	Ja	Ja	Ja
A10 (Versionierung)	Ja	Ja	Ja
A11 (Redeployment)	Ja	Ja	Ja
A12 (Entwicklungs-unabhängigkeit)	Ja	Ja	Ja
A13 (Recovery)	Ja	Ja	Ja
B1 (UI)	-	-	-
B1a (Manuelles Starten)	Ja	Ja	Ja
B1b (Schedules)	Eingeschränkt	Ja	Ja
B1c (Parametrisierung)	Eingeschränkt	Ja	Ja
B1d (Run-Übersicht)	Ja	Ja	Ja
B2 (Einfache UI)	Eingeschränkt	Ja	Nein
B3 (Einfaches Nutzermanagement)	Ja	Nein	Ja
B4 (Retry)	Ja	Ja	Ja
B5 (Statistiken)	Ja	Ja	Ja
B6 (Performance)	Nein	Eingeschränkt	Ja
C1 (Priorisierung)	Ja	Ja	Ja
C2 (Log-Anzeige)	Ja	Ja	Ja
C3 (Einfache Alarmierungen)	Ja	Ja	Ja
C4 (Eigene UI-Komponenten)	Eingeschränkt	Nein	Nein
C5 (Pausieren)	Nein	Ja	Ja

C6 (Feingranulare Rechteverwaltung)	Ja	Nein	Ja
C7 (Ereignisbasierte Alarmierungen)	Nein	Ja	Eingeschränkt
C8 (Ressourcennutzung-Anzeige)	Nein	Nein	Eingeschränkt
D1 (Auslastungs-Anzeige)	Nein	Nein	Nein
D2 (Plugins)	Ja	Eingeschränkt	Eingeschränkt

Tabelle 1: Evaluation Orchestrierungs-Frameworks

	Apache Airflow	**Prefect**	**Argo Workflows**
Nicht erfüllte A Anforderungen	0	1	0
Eingeschränkt erfüllte A Anforderungen	1	0	0
Nicht erfüllte B Anforderungen	1	1	1
Eingeschränkt erfüllte B Anforderungen	3	1	0
Nicht erfüllte C Anforderungen	3	3	1
Eingeschränkt erfüllte C Anforderungen	1	0	2
Nicht erfüllte D Anforderungen	1	1	1
Eingeschränkt erfüllte D Anforderungen	0	1	1

Tabelle 2: Zusammenfassung Evaluation Orchestrierungs-Frameworks

Zusammenfassung

In Bezug auf die Anforderungen der **Priorität A** (Muss-Kriterien) erfüllt **Argo Workflows** als einziges der betrachteten Frameworks alle Anforderungen uneingeschränkt.

Apache Airflow erfüllt die Muss-Kriterien ebenfalls weitgehend, weist jedoch Einschränkungen bei der Anforderung zur flexiblen Parametrisierung von Workflow-

Schablonen auf, da Parametrisierungen bei Apache Airflow ausschließlich codebasiert erfolgen, wodurch pro Parametrisierung eine separate DAG-Datei erforderlich ist.
Prefect erfüllt die Anforderungen der Priorität A größtenteils, abgesehen von der Anforderung der langfristig gesicherten Wartung. Obwohl Prefect ein etabliertes Projekt mit namhaften Kunden ist, ist die Open-Source-Variante nicht Teil einer Software-Foundation, und so wird diese Anforderung nicht erfüllt.

Bei den Anforderungen der **Priorität B** (Soll-Kriterien) erfüllt keines der drei Frameworks alle Anforderungen vollständig.
Apache Airflow erfüllt die Anforderung an einen geringen systembedingten Performance-Overhead nicht (B6). Zudem weist es erhebliche Einschränkungen in Bezug auf die Nutzbarkeit der grafischen Benutzeroberfläche auf. Die Anforderungen zur Definition von Schedules (B1b) sowie zur Parametrisierung über die Benutzeroberfläche (B1c) werden nur sehr eingeschränkt erfüllt, da entsprechende Funktionen ausschließlich codebasiert verfügbar sind und UI-basierte Workarounds offiziell als Bad-Practice gelten. Hinzu kommt noch eine für nicht-technische Nutzer komplex zu bedienende Benutzeroberfläche hinzu (B2).
Prefect erfüllt die Anforderungen der Priorität B größtenteils, nicht jedoch die Anforderung an ein Nutzermanagement (B3) und die Anforderung nach einem geringen Performance-Overhead (B6) nur eingeschränkt.
Argo Workflows erfüllt die meisten Soll-Kriterien, abgesehen von der Anforderung der Benutzerfreundlichkeit für nicht-technische Nutzer (B2), da die meiste Konfiguration, wie beispielsweise Schedules, im YAML-Format erfolgt.

In Hinblick auf die Anforderungen der **Priorität C** (Kann-Kriterien) unterscheiden sich die drei Frameworks enorm. **Apache Airflow** erfüllt drei Anforderungen nicht und eine weitere nur eingeschränkt. **Prefect** erfüllt ebenfalls drei Anforderungen nicht. **Argo Workflows** erfüllt lediglich eine Anforderung nicht und zwei weitere nur eingeschränkt.
Apache Airflow und **Argo Workflows** profitieren bei den Kann-Kriterien im Vergleich zu **Prefect** insbesondere bei der feingranularen Rechteverwaltung. Demgegenüber bieten **Prefect** und **Argo Workflows** Vorteile gegenüber **Apache Airflow** hinsichtlich der Möglichkeit zur Pausierung von Workflow-Ausführungen sowie der Unterstützung ereignisbasierter Alarmierungen.

2.4 Auswahl des Orchestrierungs-Frameworks

Auf Basis der qualitativen Auswertung der Anforderungen zeigt sich, dass keines der drei betrachteten Frameworks alle definierten Anforderungen vollständig erfüllt. Dennoch zeigen sich deutliche Unterschiede in der Eignung der untersuchten Frameworks für den gegebenen Einsatzzweck.

Apache Airflow überzeugt durch große Verbreitung, ausgereifte Kernfunktionen und das umfassende Rollen- und Rechtemodell. Große Nachteile zeigen sich jedoch in der stark codebasierten Konfiguration von Schedules und Parametrisierungen, sowie einem nicht unerheblich hohen systembedingten Performance-Overhead. Diese Nachteile erschweren die Nutzung für nicht-technische Nutzer und schränken die Nutzbarkeit über die grafische Benutzeroberfläche massiv ein.

Prefect bietet ein modernes Orchestrierungs-Framework mit geringer Komplexität bei Erstellung der Workflow-Schablonen. Zudem bietet Prefect eine umfassende Benutzeroberfläche, über welche die Erstellung von Schedules und die Konfiguration verschiedener Parametrisierungen auch für nicht-technische Nutzer problemlos möglich ist. Massive Einschränkungen ergeben sich jedoch durch ein fehlendes Nutzermanagement sowie durch die nicht gesicherte langfristige Wartung der Open-Source-Version, was eine langfristige Planung mit Prefect unmöglich macht.

Argo Workflows erfüllt als einziges Framework alle Muss-Kriterien uneingeschränkt und zeichnet sich insbesondere durch seine Kubernetes-native Architektur und den damit verbundenen sehr geringen systembedingten Performance-Overhead aus. Klare Einschränkungen gibt es jedoch bei der Benutzerfreundlichkeit, da die zentrale Konfiguration ausschließlich im YAML-Format erfolgt.

Zusammenfassend lässt sich feststellen, dass **Argo Workflows** für den betrachteten Anwendungsfall die geeignetste Lösung darstellt. Trotz der geringeren Benutzerfreundlichkeit überwiegen die Vorteile, sowie die vollständige Erfüllung der Muss-Kriterien.

Apache Airflow und **Prefect** stellen zwar gute Alternativen dar, sind jedoch entweder durch konzeptionelle Einschränkungen (Airflow) oder organisatorische und funktionale Einschränkungen in der Open-Source-Variante (Prefect) limitiert.

2.5 Umgang mit Einschränkungen in Argo Workflows

Wie bereits beschrieben, ist ergänzend zum Orchestrierungs-Framework die Entwicklung einer eigenen Anwendung notwendig, die detaillierte crawler-spezifische Informationen aufbereitet, wie beispielsweise die Anzahl verarbeiteter Datensätze.

Aufgrund dessen ergibt sich die Überlegung, die von Argo bereitgestellte Benutzeroberfläche nicht zu nutzen, da diese für technisch versierte Nutzer ausgelegt ist und somit für nicht-technische Nutzer kaum zu verwenden ist.

Als Folge daraus würde Argo Workflows nur serverseitig als Orchestrator genutzt werden, während die eigens entwickelte Anwendung um Funktionen zur Konfiguration erweitert werden würde.

Dadurch könnte sich die Anwendung explizit auf nicht-technische Nutzer zuschneiden lassen und mehrere der zuvor identifizierten Einschränkungen von Argo Workflows beheben.

Insbesondere die Anforderungen an eine einfach nutzbare Benutzeroberfläche für nicht-technische Nutzer (B2), die Integration eigener UI-Komponenten (C4), die Anzeige der Ressourcennutzung einzelner Ausführungen (C8) sowie die Visualisierung der Clusterauslastung (D1) wären dadurch umsetzbar.

Selbstverständlich bringt diese Lösung zusätzlichen Implementierungsaufwand mit sich, da Funktionen, die eigentlich bereits durch die Argo-Benutzeroberfläche bereitgestellt werden, neu zu implementieren sind. Aus diesem Grund werden im Folgenden die Funktionen bzw. Anforderungen identifiziert, welche eigentlich durch die Argo-UI abgedeckt wären und nun in der eigenen Anwendung umzusetzen sind.

Ein Querschnittsthema ist hierbei das **Nutzermanagement (A3, B4 und C6)**, welches jedoch ohnehin für die eigene Anwendung hätte implementiert werden müssen. Der zusätzliche Aufwand hierbei ergibt sich somit ausschließlich durch eine gegebenenfalls erforderliche feingranularere Rechteverwaltung.

Darüber hinaus sind folgende Anforderungen umzusetzen:

- *B1a: Manuelles Starten von Workflow-Ausführungen über das UI.*
  Diese Funktion benötigt die Entwicklung einer Frontend-Komponente sowie eines Backend-Endpunkts, der als Proxy fungiert und entsprechende API-Aufrufe an den Argo-Server weitergibt.
- *B1b und B1c: Definition von Schedules mit Parametrisierungen über das UI.*
  Im Frontend wird hierbei ebenfalls eine Komponente benötigt. Im Backend wird jedoch für diese Anforderungen mehr Funktionalität, als die eines Proxys benötigt. Die Konfiguration der Schedule mit deren Parametrisierung muss in ein YAML-Format konvertiert und anschließend an den Argo-Server geschickt werden.

- *B1d: Übersicht aller Ausführungen pro Schedule.*
  Hierfür wird die Entwicklung einer Frontend-Komponente sowie eines Backend-Endpunkts, der als Proxy fungiert und entsprechende API-Aufrufe an den Argo-Server weitergibt, erforderlich.
- *B5: Anzeige einfacher Statistiken zu Schedules.*
  Da Argo Workflows bereits alle Informationen über einen API-Endpunkt bereitstellt, wird hierbei ebenfalls nur die Frontend-Komponente benötigt und das Backend fungiert wieder als Proxy.
- *C2: Anzeige von Logs.*
  Auch hierbei wird lediglich eine Frontend-Komponente benötigt und das Backend als Proxy.

Zusammenfassend lässt sich somit sagen, dass die Implementierung der zusätzlichen Funktionalitäten, welche eigentlich bereits durch das Argo-UI angeboten werden, keinen unverhältnismäßig hohen Mehraufwand darstellt, da sich in den meisten Fällen der Mehraufwand auf die Implementierung einer Frontend-Komponente und des Backends als Proxy mit Authentifizierung und Rechteüberprüfung konzentriert. Aufgrund dessen überwiegen die Vorteile den zusätzlichen Implementierungsaufwand stark.
Dies sind die erleichterte Bedienung für nicht-technische Nutzer, die Möglichkeit zur Umsetzung weiterer Funktionen, welche das Argo-UI nicht bietet, sowie die Bereitstellung einer Ein-Tool-Lösung.

3 Anforderungen

In diesem Kapitel werden die Anforderungen an die Crawler-Management-Anwendung definiert, welche größtenteils in einem Anforderungsworkshop mit dem Entwicklungsteam (Leonie Färber, Dr. Georg Schwarz und Dirk Engelhard) des JValue-Biomed-Projekts gesammelt wurden.
Unterteilt werden die Anforderungen zunächst anhand ihrer Priorität:

- **Anforderungen der Priorität A** stellen diejenigen Anforderungen dar, welche umgesetzt werden müssen, um ein, für das JValue-Biomed-Projekt, gut nutzbares Crawler-Management-System bereitzustellen. Diese Anforderungen stellen das **Minimal Viable Product (MVP)** dar, welches im Rahmen dieser Arbeit implementiert werden soll.
- **Anforderungen der Priorität B** stellen diejenigen Anforderungen dar, welche umgesetzt werden sollen, jedoch für die Grundfunktionalität nicht notwendig sind. Diese stellen somit **Stretch Goals** für diese Arbeit dar, welche umgesetzt werden können, wenn die Bearbeitungszeit der Arbeit dies erlaubt.
- **Anforderungen der Priorität C** stellen diejenigen Anforderungen dar, welche umgesetzt werden können, um die Nutzbarkeit der Anwendung zu verbessern. Diese Anforderungen stellen ebenfalls **Stretch Goals** für die Arbeit dar, jedoch mit niedriger Priorität.

Zur besseren Nachvollziehbarkeit und Unterstützung der Architekturarbeit werden die Anforderungen in verschiedene Domänen unterteilt, die jeweils sowohl funktionale als auch nicht funktionale Anforderungen enthalten:

- **Betrieb, Deployment und Wartbarkeit:** Diese Domäne umfasst Anforderungen an den stabilen Betrieb der Anwendung, das Deployment, sowie die Wartbarkeit.
- **Sicherheit und Benutzerverwaltung:** Diese Domäne beschreibt Anforderungen an Autorisierung, Authentifizierung und Nutzerverwaltung.
- **Workflow und Schedule Management:** Diese Domäne enthält Anforderungen zur Verwaltung, Versionierung und geplanten Ausführung von Workflow-Schablonen.
- **Run-Übersicht und Run-Überwachung:** Diese Domäne beschreibt Anforderungen an Informationen zu Runs und Ereignissen in Bezug auf diese.
- **Statistiken:** Diese Domäne umfasst Anforderungen zur Visualisierung von Metriken für die Bewertung der Effektivität und Effizienz von Runs und Schedules.
- **Usability:** Diese Domäne beschreibt Anforderungen an die generelle Benutzerfreundlichkeit der Anwendung.

Anforderungen der Priorität A (Muss-Kriterien)

A1: Betrieb, Deployment und Wartbarkeit

Nicht funktionale Anforderungen:

A1.1: Die Anwendung muss **vollständig automatisiert** mittels CI/CD in die aktuell im Projekt genutzte Kubernetes-Umgebung deployt werden können.

A1.2: Die Anwendung muss **nach Abstürzen automatisch** ohne Datenverlust und ohne manuellen Eingriff **neu starten,** um die Zuverlässigkeit des Systems zu garantieren.

A1.3: Die Anwendung muss **unabhängig von Implementierungsdetails und interner Architektur der Crawler** sein, sodass Anpassungen an den Crawlern keine Anpassungen im Management-System erfordern.

A1.4: Die Anwendung muss so entworfen sein, dass ein **Wechsel des Orchestrierungs-Frameworks keine vollständige Neuentwicklung**, sondern lediglich die Entwicklung eines, zur Schnittstelle des neuen Orchestrierungs-Frameworks passenden, Clients erfordert, um ein Vendor Lock-in zu verhindern.

Funktionale Anforderungen:

A1.5: Die Anwendung muss **ausführliches Logging** bereitstellen, um Fehler nachvollziehbar analysieren zu können.

A2: Sicherheit und Benutzerverwaltung

Funktionale Anforderungen:

A2.1: Die Anwendung muss durch **Authentifizierung und Autorisierung** gegen unautorisierten Zugriff geschützt sein.

A2.2: Die Anwendung muss ein **Nutzermanagement mit mindestens zwei Rollen** (Leserechte/Schreibrechte) bieten, um die Rechte von bestimmten Nutzern einschränken zu können.

A3: Workflow und Schedule Management

Funktionale Anforderungen:

A3.1: Die **Workflow-Schablonen** müssen über **Git versioniert** werden, um die Nachverfolgbarkeit zu gewährleisten.

A3.2: Die Anwendung muss die Möglichkeit bieten, **parametrisierte Runs für Workflow-Schablonen manuell zu starten**, um einmalige Ausführungen zu ermöglichen.

A3.3: Die Anwendung muss eine Funktion bieten, **Schedules für Workflow-Schablonen zu definieren**, inklusive unterschiedlicher Parametrisierung je Schedule, um die regelmäßige automatisierte Ausführung von Workflow-Schablonen zu ermöglichen und dadurch die Aktualität der Datensätze zu gewährleisten.

A3.4: Die Anwendung muss eine Funktion bieten, die es ermöglicht, **Schedules pausieren, fortsetzen und löschen zu können**.

A3.5: Die Anwendung muss **konfigurierbare Retry-Mechanismen für Schedules** unterstützen, um die Fehler Resilienz zu verbessern.

A4: Run-Übersicht und Run-Überwachung

Funktionale Anforderungen:

A4.1: Die Anwendung muss **alle ausgeführten Runs anzeigen**, um dem Nutzer die Möglichkeit zu bieten, die Ausführung der Runs nachzuverfolgen und Informationen darüber zu erhalten.

A4.2: Die Anwendung muss eine **Filterung der Runs** nach folgenden Parametern anbieten, sodass die Suche nach bestimmten Runs erleichtert wird:

- Workflow-Schablone
- Start- und Enddatum
- Schedule
- Aktueller Status

A5: Statistiken

Funktionale Anforderungen:

A5.1: Die Anwendung muss **Statistiken pro Run** anzeigen, sodass Runs entsprechend dieser analysiert werden können:

- Anzahl der gesammelten Datensätze
- Anzahl der Cache Hits
- Dauer des Runs

A5.2: Die Anwendung muss **aggregierte Statistiken pro Schedule** anzeigen, um die Effektivität und Effizienz der Schedule zu analysieren und anhand dessen notwendige Änderungen zu erkennen:

- Gesamtanzahl gesammelter Datensätze
- Anzahl gesammelter Datensätze im Zeitverlauf
- Anzahl neu gesammelter Datensätze im Zeitverlauf
- Verhältnis neu gesammelter Datensätze zu insgesamt gesammelter Datensätze im Zeitverlauf

A6: Usability

Nicht funktionale Anforderungen:

A6.1: Die **Steuerung der Crawler muss vollständig über die grafische Nutzeroberfläche** möglich und somit für **nicht-technische Nutzer einfach nutzbar** sein, da die Steuerung der Crawler nicht zwangsläufig im Aufgabenbereich der Entwickler liegt.

Anforderungen der Priorität B (Soll-Kriterien)

B1: Betrieb, Deployment und Wartbarkeit

Funktionale Anforderungen:

B1.1: Die Anwendung soll einen **einfachen Health Status** mit Informationen zum Cluster und Orchestrator anzeigen, um unmittelbar einen Überblick über mögliche Probleme zu erhalten.

B2: Sicherheit und Benutzerverwaltung

Es gibt keine Anforderungen der Priorität B zu Sicherheit und Benutzerverwaltung.

B3: Workflow und Schedule Management

Funktionale Anforderungen:

B3.1: Die Anwendung soll es ermöglichen, **Zeitpläne, Retry-Mechanismen und Parameter bestehender Schedules ändern zu können**, um beispielsweise notwendige Änderungen aufgrund von Anpassungen am Crawler einfach umsetzen zu können.

B3.2: Die Anwendung soll die Möglichkeit bieten, **Schedules für die Ausführung zu priorisieren**, um die Ausführung von wichtigeren Schedules sicherzustellen.

B3.3: Die Anwendung soll **Breaking Changes in Workflow-Schablonen erkennen und eine Warnung dafür anzeigen**, sodass Nutzer darauf hingewiesen werden, dass die Ausführungen der Schedule ohne Anpassung wahrscheinlich fehlschlagen werden.

B3.4: Die Anwendung soll für jede **Schedule den nächsten geplanten Run anzeigen**, um jederzeit zu wissen, wann die Datensätze wieder aktualisiert werden.

B4: Run-Übersicht und Run-Überwachung

Funktionale Anforderungen:

B4.1: Die Anwendung soll den **Fortschritt laufender Runs** anzeigen, um diese überwachen und dadurch mögliche Deadlocks erkennen zu können.

B4.2: Die Anwendung soll **Logs der Runs** anzeigen, um eine schnelle erste Analyse bei Fehlern zu ermöglichen.

B4.3: Die Anwendung soll **Nachrichten der Crawler, klassifiziert nach Dringlichkeit,** anzeigen können, sodass die Crawler explizit auf mögliche Probleme hinweisen können und stets erkennbar ist, in welcher Phase der Crawler sich derzeit befindet.

B5: Statistiken

Es gibt keine Anforderungen der Priorität B zu Statistiken.

B6: Usability

Es gibt keine Anforderungen der Priorität B zu Usability.

Anforderungen der Priorität C (Kann-Kriterien)

C1: Betrieb, Deployment und Wartbarkeit

Funktionale Anforderungen:

C1.1: Die Anwendung kann die aktuelle **Cluster-Auslastung** anzeigen, um notwendige Anpassungen in der bereitgestellten Infrastruktur frühzeitig zu erkennen.

C1.2: Die Anwendung kann die aktuelle **Warteschlange für Runs visualisieren**, sodass schnell erkennbar ist, wenn die aktuelle Infrastruktur für die Ausführungsfrequenz von Crawlern nicht mehr ausreichend ist.

C1.3: Die Anwendung kann **automatisch die Erreichbarkeit von Ziel-URLs überprüfen und Warnungen anzeigen**, um auf wahrscheinliche Probleme bei der Ausführung von Schedules hinzuweisen.

C1.4: Die Anwendung kann **Alarmierungen für besondere Ereignisse verschicken**, um ein rechtzeitiges manuelles Eingreifen zu ermöglichen, bevor Probleme überhaupt erst auftreten:

- Überfüllte Warteschlange
- Speicherknappheit der Data-Storages der Datensätze

C2: Sicherheit und Benutzerverwaltung

Funktionale Anforderungen:

C2.1: Die Anwendung kann eine **feingranulare Rechteverwaltung** bieten, um Nutzer auf die Funktionen einzuschränken, für die sie befugt sind.

C3: Workflow und Schedule Management

Funktionale Anforderungen:

C3.1: Die Anwendung kann **einmalig geplante Runs** unterstützen, sodass diese beispielsweise als Testausführungen zu Zeiten mit erwarteter geringer Auslastung eingeplant werden können.

C3.2: Die Anwendung kann eine **Übersicht aller geplanten Schedule-Ausführungen** anzeigen, um zentral sicherzustellen, dass sich Ausführungen nicht zu bestimmten Zeitpunkten übermäßig häufen.

C3.3: Die Anwendung kann den **Aufbau einzelner Crawler visualisieren**, um einen Überblick über deren Funktionsweise zu erhalten.

C3.4: Die Anwendung kann die Funktion bieten, **Workflow-Schablonen über die Benutzeroberfläche erstellen** zu können, sodass die Erstellung von Workflow-Schablonen auch technisch unerfahrene Nutzer übernehmen könnten.

C3.5: Die Anwendung kann **Nutzer bei Breaking Changes in Workflow-Schablonen von Schedules alarmieren**, sodass Nutzer schon vor Ausführung des nächsten Runs vor potenziellen Fehlern gewarnt werden und die notwendigen Anpassungen durchführen können.

C4: Run-Übersicht und Run-Überwachung

Funktionale Anforderungen:

C4.1: Die Anwendung kann den **Ressourcenverbrauch eines Runs** anzeigen, um Hinweise auf übermäßige Ressourcennutzung zu erhalten.

C4.2: Die Anwendung kann **Warnungen bei auffälligem Run-Verhalten anzeigen**, um Probleme bei der Ausführung von Runs erkennen zu können:

- Ungewöhnlich kurze oder lange Laufzeiten
- Ungewöhnlich wenige neu gesammelte Datensätze

C4.3: Die Anwendung kann die Funktion bieten, **Runs zwischen einzelnen Verarbeitungsschritten zu pausieren**, um Ressourcen für andere potenziell wichtigere Ausführungen freizugeben.

C4.4: Die Anwendung kann Nutzer bei **fehlgeschlagenen Runs alarmieren**.

C5: Statistiken

Es gibt keine Anforderungen der Priorität C zu Statistiken.

C6: Usability

Es gibt keine Anforderungen der Priorität C zu Usability.

4 Architektur

In diesem Kapitel wird die Architektur des zu entwickelnden Crawler-Management-Systems konzipiert und vorgestellt. Das Kapitel orientiert sich dabei an der Architekturdokumentation Arc42.
Auf die Kapitel zur Einführung und den Zielen des Systems, sowie einem Glossar wurde dabei explizit verzichtet, da diese Inhalte bereits in vorherigen Kapiteln beschrieben wurden.
Ebenso wurde auf die Verteilungssicht verzichtet, da in diesem Kapitel die Architektur eines generischen Crawler-Management-Systems beschrieben und daher keine konkrete Deployment- oder Laufzeitumgebung betrachtet wird.
Zunächst beschreibt dieses Kapitel die Randbedingungen an die Architektur, gefolgt von einer Abgrenzung, des in dieser Architekturdokumentation, beschriebenen Systems im Gesamtkontext.
Anschließend folgen übergeordnete Lösungsstrategien, sowie eine Bausteinsicht des Systems.
Für die genauere Betrachtung einzelner Bausteine und Abläufe werden Laufzeitsichten in Form von Sequenzdiagrammen visualisiert.
Um den Betrieb der Anwendung darzustellen wird eine Verteilungssicht genutzt.
Darauf folgt die Beschreibung von Querschnittskonzepten und wichtigen Architekturentscheidungen, sowie Lösungsstrategien.
Zuletzt werden noch Risiken beschrieben, welche durch die präsentierte Architektur entstehen.

4.1 Randbedingungen

Randbedingungen sind jegliche Anforderungen, die Software-Architekten in ihrer Freiheit in Design- und Implementierungsentscheidungen oder ihrer Freiheit in Entscheidungen über den Entwicklungsprozess einschränken. (Starke & Hruschka, o. D.)

R1: Technologie-Stack

Im JValue-Projekt wurde bisher folgender Technologie-Stack eingesetzt:

Backend:

- Programmiersprache: TypeScript[6]
- Framework: NestJS[7] (basierend auf NodeJS[8])
- ORM: TypeORM[9]
- Datenbank: PostgreSQL[10]

[6] https://www.typescriptlang.org
[7] https://nestjs.com
[8] https://nodejs.org/en
[9] https://typeorm.io
[10] https://www.postgresql.org

Frontend:

- Programmiersprache: TypeScript
- Framework: React[11]
- Design-Bibliothek: shadcn[12]

Die genannten Technologien und Frameworks haben sich im bisherigen Projekt bewährt. Zudem besitzt das Entwicklungsteam, welches die Anwendung nach Abschluss dieser Arbeit weiterentwickeln wird, bereits umfassende Kompetenzen in diesen Technologien und Frameworks, weswegen die Nutzung dieser wünschenswert ist. Vom hier beschriebenen Technologie-Stack sollte somit nur aus triftigen Gründen abgewichen werden.

R2: Repository und Projektstruktur

Die gesamte Anwendung muss in einem GitHub Monorepository verwaltet werden. Im bisherigen Projekt wurde hierfür ein NX-Monorepo eingesetzt, was daher auch für dieses System verwendet werden soll.

R3: Parallele Entwicklung der Crawler

Die Entwicklung der Crawler erfolgt parallel zur Entwicklung dieser Anwendung, weswegen die Schnittstellen zwischen Management-System und Crawlern frühzeitig definiert werden müssen.

R4: Laufzeitumgebung

Für den Betrieb der Anwendung steht ein Kubernetes Cluster zur Verfügung, welches ausschließlich aus dem Universitätsnetzwerk der Friedrich-Alexander-Universität erreichbar ist.

Daraus ergeben sich folgende Randbedingungen:

- Die Anwendung muss containerisiert betrieben werden.
- Die Deployment- und Konfigurationslogik muss kompatibel zu Kubernetes sein.
- Das Deployment durch CI/CD Pipelines muss über einen Self Hosted Runner erfolgen, der im Universitätsnetzwerk läuft, da sonst kein Zugriff auf das Cluster möglich ist.

R5: CI/CD

Für Continous Integration und Continous Deployment müssen GitHub Action Workflows genutzt werden, da für diesen Zweck ein entsprechendes Kontingent, sowie ein Self Hosted Runner für das Deployment in die Infrastruktur im Universitätsnetzwerk zur Verfügung stehen.

[11] https://react.dev
[12] https://www.shadcn.io

4.2 Kontextabgrenzung

Die Kontextsicht zeigt, wie das zu entwickelnde System in seine Umgebung eingebettet ist. Dabei wird das System als Blackbox dargestellt und zeigt dabei dessen externe Schnittstellen, sowie angrenzende Systeme.
Ziel dieser Darstellung ist es, Verantwortlichkeiten des zu entwickelnden Systems klar einzugrenzen und Abhängigkeiten zu anderen Systemen darzustellen. (Starke & Hruschka, o. D.)
Für die Darstellung wird als Beispiel das Orchestrierungs-Framework Argo Workflows angenommen.

Im Mittelpunkt dieser Architekturdokumentation stehen das Management-Frontend und das Management-Backend, welche gemeinsam das Crawler-Management-System bilden. Darüber hinaus existieren weitere Systeme, die teilweise im Einflussbereich des Projekts oder als externe Systeme außerhalb des Einflussbereiches liegen.

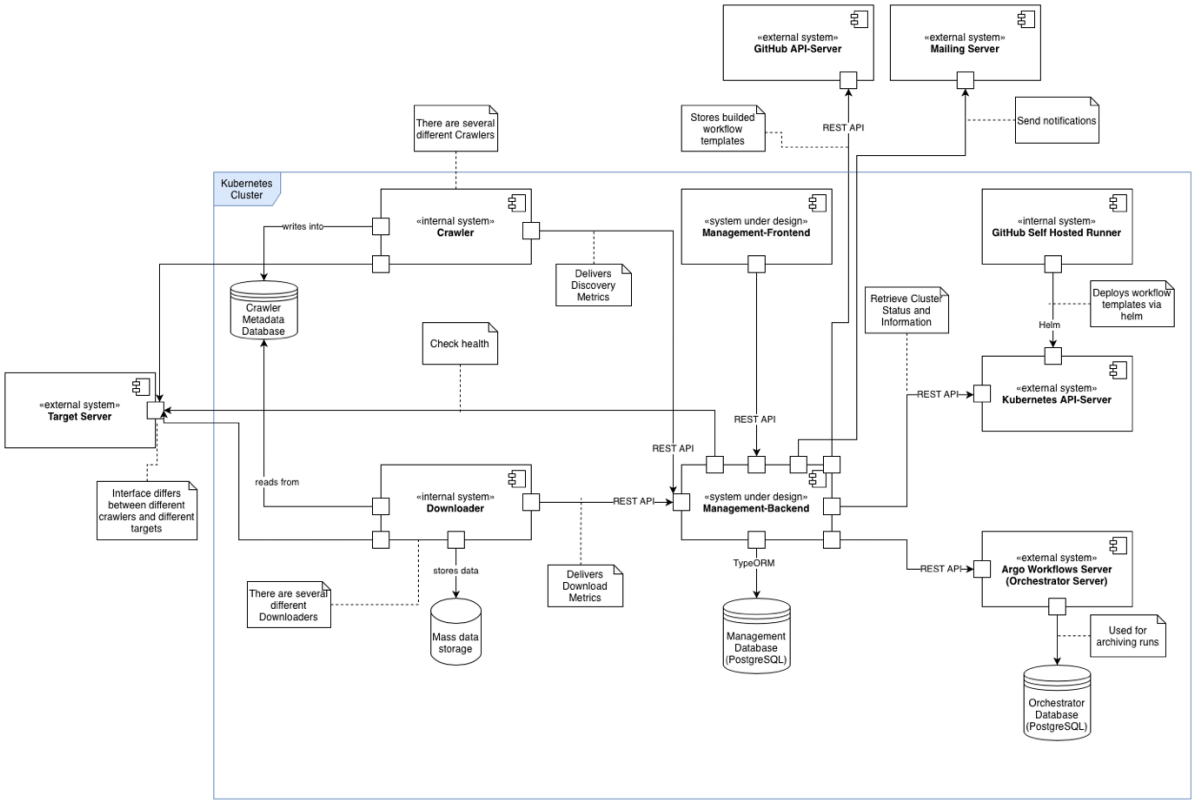

Abbildung 1: Kontextsicht

Management-Backend
Das Management-Backend stellt die zentrale Komponente des Systems dar und agiert dabei mit verschiedenen anderen Systemen.
Zu den Hauptaufgaben des Backends zählen dabei:

- Bereitstellung einer API für das Management-Frontend
- Verwaltung von Nutzern, Schedules, Runs und Metriken
- Kommunikation mit dem Server des Orchestrators (z.B. Argo Workflows)

Zudem fungiert das Management-Backend als Abstraktionsschicht zwischen dem konkret genutzten Orchestrierungs-Framework und dem Management-Frontend.

Management-Frontend
Das Management-Frontend ist die einzige Benutzeroberfläche zur Steuerung und Überwachung der Crawler. Dabei interagiert es ausschließlich mit dem Management-Backend.

Management-Database
Die Management-Database ist eine PostgreSQL Datenbank, die zur Speicherung der, vom Management-Backend, verwalteten Daten dient.

Argo Workflows Server (Orchestrator Server)
Der Argo Workflows Server steht in dieser Darstellung stellvertretend für den gesamten Orchestrator, der ebenfalls innerhalb des Kubernetes Clusters läuft.
Das Management-Backend kommuniziert mit dem Orchestrator Server, um beispielsweise Daten zu Runs abzufragen oder Schedules zu erstellen.
Im Falle von Argo Workflows wird zudem noch eine separate Datenbank zur Persistierung von beendeten Runs benötigt, wofür eine PostgreSQL Datenbank genutzt wird.

Crawler
Es existieren mehrere unterschiedliche Crawler, beispielsweise für verschiedene Schnittstellenarten, wie Representational State Transfer (REST) APIs oder File Transfer Protocol (FTP) Server.
Die Crawler werden im Rahmen des Projekts selbst entwickelt und liegen somit grundsätzlich im Einflussbereich des Systems. Dennoch sollen die Crawler unabhängig vom Management-System entwickelt werden, weswegen lediglich die Schnittstellen zwischen Crawlern und Crawler-Management-Backend vorgegeben werden. (Anforderung A1.3)
Die Crawler liefern Metriken und Events an das Management-Backend.

Downloader
Der Download der Nutzdaten ist in diesem Projekt nicht Aufgabe der Crawler selbst, sondern wird von einer separaten Komponente, im Folgenden Downloader genannt, übernommen.
Dieser liest die Metadaten aus der Crawler-Metadata Datenbank und lädt anschließend die Nutzdaten von den Target Servern in ein Mass-Data-Storage (z.B: AWS-S3 Bucket). Dabei könnte die Downloadfunktionalität grundsätzlich ebenso Teil des Crawlers sein. Für das Management-Backend ist dies jedoch irrelevant, solange die definierte Schnittstelle angesprochen wird.

Crawler Metadata Database
Die Crawler-Metadata Datenbank wird von den Crawlern mit Metadaten zu den entdeckten Datensätzen befüllt. Der Downloader liest diese danach aus und lädt anschließend die zugehörigen Nutzdaten von den Target Servern.
Die konkrete Technologie oder interne Struktur der Datenbank sind für das Management-System nicht relevant.

Target Server
Die Target Server stellen die externen Server dar, auf denen die Nutzdaten liegen. Die Target Server werden sowohl von den Crawlern zum Sammeln der Metadaten angesprochen, als auch von den Downloadern zum anschließenden Download. Zudem spricht das Management-Backend die Target Server an, um deren Verfügbarkeit zu überprüfen. (Anforderung C1.3)

Kubernetes API-Server
Der Kubernetes API-Server stellt die Schnittstelle zum Cluster dar.
Das Management-Backend greift ausschließlich lesend darauf zu, um den Health Status des Clusters zu überprüfen (Anforderung B1.1) und um Informationen zum Cluster, wie beispielsweise die Auslastung, zu erhalten (Anforderungen C1.1, C1.2, C1.4, C4.1).
Zudem wird der API-Server im Rahmen von GitHub Action Workflows über Helm genutzt, um die Anwendung selbst sowie die Workflow Templates, automatisiert zu deployen (Anforderungen A1.1, A3.1).

GitHub Self Hosted Runner
Der GitHub Self Hosted Runner wird für das Deployment benötigt, da das Kubernetes Cluster ausschließlich aus dem Universitätsnetzwerk der Friedrich-Alexander-Universität erreichbar ist. (Anforderungen A1.1, A3.1, Randbedingungen R4, R5)
Er wird innerhalb der GitHub Actions Workflows genutzt, um in das Cluster zu deployen.

GitHub API-Server
Der GitHub API-Server wird benötigt, um Crawler Templates, die über das UI erstellt wurden, in das GitHub Repository zu synchronisieren, um die Versionierung der Templates sicher zu stellen (Anforderung C3.4 (A3.1)).

Mailing Server
Der Mailing Server wird benötigt, um Alarmierungen an Nutzer zu versenden. (Anforderungen C1.4, C3.5, C4.4)

4.3 Lösungsstrategien

Lösungsstrategien beschreiben grundlegende Architekturentscheidungen, die einen signifikanten Einfluss auf den Aufbau und die Gesamtarchitektur des Systems haben. Sie leiten sich aus wichtigen Anforderungen und Randbedingungen ab.

4.3.1 Backend-Technologie-Stack

Für die Umsetzung des Backends wird das Framework NestJS in Verbindung mit der Programmiersprache Typescript eingesetzt. Diese Entscheidung folgt der, in Abschnitt 4.1 definierten, Randbedingung R1, den im JValue-Projekt genutzten Technologie-Stack weiterzuverwenden und so den Einarbeitungsaufwand zu minimieren.
Zur Persistierung der Daten wird eine PostgreSQL-Datenbank verwendet. Der Datenbankzugriff erfolgt dabei über das Object-Relational-Mapping Framework TypeORM, welches eine klare Trennung zwischen Geschäftslogik und Datenbanklogik ermöglicht.

4.3.2 Frontend-Technologie-Stack

Im Management-Frontend wird das Framework React in Kombination mit TypeScript genutzt. Als Design-Bibliothek wird Shadcn eingesetzt.
Auch diese Technologieentscheidungen ergeben sich aus der Randbedingung R1 mit dem Ziel der Minimierung des Einarbeitungsaufwands des bestehenden Entwicklungsteams.

4.3.3 Repository-Struktur

Die gesamte Anwendung wird in einem NX-Monorepo verwaltet und folgt dabei Randbedingung R2.

4.3.4 Deployment

Die Anwendung wird containerisiert betrieben und in das bestehende Kubernetes Cluster deployt. Um ein automatisches Deployment zu ermöglichen, werden Helm[13] Charts genutzt.

[13] https://helm.sh/de/

Aufgrund der Randbedingungen R4 und R5 erfolgt das Deployment über einen Github Actions Workflow, der auf einem Self Hosted Runner im Universitätsnetzwerk läuft.

4.3.5 Orchestrator-Unabhängigkeit

Zur Erfüllung von Anforderung A1.4, die einen Wechsel des Orchestrierungs-Frameworks ohne vollständige Neuentwicklung des Management-Systems fordert, wird eine Abstraktionsschicht benötigt.
Im Backend wird dafür eine Orchestrator-Abstraktionsschicht eingeführt.
Alle orchestrator-abhängigen Funktionalitäten werden über ein generisches Interface angesprochen, welches durch eine konkrete Implementierung in Abhängigkeit des gewählten Orchestrator-Frameworks umgesetzt wird.
Da diese Abstraktionsschicht im Backend liegt, ist das Frontend vollständig unabhängig vom genutzten Orchestrierungs-Framework (4.6.3 Orchestrator-Abstraktionsschicht).

4.3.6 Unabhängigkeit des Systems vom Crawler-Aufbau

Der interne Aufbau und die Zusammensetzung der Crawler werden nicht vom Management-System beeinflusst, um Anforderung A1.3 zu erfüllen.
Hierfür stellt das Management-System lediglich generische Schnittstellen zur Erfassung von Metriken, Events und Fortschrittsinformationen bereit. Welche Komponenten des Crawlers diese Schnittstellen ansprechen, ist für das Management-System daher irrelevant.

4.4 Bausteinsicht

Der überwiegende Teil der Geschäftslogik wird im Backend implementiert.
Hierfür existieren verschiedene Gründe.
Erstens verringert eine schlanke Frontend-Implementierung die Ladezeit der Website maßgeblich, da deutlich weniger Daten übertragen werden müssen. Eine Studie von Google zeigt, dass steigende Ladezeiten unmittelbare Auswirkungen auf Absprungraten und Benutzerfreundlichkeit haben. (Google, 2017)
Zweitens erhöht diese Entscheidung die Wartbarkeit, da Änderungen an der Geschäftslogik somit größtenteils nur im Backend vorgenommen werden müssen.
Drittens werden dadurch Sicherheitsrisiken verringert, da Client-Code vom Nutzer einsehbar und manipulierbar ist und somit keinerlei vertrauenswürdige Grundlage für relevante Entscheidungen, wie beispielsweise Rechteprüfung bietet. Laut OWASP stellt dies das größte Sicherheitsrisiko von Webapplikationen im Jahre 2025 dar. (The OWASP Foundation, 2025)
Viertens verbessert sich die Testbarkeit der Anwendung, da die Backend-Anwendung isoliert getestet werden kann.
Daher enthält das Frontend hauptsächlich nur Anzeigelogik, weswegen für das Frontend nur auf besondere Komponenten in den Abschnitten Querschnittskonzepte, Architekturentscheidungen und dem Implementierungskapitel eingegangen wird.

Im Folgenden werden somit alle Bausteine des Management-Backends, sowie deren Interaktionen dargestellt. Anschließend wird jeder Baustein knapp beschrieben und die Anforderungen genannt, wegen welchen der jeweilige Baustein hauptsächlich benötigt wird.

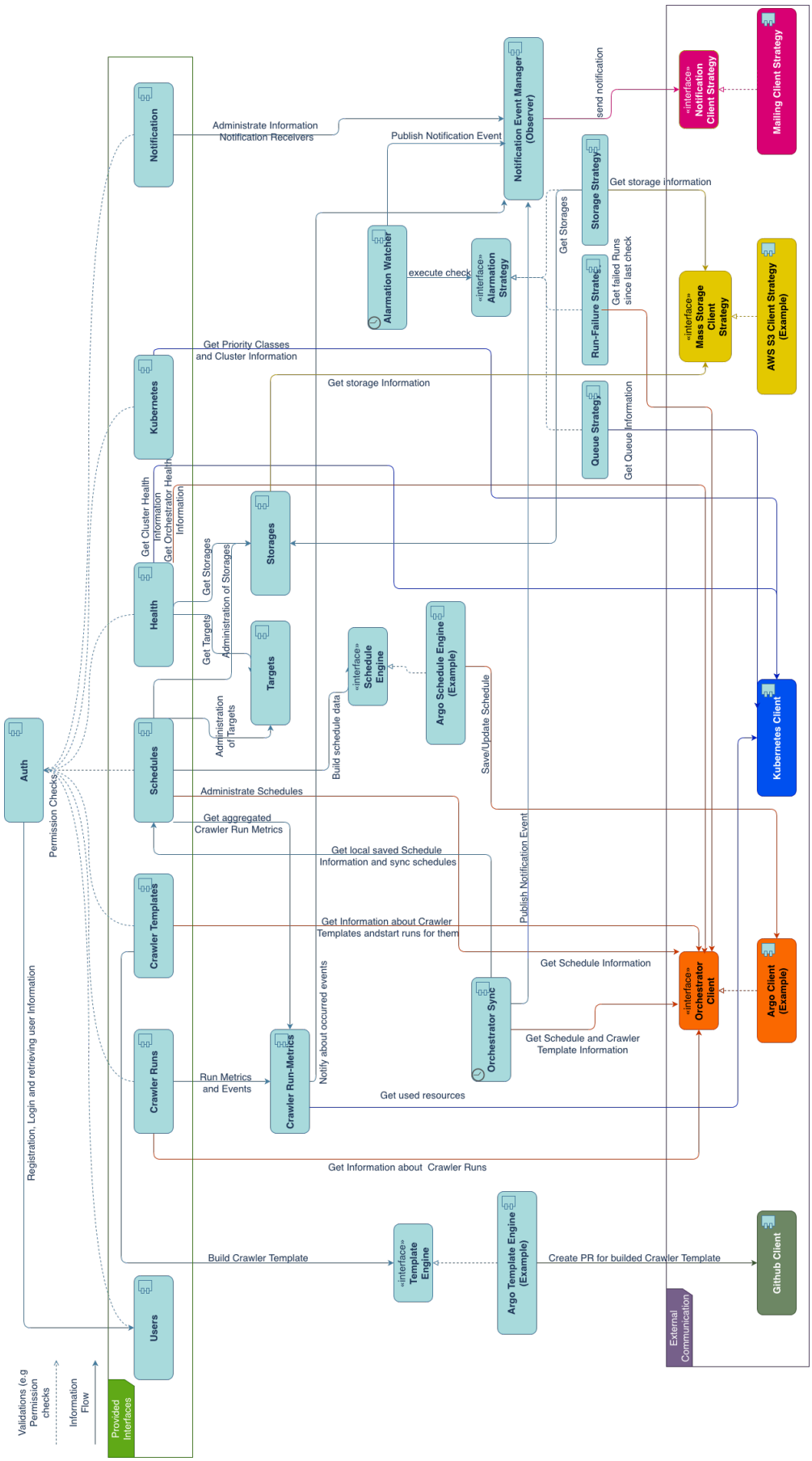

Abbildung 2: Bausteinsicht

Auth-Modul
Das *Auth-Modul* dient als zentraler Einstiegspunkt für alle API-Anfragen an das Management-Backend und übernimmt dabei die Authentifizierung von Nutzern.
Zudem ist es verantwortlich für Rechteprüfung, sowie Login und Registrierung. Hierfür interagiert es mit dem *Users-Modul*, um Informationen zu den Nutzern zu erhalten (Anforderung A2.1, C2.1).

Users-Modul
Das *Users-Modul* verwaltet sämtliche Nutzerdaten und speichert diese in der Datenbank. Dazu gehört beispielsweise das Anlegen von neuen Nutzern sowie die Anpassung der Berechtigungen (Anforderung A2.2, C2.1).

Orchestrator Client Interface
Das *Orchestrator Client Interface* definiert die Schnittstelle zur Interaktion mit dem Orchestrierungs-Framework. Alle Zugriffe auf den Orchestrator finden ausschließlich über dieses Interface statt.
Dadurch wird sichergestellt, dass bei einem Wechsel des Orchestrierungs-Frameworks lediglich eine neue Implementierung gegen das Interface notwendig ist, ohne dass die Geschäftslogik verändert werden muss (Anforderung A1.4, sowie alle Anforderungen für die mit dem Orchestrator kommuniziert werden muss).

Crawler Runs-Modul
Das *Crawler Runs-Modul* bietet API-Endpunkte an, um Informationen über laufende und abgeschlossene Runs abzurufen. Die Daten hierfür kommen vom Orchestrator Client.
Zudem dient das Modul als Schnittstelle für Crawler zur Speicherung von Metriken, Events und Fortschrittsinformationen. Die Speicherung und Auswertung dieser Informationen findet jedoch im *Crawler Run Metrics-Modul* statt (Anforderungen A4.1, A4.2, A5.1, A5.2, B4.1, B4.2, B4.3, C4.1, C4.2, C4.3).

Crawler Run Metrics-Modul
Das *Crawler Run Metrics-Modul* ist für die Speicherung der Metriken und Events von Runs in der Datenbank zuständig und liefert diese Informationen sowohl aggregiert als auch run-spezifisch. Um Informationen zu den genutzten Ressourcen eines Runs zu erhalten, wird der *Kubernetes Client* angesprochen.
Abhängig von der Dringlichkeit werden Events an den *Notification Event Manager* weitergeleitet (Anforderungen A5.1, A5.2, B4.1, B4.3, C4.1, C4.2).

Kubernetes Client
Der *Kubernetes Client* liefert Informationen zum Status des Clusters, wie beispielsweise den Health Status, oder auch zur aktuellen Auslastung und zum Ressourcenverbrauch der Pods

einzelner Runs. Außerdem liefert er die Prioritätsklassen, die für die Priorisierung von Runs zur Verfügung stehen (Anforderungen B1.1, B3.2, C1.1, C1.2, C1.4, C4.1).

Crawler Templates-Modul

Das *Crawler Templates-Modul* stellt API-Endpunkte zur Verwaltung und Anzeige von Informationen zu den Crawler Templates bereit. Auch hierbei kommen die Informationen vom *Orchestrator Client*. Zudem wird ein API-Endpunkt zum Starten eines Runs eines Templates angeboten. Das Erstellen von Crawler Templates wird an die Implementierung des *Template Engine Interfaces* delegiert (Anforderungen A3.2, A3.3, C3.4).

Template Engine Interface

Das *Template Engine Interface* stellt eine Schnittstelle bereit, über welche Crawler Templates zusammengebaut werden können. Die Schnittstelle wird durch eine orchestrator-spezifische Template Engine implementiert. Je nach Orchestrator erfolgt die Persistierung der Templates entweder über den Orchestrator, welcher die Versionierung in einem Git Repository übernimmt, oder direkt über den *GitHub Client* (Anforderung C3.4).

GitHub Client

Der *GitHub Client* kapselt die Kommunikation mit der GitHub-API und bietet dabei beispielsweise Schnittstellen zur Erstellung von Pull Requests an (Anforderung C3.4).

Schedules-Modul

Das *Schedules-Modul* bildet einen zentralen Bestandteil des Systems und verwaltet dabei sämtliche zeitgesteuerte Ausführungen.

Dabei bietet es über API-Endpunkte Funktionen zum Erstellen, Ändern und Löschen von Schedules an. Zusätzlich liefert es detaillierte Informationen zu den Schedules und aggregierte Statistiken zu deren Runs, welche das *Crawler Run Metrics-Modul* zur Verfügung stellt.

Die Schedules werden zum einen lokal in der eigenen Datenbank (siehe 4.7.3 Eigene Persistierung von Schedules) gespeichert zum anderen auch über das *Schedule Engine Interface* an den Orchestrator zur Speicherung geschickt.

Ebenso wird eine Schnittstelle zur Synchronisierung von, im Orchestrator erstellten, Schedules sowie eine Schnittstelle zur Speicherung von Fehlern pro Schedule bereitgestellt.

Ein weiterer Verantwortungsbereich des *Schedules-Moduls* liegt in der Erkennung von Target- und Storage-Parametern. Wird eine Schedule erstellt oder geändert und dabei ein solcher Parameter gefunden, wird hierfür das *Targets-* bzw. *Storages-Modul* für dessen Verwaltung angesprochen. Ein Target-Parameter gibt an, von welcher URL der Crawler die Datensätze holen soll. Ein Storage-Parameter definiert, wo die Datensätze gespeichert werden sollen. Bei Löschung einer Schedule werden die zugehörigen Targets bzw. Storages

über das jeweilige Modul entfernt (Anforderungen A3.3, A3.4, A3.5, A5.2, B3.1, B3.2, B3.3, B3.4, C1.3, C3.2, C3.5).

Schedule Engine Interface
Das *Schedule Engine Interface* definiert eine Schnittstelle zur Erstellung von Schedules, welche durch eine orchestrator-spezifische Implementierung erfüllt wird. Die erstellte Schedule wird anschließend über den Orchestrator Client zur Speicherung an den Orchestrator weitergegeben (Anforderungen A3.3, A3.5, B3.1, B3.2).

Targets-Modul
Das *Targets-Modul* verwaltet die Target-URLs, von welchen die Crawler ihre Daten abfragen. Dabei bietet es Methoden zur Speicherung und Löschung von Targets in der Datenbank an. Zudem können alle Targets, die mindestens einer aktiven Schedule zugeordnet sind, abgefragt werden (Anforderung C1.3).

Storages-Modul
Das *Storages-Modul* verwaltet die Speicherorte der Crawler, in welchen die Nutzdaten abgelegt werden. Hierfür bietet es Methoden zur Speicherung und Löschung von Informationen zu Speicherorten in der eigenen Datenbank an. Darüber hinaus besteht die Möglichkeit zur Abfrage aller Storages, inklusive deren Auslastung, die mindestens einer aktiven Schedule zugeordnet sind. Die Auslastung eines Storage wird über die entsprechende *Mass Storage Client Strategy* Implementierung abgefragt (Anforderung C1.4).

Mass Storage Client Strategy Interface
Das *Mass Storage Client Strategy Interface* definiert die Schnittstelle für storage-spezifische Clients zur Abfrage von Informationen über die Auslastung der jeweiligen Speicherorte (Anforderung C1.4).

Orchestrator Sync-Modul
Das *Orchestrator Sync-Modul* ist für die Sicherstellung der Konsistenz zwischen den im Orchestrator gespeicherten und den lokal gespeicherten Daten zuständig. Es fragt hierfür in regelmäßigen Abständen vom *Orchestrator Client* alle Schedules ab und gibt diese zum Abgleich an das *Schedules-Modul* weiter.
Zudem werden in regelmäßigen Abständen für alle lokal gespeicherten Schedules die zugehörigen Crawler Templates abgefragt. Das Modul überprüft zum einen, ob das referenzierte Crawler Template noch existiert, und zum anderen, ob die in der Schedule definierten Parameter noch mit den geforderten Parametern des Crawler Templates übereinstimmen. Bei festgestellten Inkonsistenzen werden entsprechende Fehler an das

Schedules-Modul zur Persistierung übergeben und als Alarm-Event an den *Notification Event Manager* weitergereicht (Anforderungen B3.3, C3.5).

Health-Modul
Das *Health-Modul* stellt API-Endpunkte zur Abfrage des Systemzustands zur Verfügung und ermöglicht dabei eine Abfrage über den Status des Kubernetes Clusters und des Orchestrators, sowie zur Erreichbarkeit der Target-URLs.
Der Zustand des Clusters und des Orchestrators wird über den *Kubernetes Client* bzw. *Orchestrator Client* abgefragt.
Zur Überprüfung der Erreichbarkeit der Target-URLs werden zunächst vom *Targets-Modul* alle relevanten Targets abgefragt und anschließend deren Erreichbarkeit überprüft. Zudem kann die Auslastung der Speicherorte über das *Storages-Modul* abgefragt werden (Anforderungen B1.1, C1.3).

Kubernetes-Modul
Das *Kubernetes-Modul* stellt API-Endpunkte zur Verfügung, um Clusterinformationen, wie Auslastung oder verfügbare Prioritätsklassen, abzurufen. Dafür greift das Modul ausschließlich auf den *Kubernetes Client* zu (Anforderungen B3.2, C1.1, C1.2).

Notification-Modul
Das *Notification-Modul* bietet API-Endpunkte zur Verwaltung von Notification-Empfängern und gibt diese Informationen an den *Notification Event Manager* weiter (Anforderungen C1.4, C3.5, C4.4).

Notification Event Manager-Modul (Publisher-Subscriber Pattern)
Das *Notification Event Manager-Modul* fungiert als Message-Broker im Publisher-Subscriber Pattern. Hierbei stellen die Nutzer die Subscriber dar, welche alarmiert werden wollen und die Alarmierungsquellen die Publisher. Der Event Manager bietet hierfür die Möglichkeit, Nutzer hinzuzufügen, zu löschen und deren Alarmierungspräferenzen zu ändern.
Zudem bietet das Modul eine Schnittstelle, über welche die Publisher Events bzw. Alarme senden können. Wird solch ein Event empfangen, werden über die Datenbank die Subscriber abgefragt, die dieses Event abonniert haben und anschließend über den gewünschten Alarmierungsweg (z.B. Mail) benachrichtigt (Anforderungen C1.4, C3.5, C4.4).

Alarmation Watcher-Modul
Das *Alarmation Watcher-Modul* ist ein Publisher des *Notfication-Event-Managers*.
Es überprüft regelmäßig den Systemzustand anhand registrierter *Alarmierungsstrategien* und leitet erkannte Warnungen als Events an den *Notification Event Manager* weiter (Anforderungen C1.4, C3.5, C4.4).

Alarmation Strategy Interface
Das *Alarmation Strategy Interface* definiert die Schnittstelle für Strategien zur Zustandsüberwachung. Jede Strategie überprüft dabei einen bestimmten Teil des Gesamtsystems und liefert bei Auffälligkeiten ein Warn-Event zurück (Anforderungen C1.4, C3.5, C4.4).

Queue Strategy
Die *Queue Strategy* überprüft mithilfe des *Kubernetes Clients* den Zustand der Warteschlangen und liefert bei Überlastung ein Warn-Event zurück (Anforderung C1.4).

Run Failure Strategy
Die *Run Failure Strategy* fragt über den *Orchestrator Client* fehlgeschlagene Runs seit der letzten Überprüfung ab und erzeugt entsprechende Warn-Events. Sollte der Orchestrator die Möglichkeit bieten, API-Requests bei fehlgeschlagenen Runs zu senden, so könnte stattdessen auch ein Endpunkt angeboten werden, der solche Requests annimmt und daraufhin ein entsprechendes Warn-Event generiert (Anforderung C4.4).

Storage Strategy
Die *Storage Strategy* überwacht die Auslastung der Speicherorte. Die Storages werden vom *Storages-Modul* abgerufen und anschließend wird die Auslastung über den entsprechenden *Storage Client* abgefragt. Im Falle der Überschreitung von definierten Schwellwerten wird ein Warn-Event zurückgegeben (Anforderung C1.4).

Notification Client Strategy Interface
Das *Notification Client Strategy Interface* definiert die Schnittstelle für Strategien zum Versenden von Benachrichtigungen (Anforderungen C1.4, C3.5, C4.4).

Mailing Client Strategy
Die *Mailing Client Strategy* implementiert das *Notification Client Strategy Interface* und bietet eine Schnittstelle zum Versenden von Mails an (Anforderungen C1.4, C3.5, C4.4).

4.5 Laufzeitsichten
Laufzeitsichten beschreiben das konkrete Verhalten und die Interaktionen der Bausteine eines Systems zu wichtigen Use Cases und Features.

4.5.1 Nutzer-Workflow
Im folgenden Sequenzdiagramm wird anschaulich dargestellt, wie der Nutzer-Workflow im System funktioniert, von der Anlage des Nutzers, über Rechtevergabe, bis hin zur Deaktivierung des Nutzers.

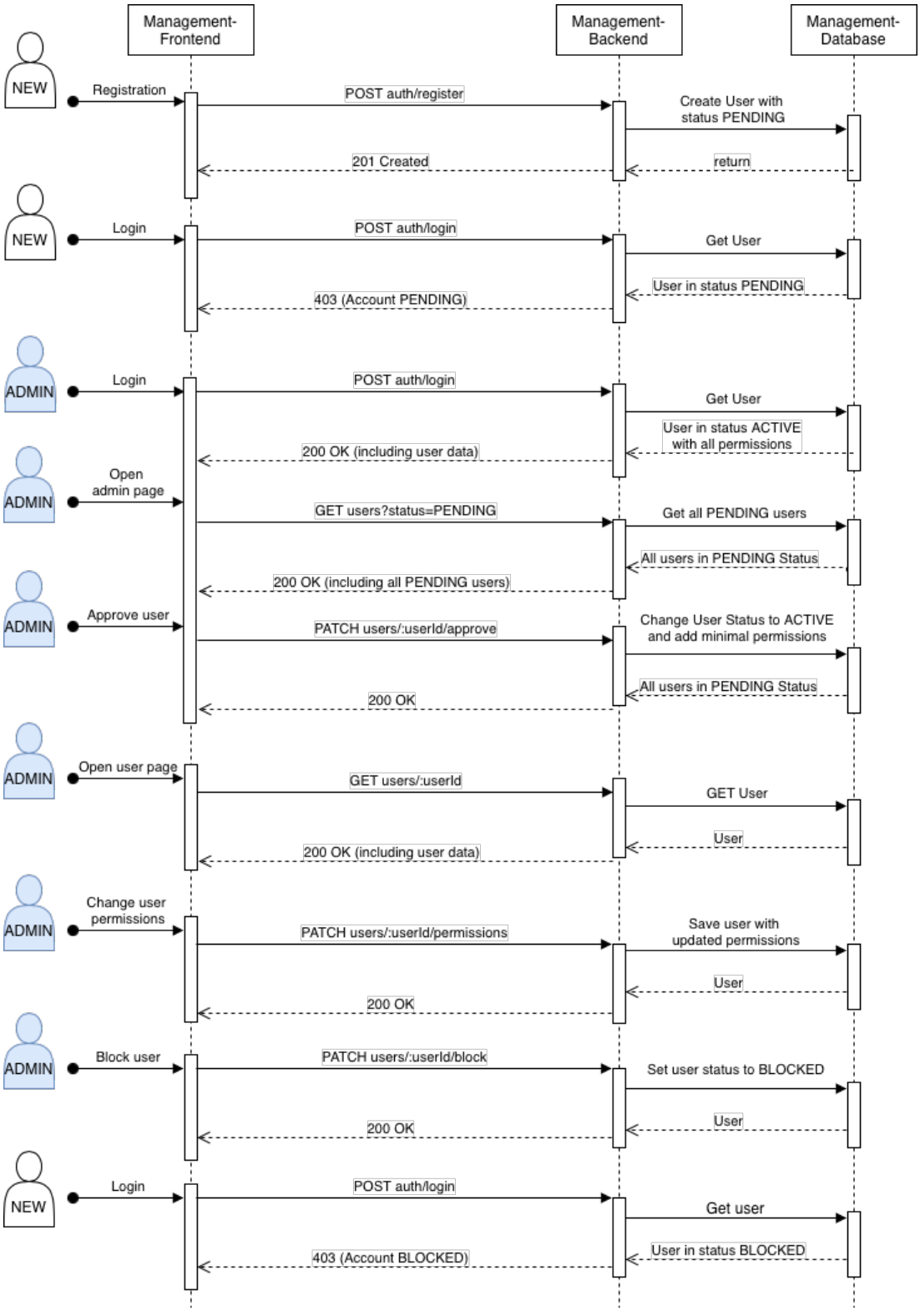

Abbildung 3: Laufzeitsicht - Nutzer-Workflow

Zunächst wird die Anlage neuer Nutzer betrachtet. Um den manuellen Aufwand für Admins bei der Nutzeranlage möglichst gering zu halten, wurde hierfür ein spezieller Workflow entwickelt.

Dabei kann sich jeder Nutzer zunächst eigenständig bei der Anwendung registrieren.

Daraufhin werden die Nutzerdaten an das Management-Backend gesendet und in der Datenbank mit dem Status *PENDING* gespeichert.
Meldet sich ein Nutzer mit dem Status *PENDING* an, so wird hierfür der Antwort-Code 403 zurückgegeben und eine entsprechende Fehlermeldung angezeigt.
Um Zugriff auf die Anwendung zu erhalten, muss ein Nutzer mit entsprechenden Rechten, im Folgenden Admin genannt, den neu registrierten Nutzer zunächst bestätigen. Daraufhin besitzt der Nutzer zunächst nur sehr minimale Rechte.
Wenn ein Nutzer bestätigt wurde, kann ein Admin weitere Rechte vergeben. Hierbei definiert er für jede Ressource, welche Rechte der Nutzer haben soll, beispielsweise die Rechte neue Runs zu erstellen (4.6.2 Authentifizierung und Autorisierung).
Wenn ein Nutzer schließlich keinen Zugriff mehr auf die Anwendung haben soll, so kann der Admin ihn blockieren, woraufhin bei Login-Versuchen der Fehlercode 403 zurückgeliefert wird und der Nutzer eine Fehlermeldung erhält.

4.5.2 Schedule-Erstellung

Zur besseren Verständlichkeit wird, beispielhaft an Argo Workflows, der Ablauf bei Erstellung einer zeitgesteuerten Ausführung (Schedule) grafisch, anhand eines Sequenzdiagramms, beschrieben.

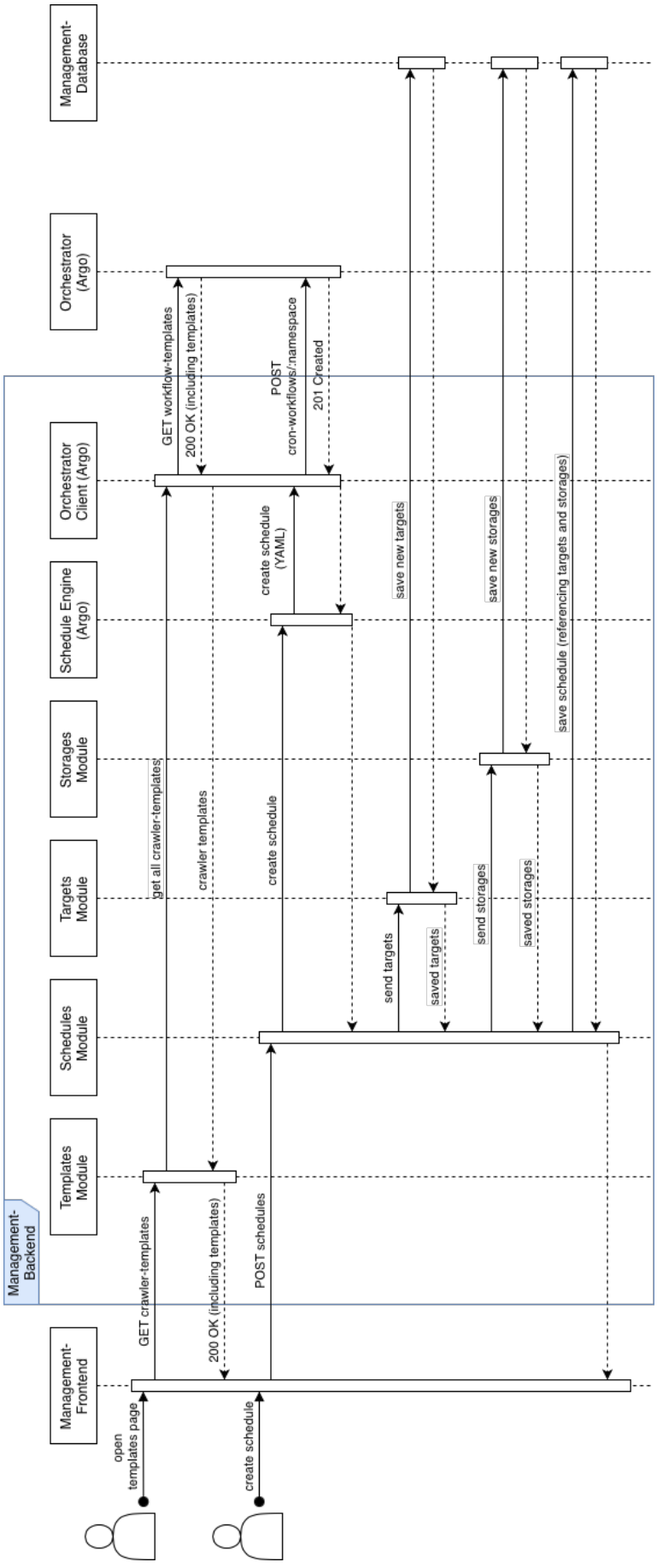

Abbildung 4: Laufzeitsicht - Schedule-Erstellung

Zur Erstellung einer Schedule muss der Nutzer zunächst die Crawler Template Übersichtsseite aufrufen. Dadurch werden alle Crawler Templates vom Backend und somit im Beispiel von Argo Workflows, alle Workflow Templates vom Argo Server abgerufen.
Anschließend erstellt der Nutzer für eines der Templates eine Schedule, wodurch die notwendigen Informationen ans Backend geschickt und dort schließlich verarbeitet werden.
Im Backend nimmt das *Schedules-Modul* die Anfrage entgegen und gibt die Informationen zur Erstellung der Schedule zunächst an die Schedule Engine weiter, die im Falle von Argo Workflows aus den Informationen ein gültiges YAML-Dokument baut und dieses anschließend an den Argo Server zur Speicherung schickt.
Darauffolgend extrahiert das *Schedules-Modul* aus den Parametern alle Target- und Storage-Parameter und schickt diese an das jeweilige Modul zur Speicherung in der Datenbank.
Schlussendlich speichert das *Schedules-Modul* die Schedule dann selbst noch in der Datenbank inklusive der Referenzierungen auf die erkannten Targets und Storages.

4.5.3 Alarmierungs-Workflow

Im folgenden Sequenzdiagramm wird beispielhaft der Alarmierungsablauf, angefangen vom Abonnieren eines bestimmten Ereignisses durch einen Nutzer, bis hin zur Alarmierung über den angegebenen Alarmierungsweg, dargestellt.

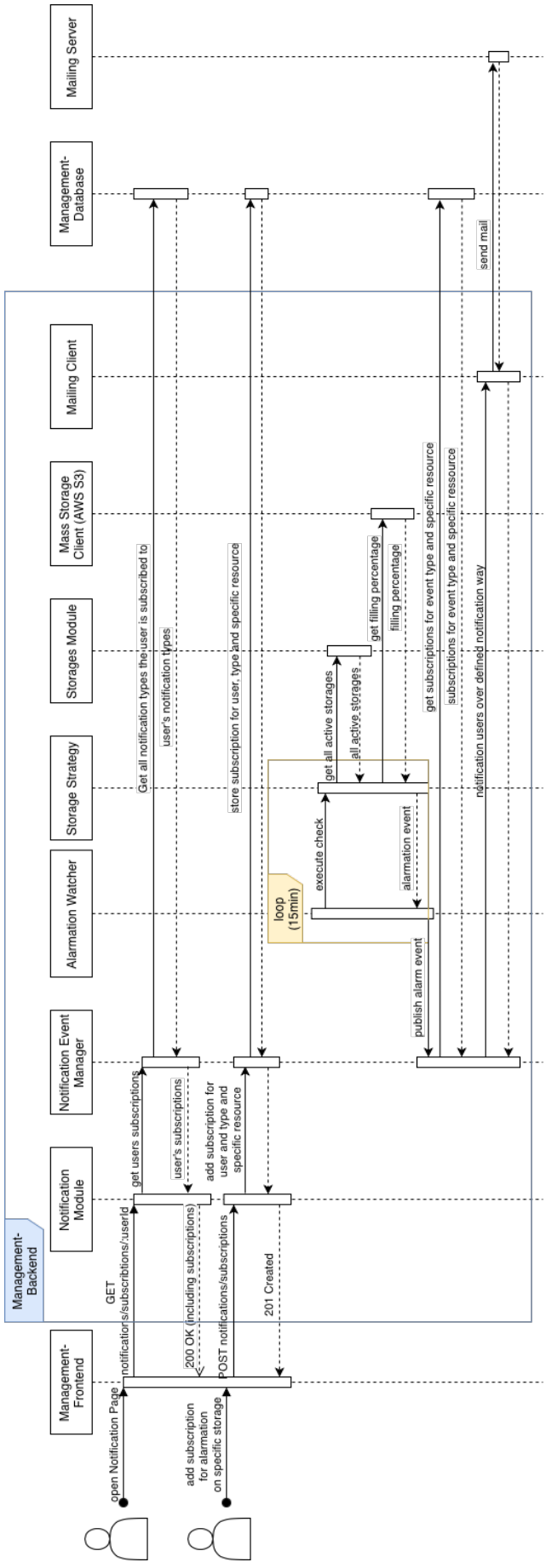

Abbildung 5: Laufzeitsicht - Alarmierungs-Workflow

Zunächst ruft der Nutzer die Übersichtsseite der Benachrichtigungen auf, auf welcher alle seine aktuellen Benachrichtigungsabonnements aufgelistet sind. Diese Informationen werden über das *Notification-Modul* des Backends abgerufen, welches diese wiederum über den *Notification Event Manager* aus der Datenbank erhält.
Anschließend abonniert der Nutzer im Frontend eine weitere Alarmierung, in diesem Beispiel ein bestimmtes Storage. Diese Informationen werden über einen API-Request an das *Notification-Modul* des Backends geschickt und daraufhin über den *Notification Event Manager* in der Datenbank gespeichert.
In regelmäßigen Abständen schickt der *Alarmation Watcher* Befehle zur Überprüfung an alle registrierten Alarmierungsstrategien. So erhält auch die Storage-Strategie einen solchen Befehl und holt sich daraufhin vom *Storages-Modul* alle aktiven Storages und überprüft für alle Speicherorte, über den jeweiligen *Storage Client,* den Füllstand dieser. Im Falle der Überschreitung eines definierten Schwellenwerts wird für den jeweiligen Storage ein Alarmierungsevent an den *Alarmation Watcher* zurückgegeben, welcher daraufhin ein Event an den *Notification Event Manager* schickt.
Der *Notification Event Manager* fragt aus der Datenbank alle Nutzer ab, die den Benachrichtigungstyp und die angegebene Ressource abonniert haben. Anschließend schickt er über den gewählten Alarmierungsweg des Nutzers, beispielsweise per Mail, eine entsprechende Benachrichtigung.

4.6 Querschnittskonzepte

4.6.1 Logging

Logging ist ein zentrales Querschnittskonzept einer jeden Anwendung und essenziell, um einen stabilen Betrieb sowie die Wartbarkeit und Nachvollziehbarkeit der Anwendung zu unterstützen, und ist daher auch explizit als Anforderung der Priorität A (A1.5) formuliert.
Insbesondere im Fehlerfall ist umfassendes Logging unabdingbar, um aufgetretene Fehler und Probleme ausführlich nachvollziehen und analysieren zu können.
Für die Umsetzung der folgenden Logging-Strategie wird der, in NestJS integrierte, Logger verwendet, da dieser nativ in das verwendete Framework integriert ist und zudem verschiedene Log Level anbietet.

Logging-Strategie

Ziel der Logging-Strategie ist es, ausreichende Informationen für den Betrieb und die Analyse bereitzustellen, ohne dabei jedoch übermäßig viele oder große Logmengen zu generieren, welche die Nachvollziehbarkeit wieder verschlechtern würden.
Aufgrund dessen werden die Logs in verschiedene Level unterteilt und so genau definiert, welche Ereignisse welches Log Level erfordern.
Die Logeinträge werden anhand ihrer Dringlichkeit in die Log Level debug, log, warn und error unterteilt.

Debug:
Logs der Kategorie **Debug** dienen hauptsächlich der Entwicklung und der Analyse von Problemen in dieser Phase. Dabei enthalten sie umfassende technische Informationen, die im Produktivbetrieb nicht benötigt werden. Dazu zählen unter anderem:

- Generierte Objekte und interne Datenstrukturen
- Ausgeführte Datenbankoperationen
- Detailinformationen zu internen Verarbeitungsschritten

Logs dieser Kategorie werden in der Regel im Produktiv-Log nicht angezeigt.

Log:
Das Log Level **Log** wird für reguläre und erwartete Systemereignisse genutzt und bildet daher den Großteil der Logeinträge ab. Hierfür gelten folgende Vorgaben:

- Jeder ankommende API-Request muss geloggt werden, inklusive aufgerufenem Endpunkt, Request Payload (ohne sensible Daten) und ID des anfragenden Nutzers.
- Jeder ausgehende Request an ein externes System muss geloggt werden, inklusive Pfad des aufzurufenden Endpunkts, Request Payload (ohne sensible Daten) und Response Code.
- Start und Ende automatisierter Prozesse (z.B. Synchronisationsvorgänge) müssen geloggt werden.
- Schreiboperationen in die Datenbank müssen geloggt werden, jedoch ohne Payload.
- Sensible Daten wie Passwörter, API-Keys oder Tokens dürfen in keinem Falle geloggt werden.

Warn:
Logs der Kategorie **Warn** beschreiben Auffälligkeiten, die jedoch die korrekte und vollständige Abarbeitung des Requests nicht verhindern. Beispiele hierfür sind:

- Abweichungen vom erwarteten Ablauf ohne unmittelbaren Fehler (z.B.: Vorheriger Synchronisationsvorgang ist noch nicht abgeschlossen).
- Fehlende Parameter, die jedoch die Abarbeitung der Anfrage nicht verhindern (z.B.: Crawler Template bei Run nicht referenziert).

Error:
Logs der Kategorie **Error** beschreiben Fehler, welche die korrekte und vollständige Abarbeitung des Requests verhindern. Dazu zählen beispielsweise:

- Fehlgeschlagener Datenbankverbindungsversuch
- Fehlgeschlagene Requests an externe Systeme
- Unbehandelte Exceptions

Insbesondere Logs der Kategorien **Warn** und **Error** müssen erweiterte Informationen enthalten, welche die Nachvollziehbarkeit von Fehlern erleichtern, wie beispielsweise das zu bearbeitende Objekt oder Teile des aktuellen, für das Problem relevanten, Systemzustands.

4.6.2 Authentifizierung und Autorisierung

Authentifizierung und Autorisierung sind zentrale sicherheitsrelevante Querschnittskonzepte der Anwendung und stellen sicher, dass ausschließlich berechtigte Personen oder Systeme Zugriff auf die Anwendung erhalten.
Zudem muss überprüft werden, dass Personen bzw. Systeme nur Zugriff auf die Ressourcen bekommen, für die sie berechtigt sind.
Da sowohl Personen als auch externe Systeme mit dem Management-Backend interagieren müssen, bedarf es hierfür verschiedener Mechanismen.
Die im Folgenden vorgestellte Authentifizierungs- und Autorisierungsstrategie dient zur Erfüllung der Anforderungen A2.1, A2.2 und C2.1.

4.6.2.1 Authentifizierung und Autorisierung im Backend

Authentifizierung

Authentifizierung beschreibt den Prozess eines Systems, in welchem überprüft wird, ob eine Identität (Benutzer oder System) diejenige ist, für die sie sich ausgibt. (Kosinski, Clark, & Scapicchio, 2026)
Für die Authentifizierung wird PassportJS[14] verwendet, da diese Bibliothek verschiedenste Authentifizierungsstrategien anbietet und zudem von NestJS für die Authentifizierung empfohlen wird.
Differenziert werden muss nun im Folgenden zwischen der Authentifizierung von Nutzern und der von externen Systemen, wie beispielsweise den Crawlern.

Frontend-Nutzer:

Das Frontend authentifiziert Nutzer über **Hypertext Transfer Protocol** (HTTP) **only Bearer Tokens.** Diese werden im Management-Backend ausgestellt und im Browser als HTTP-only Cookies gespeichert. Dieses Vorgehen ist für den Schutz vor Cyberangriffen von enormer Bedeutung, da JavaScript diese Art von Cookies nicht auslesen kann und dadurch Cross Site Scripting (XSS) Angriffe verhindert werden. (MDN, 2025)
Bei jedem API-Request wird hierbei das Bearer Token mitgesendet und anschließend durch die PassportJS Bearer-Strategie validiert. Nach erfolgreicher Validierung wird der Nutzerkontext dem Request hinzugefügt und steht den empfangenden Komponenten zur Verfügung (6.1.1 Authentifizierung und Autorisierung).

[14] https://www.passportjs.org

Externe Systeme (Crawler):
Externe Systeme authentifizieren sich über **API-Keys**, da es sich hierbei um technische Clients ohne interaktive Anmeldung handelt. Jedes externe System besitzt hierfür einen eindeutigen API-Key, der bei jedem Request übermittelt und durch die PassportJS API-Key Strategie validiert wird. Die API-Keys werden verschlüsselt in der Datenbank gespeichert und besitzen zur Dokumentation eine Beschreibung des Nutzungskontexts.

Autorisierung
Autorisierung beschreibt auf welche Ressourcen ein Nutzer zugreifen kann oder welche Handlungen er ausführen darf. (auth0, o. D.)
Für die Autorisierung und damit verbundene feingranulare Rechteverwaltung wird **CASL**[15] eingesetzt. CASL ermöglicht dabei die Definition von sogenannten Subjects und Actions, wobei ein Subject eine Ressource beschreibt (z.B. Crawler Run) und Action eine Aktion für eine Ressource (z.B. Erstellen). Nach der Authentifizierung durch PassportJS werden die übergebenen Rechte aus dem Nutzerkontext in sogenannte CASL Abilities übersetzt, anhand derer mithilfe von CASL die Rechteprüfung stattfindet.

Da sowohl Rechteprüfung als auch Authentifizierung meist direkt im Controller stattfinden, werden explizite Dekoratoren für die Authentifizierung und Rechteprüfung entwickelt, sodass die Überprüfung einfach, mithilfe von Annotationen am jeweiligen Endpunkt stattfinden kann (6.1.1 Authentifizierung und Autorisierung).

4.6.2.2 Authentifizierung und Autorisierung im Frontend
Auch im Frontend spielen Authentifizierung und Autorisierung eine entscheidende Rolle, um Schutz vor unberechtigtem Zugriff zu bieten.
Die übergeordnete Authentifizierung findet hierbei in einem Authentifizierungskontext statt, der bei Initialisierung der Anwendung das aktuelle Nutzerprofil vom Backend abruft. Ist diese Anfrage erfolgreich, so stehen dem Frontend die Nutzerinformationen inklusive der Rechte des Nutzers zur Verfügung. Schlägt der Request jedoch fehl, so ist kein Nutzer angemeldet, und es wird lediglich eine statische Willkommens-Seite angezeigt.
Die Autorisierung im Frontend findet ebenfalls über CASL statt, was die Wartbarkeit maßgeblich erhöht.
Zudem wird ein Auth Guard erstellt, der überprüft, ob ein Nutzer angemeldet ist, und falls nicht auf die Login-Seite weiterleitet. Ebenso wird ein Permission Guard entwickelt, welcher mithilfe von CASL überprüft, ob der Nutzer die geforderten Rechte hat und falls nicht eine Fehlermeldung anzeigt. Die beiden Guards werden in den entsprechenden Seiten genutzt und übernehmen so zentral die Rechteüberprüfung, sodass die eigentliche Komponente lediglich die fachliche Logik enthält.

[15] https://casl.js.org/v6/en/

4.6.3 Orchestrator-Abstraktionsschicht

Um Anforderung A1.4 zu erfüllen, sodass ein Wechsel des Orchestrierungs-Frameworks keine vollständige Neuentwicklung des Management-Systems erfordert, sondern lediglich die Neuentwicklung eines, für den Orchestrator passenden, Clients, muss eine Abstraktionsschicht eingeführt werden.
Wie bereits im Kapitel 4.3.5 Orchestrator-Unabhängigkeit beschrieben, liegt diese Abstraktionsschicht im Backend. Die Benennung von Ressourcen vor dieser Abstraktionsschicht wird, wie folgt, definiert:

- **Crawler Run:** Als Crawler Run wird die eindeutige Ausführung eines Crawlers beschrieben.
- **Crawler Template:** Als Crawler Template wird die nicht parametrisierte Schablone eines Crawlers bezeichnet, die durch Parametrisierung ausgeführt werden kann und dadurch bei jeder Ausführung einen Crawler Run erzeugt.
- **Schedule:** Eine Schedule beschreibt die zeitgesteuerte regelmäßige Ausführung eines Crawler Templates mit Parametrisierung.

Betrachtet man die orchestrator-abhängigen Verantwortlichkeiten, so ergeben sich die folgenden Aufgaben:

- Kommunikation mit dem Orchestrator
- Umwandlung von, im Frontend, erstellten Crawler Templates in das Format des Orchestrators
- Umwandlung von, im Frontend, erstellten Schedules in das Format des Orchestrators

Nach dem Single Responsibility Principle aus den SOLID Prinzipien, soll eine Komponente lediglich eine Verantwortung haben (Martin, 2003, S. 95), woraus sich für die drei Verantwortungen drei einzelne Komponenten ergeben. Um die Unabhängigkeit zwischen Business-Logik und konkreter Implementierung der orchestrator-abhängigen Logik zu gewährleisten, werden hierfür drei Interfaces definiert, welche die, von den orchestrator-spezifischen Komponenten, zu implementierenden Schnittstellen definieren.
Das erste Interface ist das *Orchestrator Client Interface* und definiert die Schnittstellen zur Kommunikation mit dem Orchestrator.
Das zweite Interface ist das *Template Engine Interface*, welches die Schnittstelle zur Erstellung von Crawler Templates im Orchestrator bietet. Im Falle von Argo Workflows muss hierbei beispielsweise ein YAML-Dokument aus den Parametern gebaut werden.
Hinzu kommt noch das dritte Interface, das *Schedule Engine Interface*, welches die Schnittstelle zur Erzeugung von Schedules definiert. Im Falle von Argo Workflows muss auch hier wieder ein YAML-Dokument gebaut werden.

4.6.4 Disaster Recovery

Ein gutes Konzept für Disaster Recovery ist unabdingbar, um Zuverlässigkeit und Verfügbarkeit sicherzustellen. Ziel dabei ist es, die Anwendung nach Abstürzen ohne Datenverlust wieder in einen konsistenten Zustand zu überführen und damit Anforderung A1.2 zu erfüllen.

Ein Großteil des Systemzustands, wie Crawler Templates, Schedules und Crawler Runs, wird vom Orchestrator verwaltet und somit auch durch diesen persistiert. Im Falle von Argo Workflows werden diese Daten im etcd des Kubernetes Clusters und in der Argo eigenen Datenbank gespeichert. Ein möglicher Datenverlust durch Absturz des gesamten Kubernetes Clusters wird durch Sicherung des etcds des Clusters verhindert.

Der anwendungseigene Systemzustand, wie beispielsweise Nutzerdaten und API-Keys, wird in einer eigenen PostgreSQL Datenbank gespeichert und ist dadurch gegen Datenverlust bei Ausfällen geschützt.

Der Betrieb der Anwendung, sowie der zugehörigen Datenbanken und auch Argo Workflows erfolgt vollständig containerisiert im Kubernetes Cluster und wird über Helm Charts deployt. Ausgefallene Pods werden dabei automatisch neu gestartet, ohne dass ein manueller Eingriff notwendig ist. Die Daten der Datenbank werden in einem persistenten Volume gespeichert, sodass diese auch über einen Absturz hinaus gesichert sind.

Durch die Kombination aus der, durch Argo Workflows umgesetzten, Persistierung im etcd des Clusters und der persistenten Speicherung der Backend Daten in einer eigenen Datenbank, sowie der automatischen Neustart-Mechanismen von Kubernetes ist sichergestellt, dass die Anwendung nach Absturz in einem konsistenten Zustand ohne Datenverlust automatisch neustartet.

4.6.5 Generische Alarmierungen

Um Alarmierungen anzubieten und damit die Anforderungen C1.4, C3.5, C4.4 zu erfüllen, braucht es ein Alarmierungskonzept. So soll das einfache Hinzufügen von weiteren Alarmierungswegen sowie Alarmierungs-Events ermöglicht werden.

Zudem sollen beliebige Nutzer hinzugefügt werden können, die dann die Wahl haben, für welche Alarmierungen sie benachrichtigt werden möchten.

4.6.5.1 Entkopplung von Alarmierungsquellen und Alarmierungsempfängern

Für die Entkopplung von Alarmierungsquellen und Alarmierungsempfängern bietet sich das Publisher-Subscriber Pattern an.

Geeignet ist dieses Architekturmuster insbesondere für Systeme mit Ereignissen, ohne eine enge Kopplung zwischen Ereigniserzeugern (Publisher) und Ereignisempfängern (Subscriber) entstehen zu lassen. (Azure, 2026)

Publisher erzeugen dabei Events, ohne zu wissen, welche Komponenten diese konsumieren. Subscriber abonnieren bestimmte Event-Typen, ohne die Publisher zu kennen. Für die

Entkopplung sorgt dabei ein generischer Message Broker, der ein Event von den Publishern empfängt und anhand des Event-Typs an die abonnierenden Subscriber weitergibt.
In diesem System stellt der *Notification Event Manager* den Message Broker dar und ist daher die zentrale Komponente des Patterns. Dieser verwaltet auch die Subscriber, in diesem Fall die Nutzer, inklusive der Event-Typen, die sie abonniert haben, sowie die gewünschten Alarmierungswege.
Als Publisher fungieren mehrere Komponenten im System, zum aktuellen Zeitpunkt das *Crawler Run Metrics-Modul*, das *Orchestrator Sync-Modul* und das *Alarmation Watcher-Modul*, wobei jederzeit weitere Publisher hinzukommen können. Die Publisher erzeugen dabei strukturierte Alarm-Events und übergeben diese an den *Notification Event Manager,* ohne Kenntnis über die Alarmierungsempfänger zu haben.
Beim Eintreffen eines Events ermittelt der *Notification Event Manager* anhand der Datenbank die gewünschten Empfänger und leitet die Alarmierung über den angegebenen Kanal weiter (z.B. über den *Mailing Client*).
Ein Vorteil des Einsatzes dieses Design Patterns ist insbesondere die geringe Kopplung zwischen Publisher und Subscriber, sodass Änderungen an der Alarmierungslogik sich nicht auf die Benachrichtigungslogik auswirken und umgekehrt. Ein weiterer Vorteil ist die hohe Erweiterbarkeit für neue Alarmierungsquellen und Benachrichtigungswege.
Einen Nachteil allerdings stellt der erhöhte Entwicklungsaufwand bei geringer Anzahl an Publishern und Benachrichtigungswegen dar. Die Vorteile der Nutzung des Design Patterns überwiegen jedoch gegenüber diesem Nachteil bei weitem.

4.6.5.2 Erweiterbarkeit von Alarmierungsquellen

Zur flexiblen Erweiterbarkeit der Alarmierungsquellen wird das Strategy Pattern eingesetzt. Dieses Entwurfsmuster eignet sich insbesondere für Problemstellungen, in denen verschiedene Implementierungen zur Laufzeit verwendet werden sollen, ohne die aufrufende Komponente an eine konkrete Implementierung zu koppeln. Eine Strategie kapselt dabei einen Algorithmus oder ein Verhalten und ermöglicht es, dieses unabhängig von den verwendenden Clients zu variieren. (Gamma, Helm, Johnson, & Vlissides, 1993, S. 17)
In diesem Kontext wird das Strategy Pattern genutzt, um verschiedene Arten von Überprüfungen, wie beispielsweise Überlastung der Warteschlange oder Storage-Auslastung, durchzuführen und dabei flexibel erweiterbar für weitere Überprüfungen zu sein.
Dabei fungiert der *Alarmation Watcher* als zentrale Komponente für die Alarmierungsüberprüfungen. Im *Alarmation Watcher* sind alle Alarmierungsstrategien registriert. Der *Alarmation Watcher* führt diese dann periodisch aus und leitet erkannte Warn-Events an den *Notification Event Manager* weiter. Dabei kennt der *Alarmation Watcher* ausschließlich die Schnittstelle der Strategien (*Alarmation Strategy Interface*) und ist daher von der konkreten Implementierung der Strategien entkoppelt.

Das *Alarmation Strategy Interface* definiert die Schnittstelle, welche die Aufruflogik enthält und jede Alarmierungsstrategie implementieren muss.
Vorteile der Verwendung des Strategy Patterns in diesem Kontext sind die einfache Erweiterbarkeit um Alarmierungsstrategien und die geringe Kopplung zum *Alarmation Watcher*, welcher lediglich an das Interface gebunden ist. Ebenso wird das Single Responsibility Principle eingehalten, indem jede Strategie ihre eigene Verantwortung hat und der *Alarmation Watcher* die Zeitsteuerung übernimmt.
Nachteile des Strategy Patterns in diesem Kontext sind der erhöhte Implementierungsaufwand durch das Interface und die festgelegte Aufrufstruktur, wobei die Vorteile diesen Nachteilen deutlich überlegen sind.

4.6.5.3 Beispiel: Erweiterung um eine weitere Alarmierungsüberprüfung

Um die hohe Erweiterbarkeit des beschriebenen Systems zu veranschaulichen, wird im Folgenden der Aufwand dargestellt, der durch Hinzufügen einer weiteren Alarmierungsüberprüfung entstehen würde.
Zum einen wird eine *Alarmation Strategy* benötigt, die das *Alarmation Strategy Interface* implementiert und die Überprüfungslogik umsetzt. Zum anderen muss die Strategie im *Alarmation Watcher,* beispielsweise durch Dependency Injection registriert werden, wobei dies je nach konkreter Implementierung auch automatisch passieren kann. Anschließend muss noch ein neuer Event-Typ hinzugefügt werden, den die Nutzer abonnieren können.
Ohne Verwendung der genannten Patterns wäre die Alarmierungslogik zentral umgesetzt und Erweiterungen würden Veränderungen an der bestehenden Logik erfordern, was gegen das Open-Closed Principle (Meyer, 1997) verstößt und zu einem erhöhten Fehlerrisiko bei der Wartung führt.
Durch die Nutzung der Patterns wiederum sind Erweiterungen an der Logik unabhängig von bisherigen Überprüfungen und die klare Trennung von Verantwortlichkeiten ist gegeben.

4.6.5.4 Erweiterbarkeit um Alarmierungswege

Auch die Alarmierungswege sollen leicht erweiterbar sein. Zum aktuellen Zeitpunkt ist hierfür lediglich die Alarmierung per Mail vorgesehen. Um jedoch nach dem Open-Closed Principle für Erweiterung offen zu sein, wird auch hier das Strategy Pattern angewandt. Dabei wird ein *Notification Client Strategy Interface* definiert, welches alle Clients für Alarmierungswege implementieren müssen. Alle Strategien für die angebotenen Alarmierungswege werden dabei beim *Notification Event Manager* registriert, welcher im Alarmierungsfall dann, entsprechend der Auswahl des Nutzers, die gewünschte Alarmierungsstrategie anwendet.

4.6.6 Konsistenter Umgang mit API-Requests im Frontend

Die Kommunikation mit dem Management-Backend ist ein bedeutendes Querschnittskonzept, da es nahezu alle Funktionen des Frontends betrifft.

Ziel des Querschnittskonzepts für API-Request im Frontend ist es, eine Basis zu schaffen, die bereits einfache Grundmechanismen für konsistente Datenhaltung, Fehlerhandlung, Retry-Mechanismen und Caching bietet. Als Grundlage dafür wird **TanStack Query**[16] in Verbindung mit **Axios**[17] verwendet.

TanStack Query abstrahiert dabei die Verwendung der, von Axios ausgeführten, HTTP-Requests stark und lässt den Entwickler auf die fachliche Logik konzentrieren. So orchestriert TanStack Query die Bereiche Laden der Daten, automatisches Refetching, Fehlerzustände, Cache Management und Retry-Mechanismen komplett eigenständig.

4.6.6.1 Caching im Frontend

Caching spielt eine zentrale Rolle in Bezug auf Latenz bei der Interaktion mit dem Backend. Es unterstützt dabei die Netzwerkanfragen an das Backend zu reduzieren und damit sowohl die Netzwerklast zu verringern als auch die Last auf das Backend. Caching bringt jedoch auch das Risiko von Inkonsistenz in Bezug auf Daten mit sich, weswegen hierfür ein spezielles Konzept notwendig ist. Durch den Einsatz von TanStack Query wird ein clientseitiger Cache aufgebaut.

Die Caching-Strategie basiert dabei auf drei Konzepten: Die Identifikation von Ressourcen über Query Keys, die Aktualisierungsstrategie mittels Polling und die Invalidierung bei Schreibzugriffen.

Jede API-Anfrage wird durch einen eindeutigen Schlüssel, den Query Key identifiziert und im Cache abgelegt. Ein solcher Schlüssel ist dabei hierarchisch als Array aufgebaut (z.B.: *[„schedule“, namespace, name]*). Dadurch wird nicht nur das Caching spezifischer Ressourcen, sondern auch die Adressierung gesamter Ressourcengruppen ermöglicht.

Komplexere Abfragen, wie beispielsweise die Abfrage von Listen mit Filterparametern (z.B. Crawler Runs), beinhalten dabei die Filterparameter im Query Key, um sicherzustellen, dass unterschiedliche Parametrisierungen der Abfrage separat gecacht werden und nicht auf denselben Cache zugreifen.

Bei der Anzeige von Crawler Runs und auch anderen Ressourcen ist die Aktualität der Daten von enormer Bedeutung. Die Daten müssen somit laufend aktualisiert werden.

Hierfür muss zwischen statischem Polling und dynamischen Polling unterschieden werden. Statisches Polling bedeutet die Aktualisierung und die damit verbundene erneute API-Anfrage in periodisch definierten Zeitintervallen. Diese Art des Pollings ist beispielsweise für Schedules oder Crawler Templates sinnvoll, die sich seltener ändern.

Bei Ressourcen, wie beispielsweise die Metriken zu Crawler Runs, die sich während ihrer Ausführung ständig ändern, muss eine höhere Polling-Frequenz gewählt werden. Wenn

[16] https://tanstack.com/query/latest
[17] https://axios-http.com/docs/intro

jedoch ein Run bereits beendet ist, so ändern sich seine Metriken nicht mehr und eine hohe Polling-Frequenz würde zu unnötigen API-Anfragen führen. Die Polling-Frequenz wird in solchen Fällen somit dynamisch, in diesem Falle anhand der Phase des Runs bestimmt.
Ein weiterer Fall, in welchem die Daten einer Ressource ungültig werden, tritt dann ein, wenn eine Ressource verändert wird. Wird beispielsweise eine Schedule gelöscht, so stimmt die aktuell gecachte Liste an Schedules nicht mehr. Um diese Problemstellung zu lösen, wird das Invalidation Pattern eingesetzt. Wird eine Ressource verändert (Mutation), so wird deren Query Key invalidiert und damit auch der Cache Eintrag. Dies triggert ein erneutes Holen der Daten, um wieder einen konsistenten Zustand zu erreichen. Diese Methodik funktioniert jedoch nur, wenn die Ressource auch auf demselben Client geändert wurde.
Zusammengefasst wird durch die beschriebene Caching-Strategie ein Kompromiss zwischen Aktualität der Daten und Netzwerkauslastung getroffen. Durch die Polling-Strategie und die feingranulare Steuerung des Cachings über Query Keys minimiert sich das Risiko von veralteten Daten, während das dynamische Polling eine möglichst echtzeitnahe Erfahrung bietet, ohne den Server mit unnötigen Anfragen zu überlasten.

4.6.6.2 Fehlerbehandlung und Retry-Strategie

Die Fehlerbehandlung von API-Requests soll ebenfalls als Querschnittskonzept definiert werden, um möglichst ähnliches Verhalten bei verschiedenen Komponenten zu erreichen. So sollen Retries nicht unkontrolliert durchgeführt werden, sondern explizit über eine Refetch-Logik gesteuert werden. Hierfür wird zwischen retry-qualifizierten und retry-unqualifizierten Fehlertypen unterschieden. Retry-unqualifizierte Fehlertypen sind alle, deren HTTP Response Code zwischen 400 und 500 liegt, da dies Client-Fehler sind und somit höchstwahrscheinlich beim nächsten Request wieder auftreten würden. Alle anderen Fehler sind für Retry qualifiziert und die betroffenen Anfragen werden daher nach einer sinnvollen Wartezeit wiederholt.

4.6.7 Fehlerbehandlung im Frontend

Um ein robustes Fehlerhandling mit aussagekräftigen Fehlermeldungen im Frontend zu erreichen, bedarf es einer expliziten Strategie.
Fehlerhandling wird hierarchisch in verschiedenen Ebenen betrieben.
Übergeordnet wird hierfür auf höchster Ebene eine globale Fehler Komponente (Error Boundary) definiert, welche alle, nicht auf tieferen Ebenen gefangenen, Fehler behandelt.
Ein solcher Fehler wird sowohl geloggt als auch dem Nutzer als Meldung mit dem angegebenem Fehlertext angezeigt.
Fehler, die bei der Kommunikation mit dem Backend über API-Requests auftreten, werden durch einen zentralen HTTP-Client behandelt. Tritt ein solcher Fehler auf, wird, je nach Fehlerursache, der Fehler geloggt und dem Nutzer mit entsprechender Fehlermeldung angezeigt.

Neben technischen Fehlern, wie beispielsweise der Abbruch einer Verbindung, können auch fachliche Fehler auftreten. Wie mit diesen Fehlern umzugehen ist, weiß lediglich die fachliche Komponente, die in diesen Fällen auch die Fehlerbehandlung übernimmt. Ein Beispiel hierfür stellt die Eingabe eines falschen Passworts beim Login dar.
Durch die klare Trennung von Fehlern in verschiedene Fehlerhierarchien und -verantwortungen, stellt die Anwendung eine umfassende Fehlerbehandlung bereit und somit eine stabile, sowie für den Nutzer verständliche Fehlerkommunikation sicher.

4.6.8 Ordnerstruktur im Repository

Ein Querschnittskonzept, welches zwar weniger Einfluss auf die Anwendung an sich hat, aber dennoch von enormer Bedeutung für die Entwicklungsarbeit ist, ist die Ordnerstruktur im Repository. Ohne ein definiertes Ordnerkonzept sinkt die Wartbarkeit eines Projekts stark.
Frontend und Backend befinden sich dabei in zwei unterschiedlichen Ordnern, die im Folgenden getrennt voneinander beschrieben werden.

4.6.8.1 Frontend

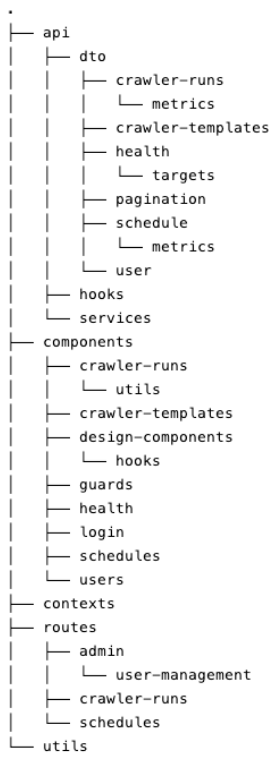

Abbildung 6: Ordnerstruktur-Frontend

Auf höchster Ebene im Ordner der Quelldateien gibt es fünf Unterordner, wobei der Ordner *api* alle relevanten Komponenten zur Kommunikation mit dem Backend enthält. Er wird dabei weiter unterteilt in *dto* mit allen Objekten, die vom Backend kommen, *hooks* für alle Hooks und *services* mit allen Service Komponenten, die Methoden anbieten, welche in API-Requests an das Backend übersetzt werden.

Im Ordner *components* befinden sich alle grafischen Komponenten, unterteilt nach Domäne. Eine Besonderheit stellt hierbei der *design-components* Ordner dar, der ausschließlich nicht fachliche Design-Komponenten enthält, die meisten davon aus der Design-Bibliothek shadcn. Der Ordner *guards* unter *components* enthält die bereits beschriebenen Guards für die Rechteprüfung der Nutzer.
Übergeordnete Komponenten, wie der Auth Context oder auch die Error Boundary, befinden sich im Ordner *contexts*.
Die für das Routing benötigten Dateien sind im Ordner *routes* zu finden (4.7.1 Routing im Frontend).
Im Ordner *utils* befinden sich noch allgemeine nicht grafische Komponenten, die wiederverwendbare Logik enthalten.

4.6.8.2 Backend

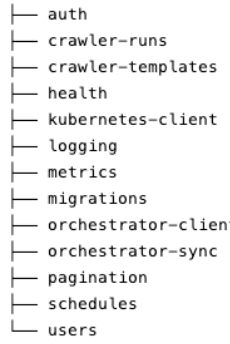

Abbildung 7: Ordnerstruktur-Backend

Im Backend wiederum findet die Aufteilung nach Domäne bereits auf oberster Ebene statt. Zudem gibt es auf dieser Ebene bereits einige technische Ordner.
Die Verantwortlichkeiten der einzelnen Module wurden bereits im Kapitel 4.4 Bausteinsicht beschrieben, weswegen hier nicht weiter darauf eingegangen wird.
Die einzelnen Ordner wiederum werden, wie folgt, weiter unterteilt:

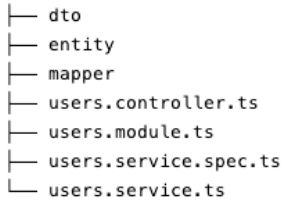

Abbildung 8: Ordnerstruktur-Backend Detailansicht

Zum einen liegen hier die NestJS typischen Komponenten, wie der Controller zur Bereitstellung der API-Endpunkte, der Service mit der eigentlichen Logik, das Module mit den Abhängigkeiten und exportierten Komponenten, sowie Tests. Zum anderen gibt es noch Ordner zur weiteren Strukturierung für *dto* (Data Transfer Objects), *entity* für die Datenbankobjekte und *mapper* für Umwandlungen von einem Objekt in ein anderes. Nicht in allen Modulen werden alle diese Ordnerunterteilungen benötigt, und ebenso gibt es in manchen Ordnern noch weiterführende Unterteilungen. Dieses Beispiel soll lediglich ein grobes Konzept für die Feingranularität der Ordner zeigen.

4.7 Architekturentscheidungen

In diesem Kapitel werden Architekturentscheidungen beschrieben, welche vor oder auch während der Entwicklung getroffen wurden. Die Entscheidungen werden dabei im Format von Architecture Decision Records[18] beschrieben, sodass sowohl der Kontext als auch resultierende Konsequenzen aus den Entscheidungen strukturiert dargestellt werden können.

4.7.1 Routing im Frontend

Kontext

Die Verwaltung von Routen im Frontend soll möglichst einfach, ohne die Notwendigkeit einer manuell gepflegten Konfigurationsdatei, geschehen.

Entscheidung

Für das Routing wird datei-basiertes Routing genutzt. Hierfür wird **TanStack Router**[19] gewählt.

Konsequenzen

Durch datei basiertes Routing entfällt die Notwendigkeit einer manuell gepflegten Konfigurationsdatei. Daraus resultiert eine Reduzierung des Boilerplate Codes.

4.7.2 Datenbank Migrationen

Kontext

In der agilen Entwicklung des Management-Backends werden immer wieder Anpassungen und Erweiterungen der Datenbankstruktur notwendig sein. Diese sollen zum einen nicht manuell durch SQL-Skripte durchgeführt werden und zum anderen nicht automatisch durch Analyse des Codes passieren, da dies zu erhöhtem Risiko von Datenverlust führt.

Entscheidung

Für Anpassungen und Erweiterungen an der Datenbankstruktur werden TypeORM Migrations[20] verwendet. Hierfür wird nach Änderungen an den Entitäten im Code ein Migrationsbefehl in der Kommandozeile ausgeführt, wodurch eine Migrationsdatei generiert wird, welche nach manuellem Review automatisch auf die Datenbank ausgeführt wird.

Konsequenzen

Zum einen entsteht durch die erstellten Migrationen nach Deployment ein konsistenter Zustand über alle Stages hinweg und zum anderen wird durch den manuellen Reviewprozess der Migrationsdatei ungewollter Datenverlust verhindert, ohne dass manueller Aufwand durch die Erstellung eines SQL-Skriptes entsteht.

[18] https://adr.github.io
[19] https://tanstack.com/router/latest
[20] https://typeorm.io/docs/migrations/why

4.7.3 Eigene Persistierung von Schedules

Kontext

Für Schedules sollen Fehler in Bezug auf Inkonsistenzen zu den referenzierten Crawler Templates angezeigt werden. Dabei soll jedoch nicht bei jedem Aufruf einer Schedule im Frontend die Überprüfung auf Inkonsistenzen stattfinden müssen, da dies zu unnötigem Overhead führt. Dies könnte bei vielen Schedules und gleichzeitigen Aufrufen potenziell hohe Last auf den Orchestrator und das Management-Backend bringen.

Entscheidung

Schedules werden bei Erstellung über das Management-Backend in der eigenen Datenbank persistiert, sodass dort die Fehler zu den Schedules gespeichert werden und die Überprüfung auf Inkonsistenzen periodisch stattfinden kann. Zudem speichert die Anwendung die Version des Crawler Templates bei der letzten Überprüfung und führt nur bei Veränderung dessen eine Überprüfung durch.

Konsequenzen

Durch das Speichern der Schedules in der eigenen Datenbank entsteht zwar erhöhter Speicherbedarf, jedoch sinkt der Overhead durch dynamische Überprüfungen und damit verbundener Last bei jedem Schedule Abruf enorm. Durch Speicherung der letzten überprüften Version der referenzierten Crawler Templates wird die Häufigkeit der notwendigen Überprüfungen weiter drastisch reduziert.

4.7.4 Synchronisation von Schedules mit dem Orchestrator

Kontext

Aufgrund der Entscheidung, Schedules auch in der eigenen Datenbank zu speichern, ist es nun essenziell, für konsistente Datenhaltung in Bezug auf die Schedules zwischen Orchestrator und eigener Anwendung, zu sorgen. Zwar sollten Schedules eigentlich ausschließlich über das Crawler-Management erstellt und somit in der Datenbank gespeichert werden, doch ist es theoretisch weiterhin möglich, Schedules über Custom Ressource Definitions direkt in Kubernetes zu erstellen.

Entscheidung

In regelmäßigen Abständen werden durch die Orchestrator-Sync-Komponente alle, dem Orchestrator bekannten, Schedules geholt und anhand derer, falls notwendig, in der eigenen Datenbank aktualisiert bzw. unbekannte Schedules hinzugefügt.

Konsequenzen

Durch die Synchronisation entsteht ein Mehraufwand, da diese regelmäßig stattfindet. Da jedoch Konsistenz zwischen Orchestrator und Management-System geboten sein muss, ist dieser Mehraufwand unabdingbar.

4.7.5 Generische Metriken

Kontext

Das Crawler-Management-System soll möglichst generisch und vollständig unabhängig vom Aufbau der Crawler sein, dabei jedoch trotzdem Metriken der Crawler anzeigen können (Anforderungen A5.1, A5.2).

Entscheidung

Für die Speicherung von Metriken der Crawler wird sowohl eine Schnittstelle mit vordefinierten Übergabeparametern, als auch eine mit generischen Übergabeparametern angeboten. Die generische Schnittstelle erwartet dabei die Parameter Name, Wert, Datentyp, Beschreibung und einen Zeitstempel der zu speichernden Metrik.

Konsequenzen

Durch die generische Schnittstelle für Metriken mit vordefinierten Übergabeparametern können ausführliche und spezifische Statistiken dargestellt werden. Durch die generische Schnittstelle können trotzdem beliebig viele andere unbekannte Metriken gespeichert und dem Nutzer angezeigt werden. Bei den unbekannten Metriken können hierbei jedoch nur einfache Statistiken angezeigt werden, da bei diesen der konkrete Kontext nicht bekannt ist.

4.7.6 Visualisierung des Crawler Aufbaus

Kontext

Für Anforderung C3.3 soll der Aufbau einzelner Crawler visualisiert werden. Da dieser jedoch unabhängig vom Crawler-Management-System sein soll und dieses daher den Aufbau der Crawler nicht kennt, kann diese Information nicht vom Crawler-Management-System selbst kommen.

Entscheidung

Jeder Crawler muss als direkter azyklischer Graph (DAG) darstellbar sein. Da ausschließlich der Crawler selbst seinen Aufbau kennt, wird eine Schnittstelle angeboten, über welche er seine Struktur dem System mitteilen kann. Dies funktioniert durch die Übergabe einer Liste von Knotenpunkten mit Vorgängern und Nachfolgern. Da jedoch auch der Crawler aus mehreren Containern bestehen kann, muss das Management-Backend damit umgehen können, diese Informationen von mehreren Teilen des Crawlers unabhängig voneinander zu erhalten und anschließend entsprechend zusammenzubauen.

Konsequenzen

Durch die generische Schnittstelle zur Übergabe von Crawlerstrukturen wird eine möglichst geringe Kopplung zwischen Crawler und Backend erreicht und damit auch die Unabhängigkeit zum Crawler-Aufbau garantiert. Weil das Backend jedoch auf Daten des Crawlers angewiesen ist, kann die Visualisierung nicht garantiert und zudem erst ab dem ersten Run eines Crawler Templates angeboten werden.

4.7.7 Initialer Nutzer

Kontext

Wie bereits in der Laufzeitsicht zum Nutzerworkflow beschrieben, können sich Nutzer selbst registrieren und müssen anschließend von einem Nutzer mit entsprechenden Rechten bestätigt werden. Dabei ergibt sich jedoch das Problem, wie der allererste Nutzer ins System kommt.

Entscheidung

Über Umgebungsvariablen werden Nutzername und Passwort eines initialen Nutzers mitgegeben, der beim Starten der Anwendung automatisch mit allen Rechten angelegt wird, sollte es nicht bereits einen Nutzer im System geben.

Konsequenzen

Durch die Anlage des initialen Nutzers wird sichergestellt, dass zu jeder Zeit mindestens ein Nutzer im System aktiv ist, der andere Nutzer bestätigen kann.

4.8 Risiken und Technische Schulden

Da sich diese Architekturdokumentation nicht auf den IST-Zustand der Implementierung bezieht, sondern eine geplante Zielarchitektur beschreibt, können hier keine technischen Schulden benannt werden. Diese werden im Kapitel 7 Evaluation dargestellt, welches die tatsächliche Umsetzung analysiert.

Im Folgenden werden somit ausschließlich Risiken betrachtet, die bei der aktuell geplanten generischen Architektur bestehen. Risiken, die sich aus der Nutzung eines spezifischen Orchestrators ergeben, werden ebenfalls im Kapitel 7 Evaluation erörtert.

4.8.1 Rechteverwaltung nur auf Ressourcentypen

Wie bereits beschrieben, findet die Rechteverwaltung auf Basis von Aktionen und Ressourcentypen statt. So kann beispielsweise definiert werden, dass ein Nutzer Schedules zwar erstellen, aber nicht löschen darf. Diese Rechte gelten jedoch jeweils global für alle Ressourcen dieses Ressourcentyps.

Eine feingranularere Unterteilung, wie beispielsweise auf Basis von Organisationseinheiten, ist in der aktuellen Architektur nicht vorgesehen.

Dadurch wird zwar die Komplexität der Berechtigungslogik reduziert, jedoch sind Szenarien mit mehreren unabhängigen Teams aufgrund der fehlenden Unterteilung in Organisationseinheiten nicht ideal umsetzbar. Hierfür wäre die Nutzung einer eigenen Instanz pro Team oder gute Abstimmungsmechanismen zwischen den Teams notwendig.

4.8.2 Überprüfbarkeit der Target-URLs ohne Authentifizierung

Die Überprüfung der Erreichbarkeit von Target-URLs findet in der aktuellen Architektur ohne die Möglichkeit zur Authentifizierung statt.

Sollte ein Target Server ausschließlich mit vorheriger Authentifizierung erreichbar sein, kann dieser in der aktuellen Architektur nur als eingeschränkt erreichbar gekennzeichnet werden.
Die Integration sämtlicher Authentifizierungsverfahren würde jedoch zu einer erheblichen Erhöhung des Entwicklungsaufwands führen. Dies steht damit in keinem angemessenen Verhältnis zum Nutzen dieser Funktion, da sie nur eine Hilfestellung darstellt und für die Kernfunktionen der Anwendung nicht von kritischer Bedeutung ist.

5 Design

In diesem Kapitel werden die Gestaltung und der Aufbau der umgesetzten Benutzeroberfläche vorgestellt. Wie bereits im Kapitel 3 Anforderungen beschrieben, konnten aus zeitlichen Gründen nicht alle Anforderungen vollständig umgesetzt werden. Die folgende Beschreibung bezieht sich somit ausschließlich auf die realisierten Komponenten. Wie in Anforderung A6.1 beschrieben, soll die Benutzeroberfläche für Nutzer ohne technisches Wissen einfach nutzbar sein, worauf bei ihrem Design besonderen Wert gelegt wurde.

5.1 Willkommensseite

Die Startseite der Anwendung gibt einen Überblick über die wichtigsten Funktionen sowie Hinweise für einen einfachen Einstieg in die Anwendung für neue Nutzer. Im rechten Bereich des Headers befindet sich zudem ein Login-Button, über welchen sich nicht authentifizierte Nutzer anmelden oder registrieren können.

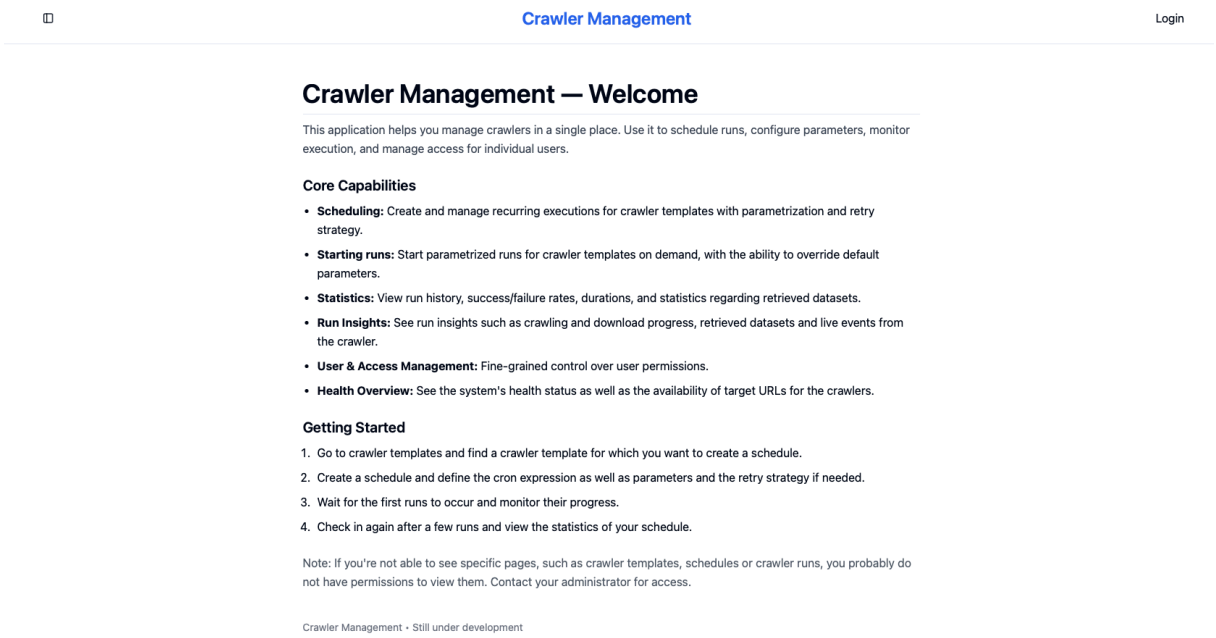

Abbildung 9: Crawler-Management Start Seite

5.2 Login-Seite

Die Login-Seite ist in zwei separate Tabs unterteilt, die sowohl die Anmeldung bestehender Nutzer als auch die Registrierung neuer Nutzer anbieten.
Fehler bei der Anmeldung oder Registrierung werden dabei deutlich durch eine klare Fehlermeldung hervorgehoben.
Im Registrierungs-Tab wird zudem der Registrierungsprozess erläutert (4.5.1 Nutzer-Workflow).

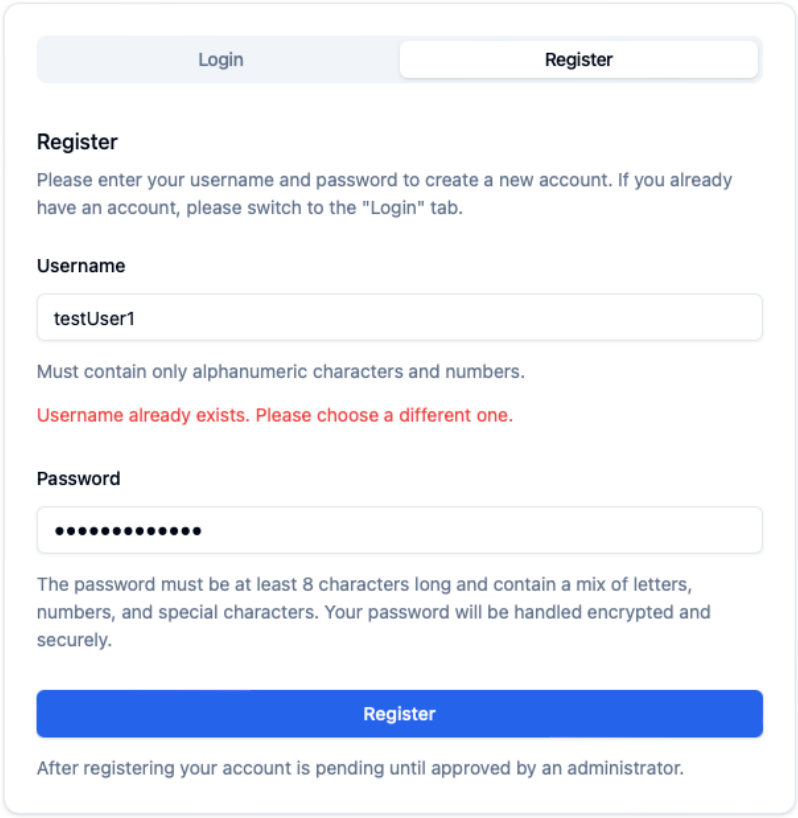

Abbildung 10: Crawler-Management Login-Page

5.3 Header

Nach erfolgreicher Anmeldung wird der Nutzer auf die Startseite weitergeleitet. Im Header wird nun eine Sidebar, der Titel der Anwendung, ein Status-Icon, die Initialen des Nutzers sowie ein Logout-Button angezeigt.
Das Status-Icon nimmt dabei je nach Gesundheitsstatus der Anwendung verschiedene Symbole und Farben an und kann dabei die Zustände verfügbar, eingeschränkt verfügbar und nicht verfügbar darstellen. Das Icon verlinkt dabei auf die Health Status Seite.

Abbildung 11: Crawler-Management Header

5.4 Sidebar

Die Sidebar stellt alle Seiten der Anwendung unterteilt nach Themen dar. Dabei werden ausschließlich die Seiten angezeigt, für die der Nutzer die jeweiligen Berechtigungen besitzt. Die einzelnen Menüpunkte werden im Folgenden detailliert beschrieben.

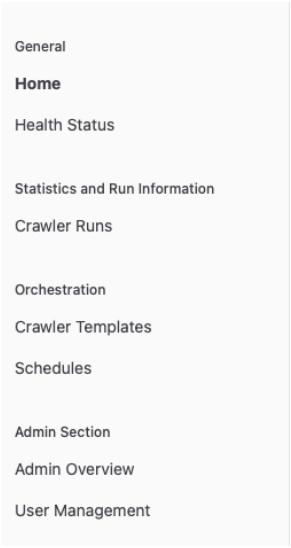

Abbildung 12: Crawler-Management Sidebar

5.5 Health Status Seite

Die Health Status Seite bietet eine Übersicht über den Gesundheitsstatus der Systemkomponenten sowie die Erreichbarkeit der, in den Schedules definierten, Target-Server.

In der oberen Hälfte wird der Status der Systemkomponenten dargestellt, darunter der Status des Kubernetes Clusters sowie des verwendeten Orchestrators. Die Daten werden dabei in regelmäßigen Abständen aktualisiert, können jedoch auch manuell über den bereitgestellten Refresh-Button aktualisiert werden.

In der unteren Hälfte wird der Health Status der Target-Server der Crawler angezeigt. Zunächst wird eine aggregierte Übersicht angezeigt, welche die Anzahl an verfügbaren, eingeschränkt verfügbaren und nicht verfügbaren Target-Servern anzeigt. Darunter befindet sich eine Detailansicht für alle Crawler Targets inklusive der Referenzen auf die Schedules, die diese Targets jeweils verwenden. Dadurch kann frühzeitig erkannt werden, welche geplanten Ausführungen bei einem Ausfall des Target-Servers betroffen wären.

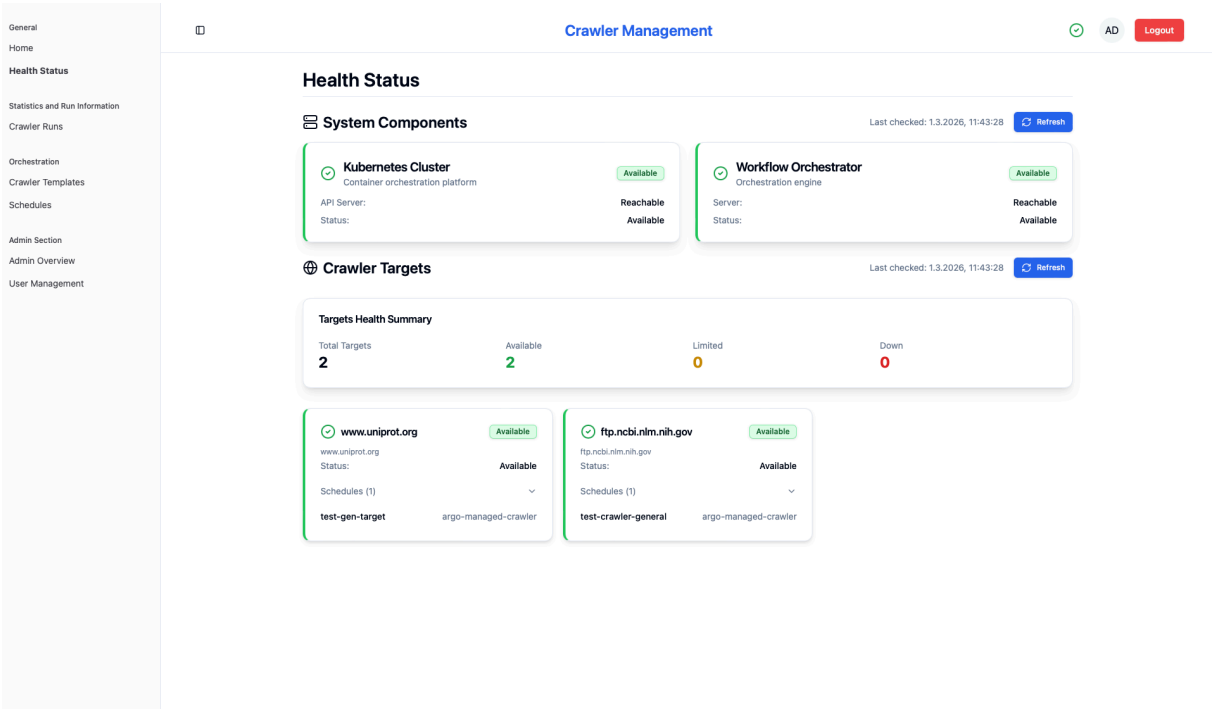

Abbildung 13: Crawler-Management Health Status Seite

5.6 Crawler Runs

5.6.1 Crawler Runs Übersichtsseite

Die Crawler Runs Übersichtsseite zeigt alle ausgeführten Runs an. Hierbei werden zunächst nur die neuesten Runs geladen, um die Netzwerkbelastung möglichst gering zu halten und den Client sowie den Server nicht zu überlasten. Bei Bedarf können weitere Runs über einen entsprechenden Button nachgeladen werden.

Zudem stehen Filtermöglichkeiten nach Namespace, Crawler Template, Schedule, Phase, sowie Start- und Enddatum bereit.

Pro Run werden die wichtigsten Informationen angezeigt und bei Klick auf eine Run-Zeile öffnet sich die Detailseite des Runs.

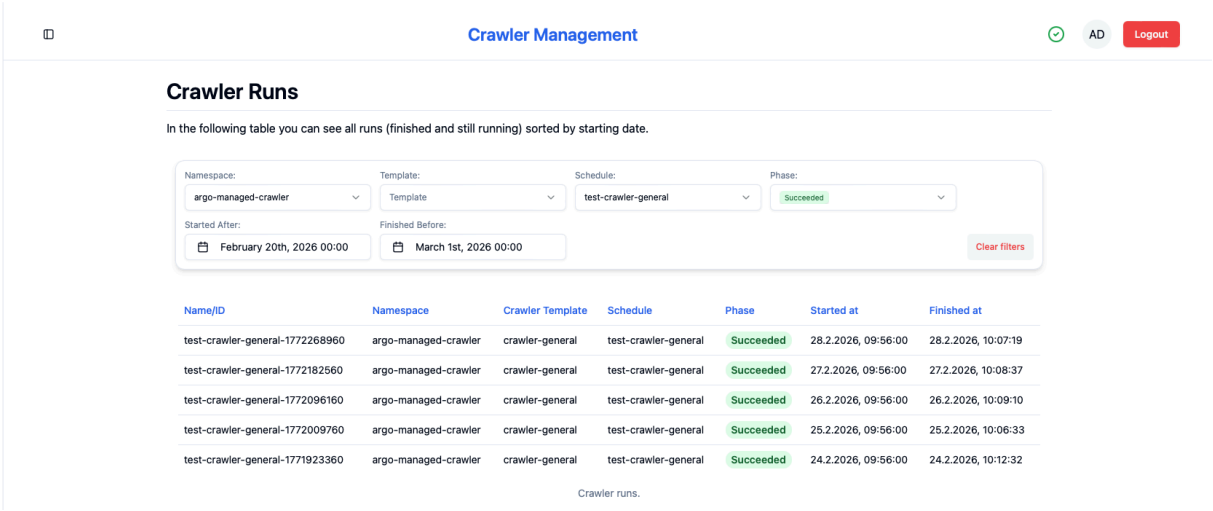

Abbildung 14: Crawler-Management Crawler Runs Übersichtsseite

5.6.2 Crawler Run Detailseite

Die Detailseite eines Crawler Runs zeigt sämtliche Informationen zum Run an.
Im oberen Bereich wird der Name des Runs, der Namespace, sowie die aktuelle Phase angezeigt. Darunter wird das zugehörige Template, sowie falls durch eine Schedule ausgelöst, die entsprechende Schedule mit Link darauf angezeigt.
Darunter folgen Informationen zu Start- und Endzeitpunkt des Runs, sowie dessen Dauer und die gesetzten Parameter.
Die Metriken des Runs sind in die zwei Bereiche Entdecken (Discovering) und Download von Datensätzen unterteilt, für die beide der jeweilige Fortschritt angezeigt wird.
Der Discovering-Bereich stellt unter anderem die Gesamtanzahl identifizierter Datensätze, Cache Hits, sowie die Anzahl neu entdeckter Datensätze dar.
Im Download-Bereich werden die Anzahl der zu herunterladenden Datensätze, sowie deren Größe und das bereits heruntergeladene Datenvolumen angezeigt.

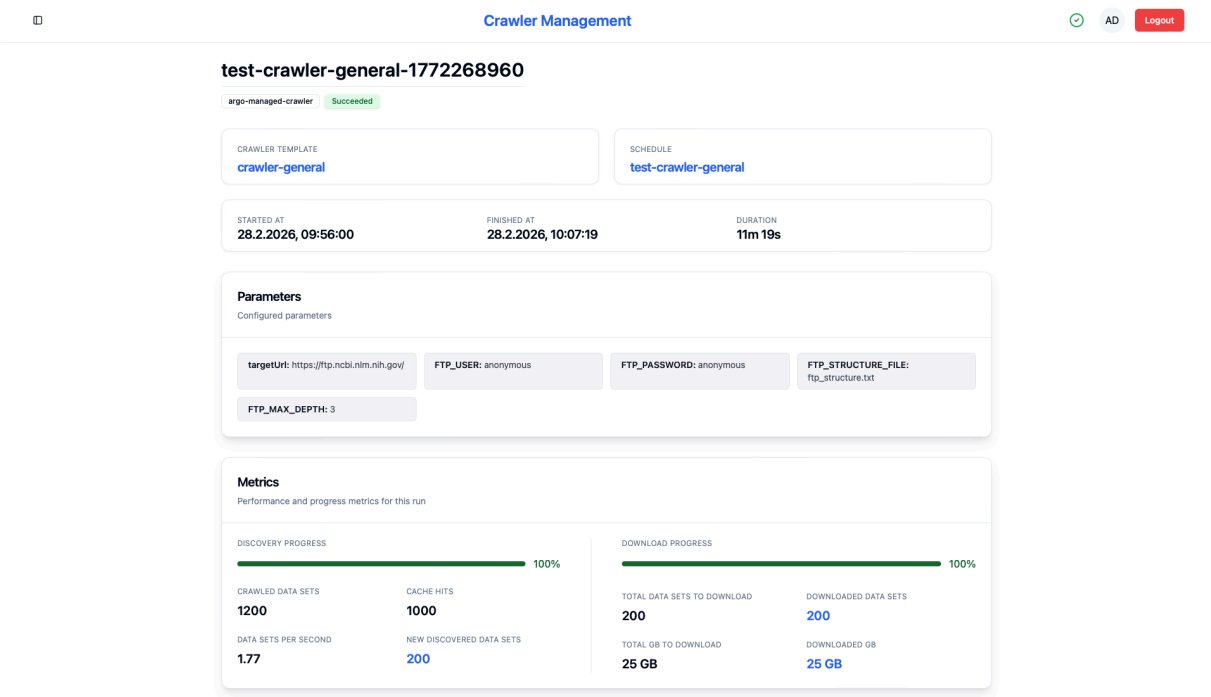

Abbildung 15: Crawler-Management Crawler Run Detailseite - Übersicht

Abschließend werden alle vom Crawler gesendeten Events nach Datum sortiert und nach Dringlichkeit kategorisiert.

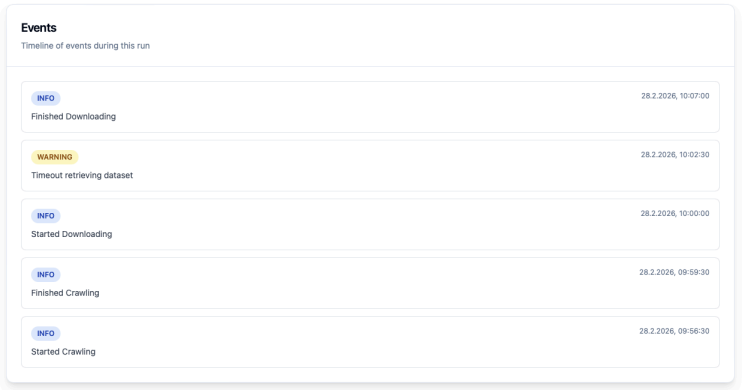

Abbildung 16: Crawler-Management Crawler Run Detailseite - Events

5.7 Crawler Templates Seite

Auf der Crawler Templates Übersichtsseite werden alle Crawler Templates mit Namen, Namespace, Version sowie dem Erstellungsdatum angezeigt. Pro Template stehen zudem zwei Aktions-Buttons zur Erstellung eines Runs sowie einer Schedule zur Verfügung.

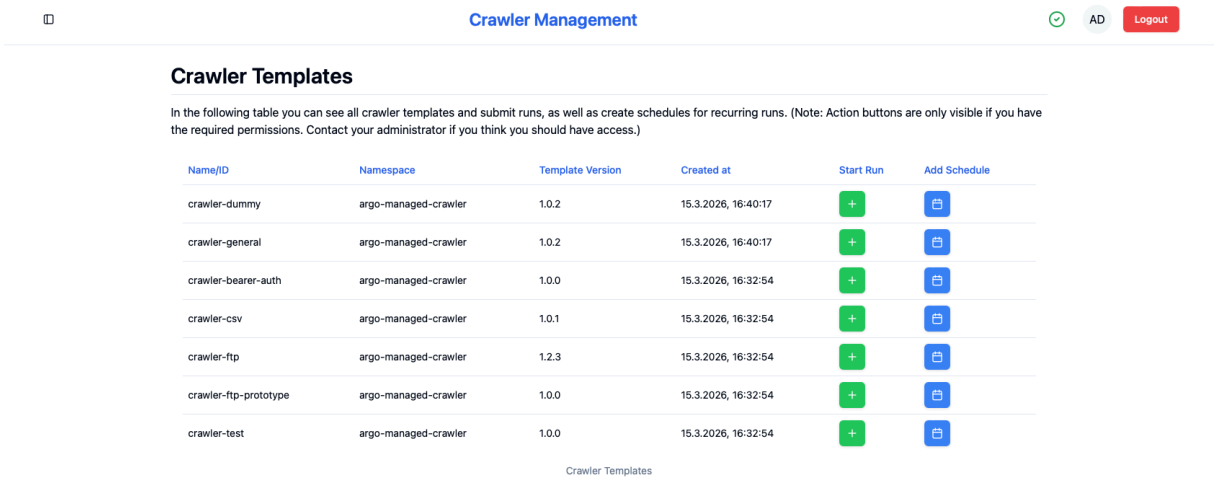

Abbildung 17: Crawler-Management Crawler Templates Übersichtsseite

Wird der Button zur Erstellung eines Runs betätigt, öffnet sich ein Dialogfenster zur Eingabe der Run Parameter.

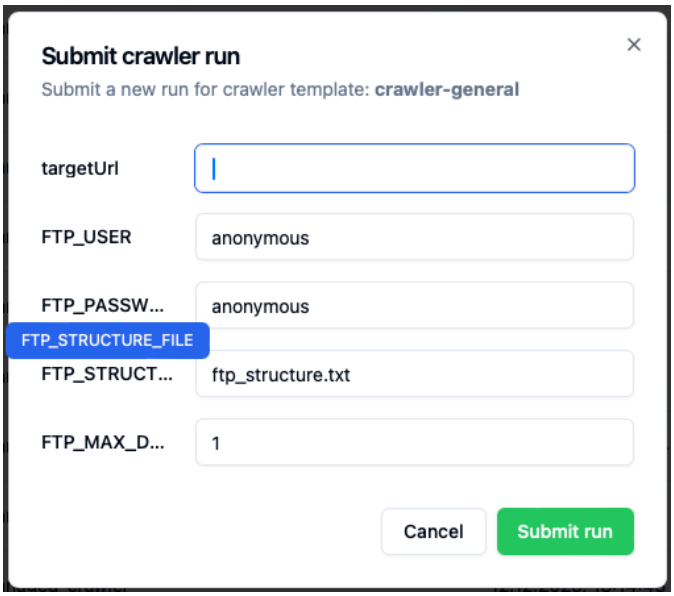

Abbildung 18: Crawler-Management Submit Crawler Run Dialog

Zur Erstellung einer Schedule wird zunächst ein Name benötigt, dessen Eindeutigkeit das Backend validiert. Anschließend definiert ein Cron-Ausdruck, der automatisch validiert und in ein menschenlesbares Format übersetzt wird, den Zeitplan der Schedule. Der Cron-Ausdruck wird, wenn nicht anders angegeben, in der Zeitzone des Nutzers interpretiert.
Zudem kann eine Retry-Strategie, einschließlich der Anzahl an Wiederholungsversuchen, Backoff-Dauer, Backoff-Faktor, maximaler Backoff-Zeit, sowie maximaler Gesamtwartedauer konfiguriert werden. Abschließend folgt die Definition der Parameter. Nach erfolgreicher Erstellung der Schedule wird der Nutzer zur Detailseite der Schedule weitergeleitet.

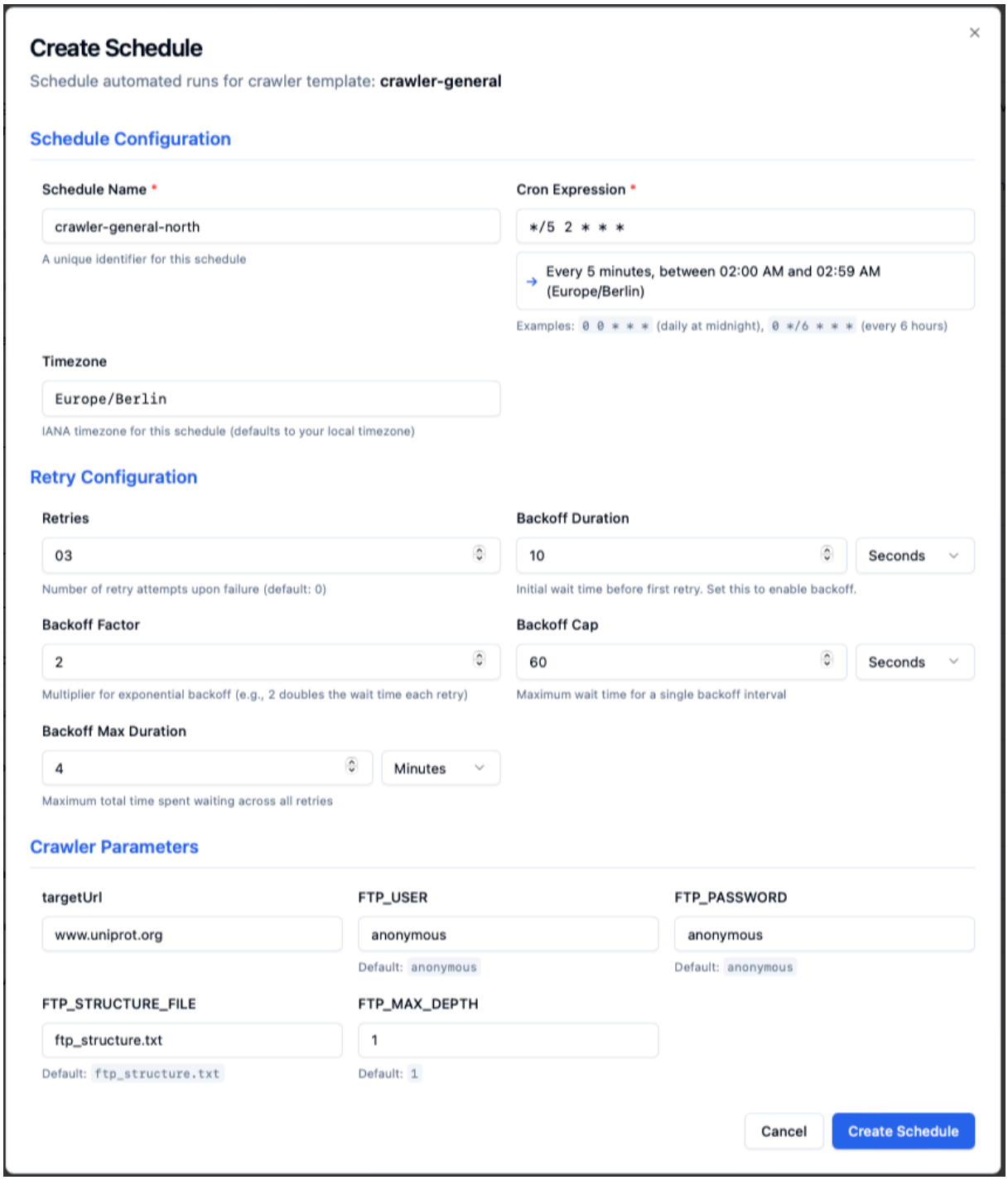

Abbildung 19: Crawler-Management Create Schedule Dialog

5.8 Schedules

5.8.1 Schedule Übersichtsseite

Auf der Schedule Übersichtsseite werden alle Schedules mit Namen, referenziertem Crawler Template, Namespace, lesbarem Zeitplanformat, nächstem Run, sowie Erstellungsdatum aufgelistet. Dabei zeigt die Seite zuerst alle aktiven Schedules.

Der Zeitplan wird dabei in der Zeitzone des Nutzers, der die Schedule erstellt hat, angezeigt, da eine Umrechnung von Cron-Ausdrücken in andere Zeitzonen nicht sinnvoll umsetzbar

ist. Der Zeitpunkt des nächsten Runs berechnet sich dabei anhand des Cron-Ausdrucks der Schedule und der lokalen Zeit des Nutzers. Bei Auswahl einer Schedule erfolgt eine Weiterleitung auf die Detailseite der Schedule

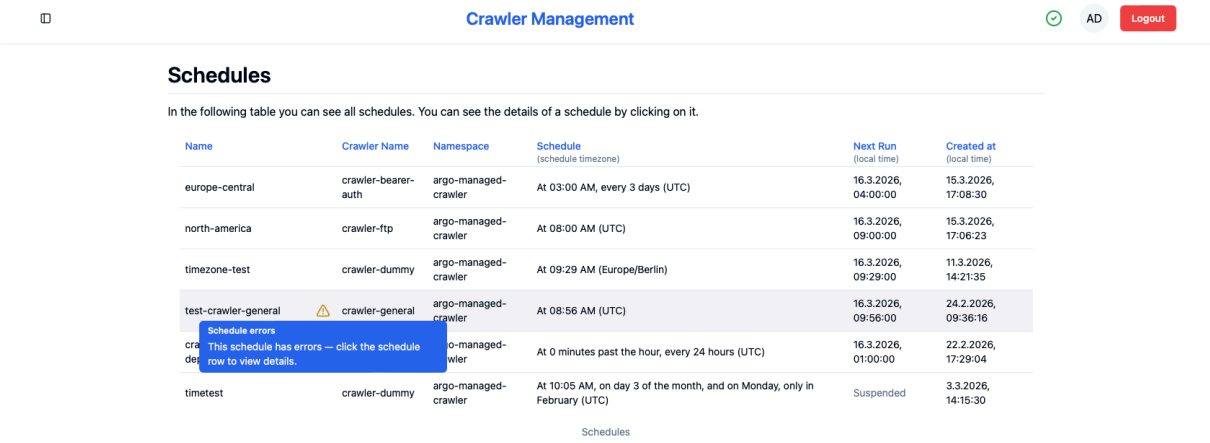

Abbildung 20: Crawler-Management Schedules Übersichtsseite

5.8.2 Schedule Detailseite

Die Detailseite einer Schedule zeigt oben den Namen, Namespace, sowie den Status der Schedule an und bietet Buttons zum Deaktivieren, Bearbeiten und Löschen der Schedule. Zudem werden Informationen zum Zeitplan in der erstellten Zeitzone, der Zeitpunkt der nächsten Ausführung in der Zeitzone des aktuellen Nutzers, sowie eine Statistik zu erfolgreichen und fehlgeschlagenen Runs angezeigt.
Falls Auffälligkeiten in Bezug auf die Schedule erkannt wurden, werden diese mit Titel, Beschreibung und Erkennungsdatum angezeigt.

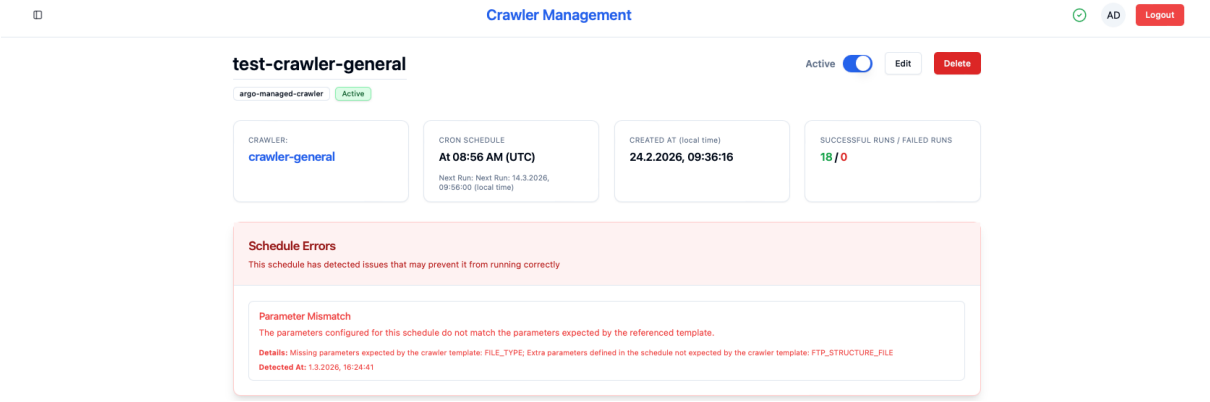

Abbildung 21: Crawler-Management Schedule Detailseite Übersicht

Darunter befindet sich eine Übersicht der Metriken aller Runs der Schedule mit der Anzahl an insgesamt gesammelten Datensätzen über alle Runs, aller neu gesammelten Datensätzen, sowie dem durchschnittlichen Anteil an neuen Datensätzen pro Run.
Zudem wird ein Diagramm angezeigt, bei dem zwischen den drei eben beschriebenen Metriken ausgewählt werden kann. Insbesondere der Anteil neu entdeckter Datensätze stellt ein aussagekräftiges Indiz für die Angemessenheit der gewählten Ausführungsfrequenz dar.

Ist der Wert sehr hoch, so sind die Daten auf dem Target-Server sehr volatil und die Aktualität der Daten wird möglicherweise nicht ausreichend sichergestellt, weswegen die Ausführungsfrequenz der Schedule erhöht werden sollte. Ist der Prozentsatz jedoch sehr niedrig, so könnte die Ausführungsfrequenz verringert werden, um Ressourcen zu sparen. Dies wird dem Nutzer mithilfe von Farben dargestellt. So ist der Bereich zwischen 10 und 20 Prozent grün eingefärbt, da solch ein Wert einen sinnvollen Anteil an neuen Datensätzen darstellt. Die Bereiche darüber und darunter sind gelb, bzw. rot eingefärbt, um dem Nutzer zu signalisieren, dass eine Anpassung der Ausführungsfrequenz notwendig sein könnte.
Der Nutzer kann für die Anzeige verschiedene Zeiträume auswählen.

Abbildung 22: Crawler-Management Schedule Detailseite Metriken

Der darunter liegende Abschnitt zeigt die Parameter der Schedule sowie die konfigurierte Retry-Strategie mit Erklärung zu den einzelnen Werten.

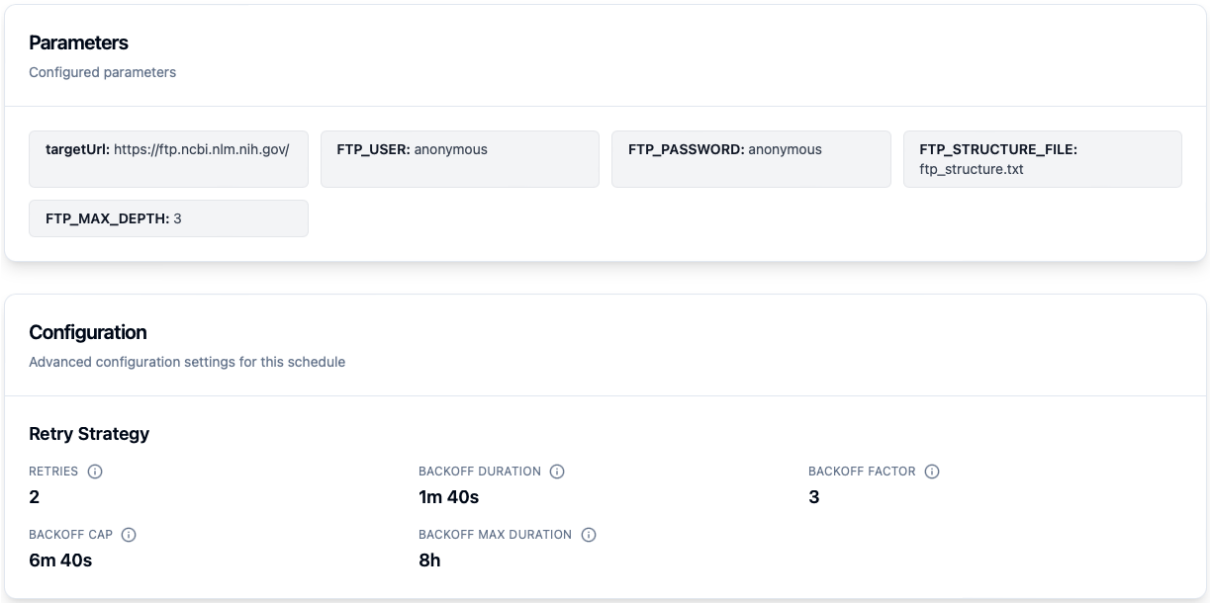

Abbildung 23: Crawler-Management Schedule Detailseite Konfiguration

Zuletzt werden alle zugehörigen Runs der Schedule seitenweise mit Filtermöglichkeiten angezeigt.

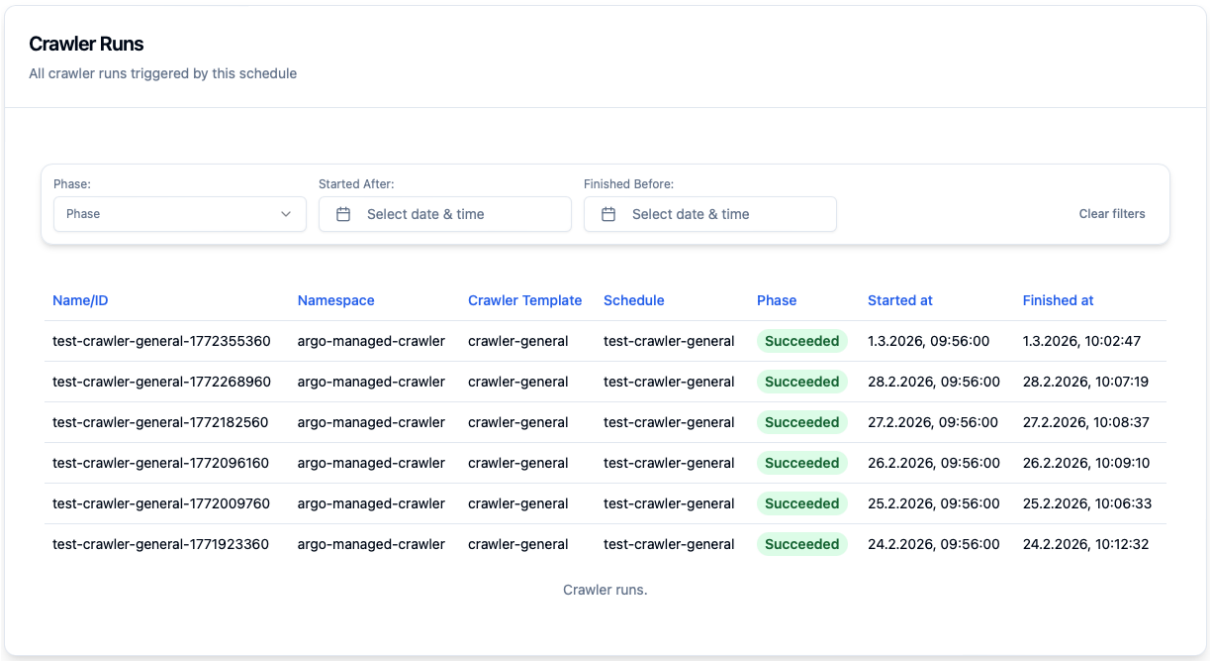

Abbildung 24: Crawler-Management Schedule Detailseite Runs

5.9 Nutzermanagement

5.9.1 Admin Übersichtsseite

Die Admin Übersichtsseite soll zentrale, für Admins wichtige Informationen anzeigen. Zum jetzigen Zeitpunkt befindet sich hier ausschließlich eine Übersicht aller Registrierungsanfragen mit der Möglichkeit zur Genehmigung oder Ablehnung.

Abbildung 25: Crawler-Management Admin Übersichtsseite

5.9.2 Nutzermanagement Seite

Auf der Nutzermanagement Seite werden alle Nutzer mit Status und Erstellungsdatum angezeigt. Administratoren können Nutzer genehmigen, blockieren oder deren Berechtigungen über eine separate Seite anpassen.

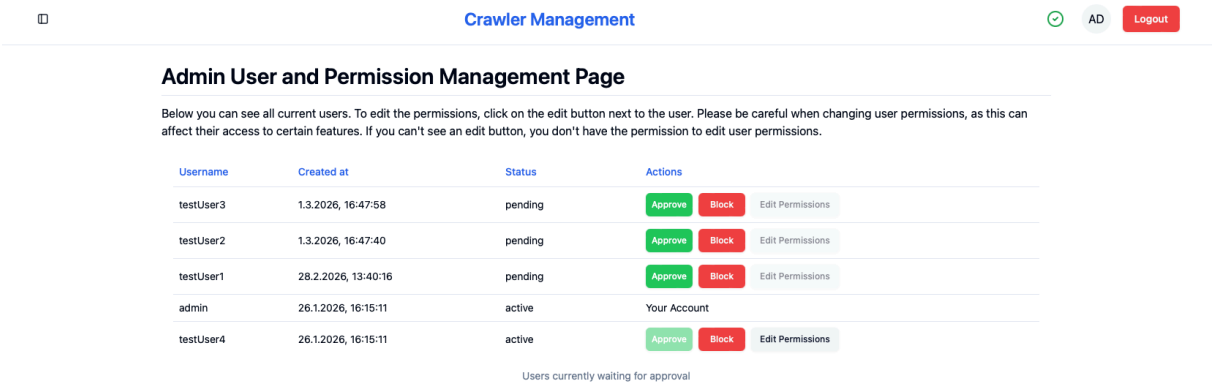

Abbildung 26: Crawler-Management Nutzermanagement Seite

5.9.3 Nutzer Berechtigungsmanagement

Auf der Berechtigungsmanagementseite können die Berechtigungen eines Nutzers pro Ressourcentyp und Aktion gesetzt werden, was die Grundlage für eine feingranulare Rechteverwaltung darstellt.

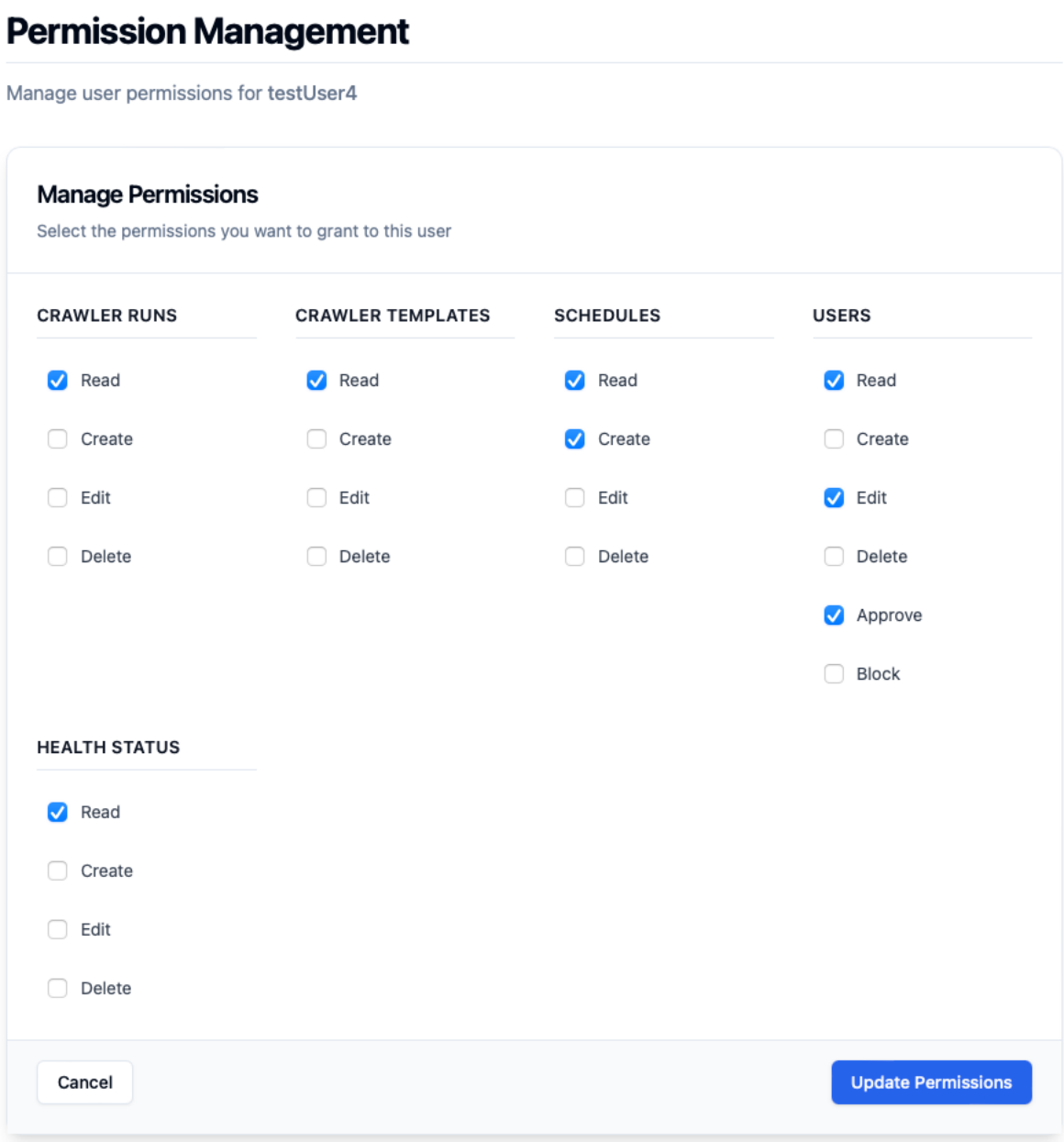

Abbildung 27: Crawler-Management Berechtigungsmanagement Seite

6 Implementierung

Im diesem Kapitel werden ausgewählte Aspekte der Implementierung des entwickelten Systems beschrieben. Da der Gesamtumfang der Codebasis jedoch mehr als 80.000 Zeilen umfasst, wird hier lediglich auf zentrale Implementierungsentscheidungen und besondere Konzepte eingegangen.

6.1 Management-Backend

6.1.1 Authentifizierung und Autorisierung

Die Umsetzung von Authentifizierung und Autorisierung in Kombination mit feingranularer Rechteverwaltung erfordert eine umfassende Logik. Ziel dabei war es, diese Logik zentral zu kapseln und somit von der eigentlichen Geschäftslogik klar zu trennen und damit dem Prinzip der Trennung von Belangen (Separation of Concerns) zu folgen, was sowohl die Wartbarkeit als auch die Erweiterbarkeit des Systems signifikant verbessert.
Hierfür wurde ein *Auth Service*, Authentifizierungsstrategien für verschiedene Authentifizierungsmethoden, eine *Ability Factory* zur Übersetzung von Berechtigungen, sowie Guards und Dekoratoren zur einfachen Integration in den NestJS Request-Ablauf implementiert.

Auth Service

Der *Auth Service* stellt sämtliche Funktionalitäten zur Authentifizierung von Nutzern, sowie von externen Systemen zur Verfügung und kapselt dabei sowohl Bearer Token als auch API-Key Verwaltung und Generierung.
Nutzerregistrierung und Login:
Zur Registrierung eines neuen Nutzers dient eine Methode, die Benutzername und Passwort des Nutzers erwartet. Daraufhin werden die Daten des Nutzers inklusive des gehashten Passworts über den *Users Service* in der Datenbank gespeichert. Der neu erstellte Nutzer erhält dabei initial den Status **PENDING** und benötigt zuerst eine Freischaltung durch einen Administrator, bevor der Zugriff auf die Anwendung gestattet wird. Dadurch wird sichergestellt, dass ausschließlich autorisierte Personen Zugriff auf die Anwendung erhalten, dennoch aber nicht alle Nutzer händisch von einem Administrator angelegt werden müssen (4.5.1 Nutzer-Workflow).
Für die Anmeldung eines Nutzers wird ebenfalls eine Methode angeboten, die zunächst die übergebenen Zugangsdaten des Nutzers validiert. Hierfür wird der entsprechende Nutzer über den *Users Service* aus der Datenbank geladen und anschließend das Passwort auf Übereinstimmung überprüft. Existiert der Nutzer nicht oder sind die Zugangsdaten ungültig, liefert die Anwendung den HTTP-Fehlercode 401 zurück. Existiert der Nutzer, ist jedoch nicht im Status **ACTIVE**, so wird der HTTP-Statuscode 403 zurückgegeben und die Anfrage dadurch abgelehnt.

Bei erfolgreicher Authentifizierung wird ein JSON Web Token (JWT[21]), mithilfe des *JWT-Service* von NestJS und einem per Umgebungsvariable übergebenen JWT-Secret, generiert. Dieses Token hat eine Gültigkeit von 30 Minuten und wird als HTTP-Only Cookie an das Response-Objekt angehängt. Dadurch wird ein Zugriff über clientseitiges JavaScript verhindert und das Risiko eine Cross Site Scripting Angriffs deutlich reduziert (4.6.2 Authentifizierung und Autorisierung).
Schlussendlich werden mithilfe der, im Folgenden noch vorgestellten, *Ability Factory* die Autorisierungsregeln (Ability Rules) des Nutzers erstellt und gemeinsam mit weiteren Nutzerdaten zurückgeliefert.
Externe Systeme:
Wie bereits im Architekturkapitel beschrieben, werden für die Authentifizierung von externen Systemen API-Keys verwendet. Ein API-Key ist ein eindeutiger geheimer alphanumerischer Code, der bei der Kommunikation mit einer Schnittstelle zur Authentifizierung dient. Einen solchen API-Key können ausschließlich Benutzer mit den dafür notwendigen Rechten erstellen. Ein generierter API-Key wird dann beispielsweise für einen Crawler zur Kommunikation mit dem Crawler-Management-Backend hinterlegt.
Zur Generierung eines API-Keys wird eine Methode angeboten, die den erstellenden Nutzer sowie eine Beschreibung erwartet. Daraufhin generiert das System einen zufälligen API Key, der gehasht in der Datenbank gespeichert und anschließend einmalig im Klartext zurückgeliefert wird. Eine erneute Abfrage eines vollständigen API-Keys ist aus Sicherheitsgründen nicht möglich. Falls der API-Key verloren geht, sollte dieser über eine hierfür angebotene Methode deaktiviert und anschließend ein neuer API-Key erstellt werden.
Zur Validierung eines API-Keys dient eine Methode, die diesen zunächst übergeben bekommt, hasht und anschließend mit allen aktiven gespeicherten API-Keys vergleicht. Existiert ein entsprechender Eintrag in der Datenbank, so wird das zugehörige API-Key-Objekt zurückgegeben und das Datum der letzten Nutzung des API-Keys in der Datenbank aktualisiert. Andernfalls wird *null* zurückgegeben.
Zudem steht eine Methode zur Verfügung, welche alle API-Keys, inklusive deren Metadaten, zurückliefert. Dabei wird aus Sicherheitsgründen nur ein Präfix von 16 Zeichen des Schlüssels geliefert.

Authentifizierungsstrategien
Die eigentliche Authentifizierungslogik übernimmt in großen Teilen PassportJS und wird über spezifische Strategy-Klassen integriert.
Für die Authentifizierung mit JWT wird zunächst der Token aus dem Request extrahiert und anschließend gemeinsam mit dem Secret Key, der als Umgebungsvariable definiert wurde, an den Konstruktor der Passport-Basisklasse übergeben.

[21] https://www.jwt.io

Innerhalb der *validate*-Methode wird die User-ID aus dem Token gelesen und der zugehörige Nutzer über den *Users Service* geladen. Sollte der Nutzer nicht existieren oder nicht den Status **ACTIVE** haben, so wird der HTTP-Statuscode 401 bzw. 403 zurückgegeben.
Für die Authentifizierung mittels API-Key existiert ebenfalls eine eigene Strategie. Hier wird der Konstruktor der Basisklasse von PassportJS mit dem Key des *Authorization Headers*, der den API-Key enthalten soll, aufgerufen. Zudem existiert eine Methode, die, mithilfe der bereits beschriebenen Validierungsmethode des *Auth Service,* den API-Key entsprechend validiert.

Ability Factory

Die *Ability Factory* stellt das Kernstück der feingranularen Rechteverwaltung dar und ist dafür verantwortlich, die in der Datenbank gespeicherten Berechtigungen eines Nutzers in ein CASL Ability-Objekt umzuwandeln.
Dafür wird eine Methode definiert, die das Nutzer-Entity, welches ein Array aus *User Permission* Werten enthält, übergeben bekommt. Das *User Permission* Enum repräsentiert dabei eine einzelne Berechtigung. Diese Berechtigungen werden in CASL-Regeln übersetzt. Die CASL-Regeln bestehen dabei aus einer Kombination von *Actions* und *Subjects*, wobei *Actions* mögliche Operationen beschreiben und *Subjects* die Ressourcentypen darstellen, auf denen diese Operationen ausgeführt werden können. Für die Umwandlung von *User Permissions* in CASL-Regeln wird eine Datenstruktur zur Zuordnung definiert.
Der Rückgabewert der Methode ist ein Ability-Objekt, mithilfe dessen die Berechtigungen eines Nutzers einfach zu überprüfen sind.

Auth Guards und Dekoratoren

Authentifizierung und Autorisierung stellen Querschnittsthemen dar und müssen vor der Ausführung von jeglicher Geschäftslogik überprüft werden.
Normalerweise sollten diese Überprüfungen bereits vor der Ausführung des Request Handlers durchgeführt werden, sodass die zugehörigen Methoden bei nicht angemeldetem Nutzer oder nicht ausreichenden Rechten gar nicht erst aufgerufen werden.
Dies wird über Dekoratoren und Guards definiert. Dekoratoren dienen dabei ausschließlich der Definition von Metadaten und enthalten keine eigene Logik.
Die tatsächliche Prüflogik wird in den Guards implementiert, die vor der Ausführung des Request Handlers aufgerufen werden und über Annotationen an die jeweiligen Endpunkte gebunden sind.

Authentifizierung:
Für die Authentifizierung werden zwei verschiedene Guards definiert. Einer für die Authentifizierung via Bearer Token und einer für API-Key basierte Authentifizierung. Beide Guards nutzen dafür die von PassportJS bereitgestellten Auth Guard Implementierungen.
Da sowohl der JWT als auch der API-Key Teil des Requests sind, ist kein zusätzlicher Dekorator notwendig.
Für Anwendungsfälle, die Informationen über den aktuell authentifizierten Nutzer benötigen, wurde ein *Current User* Dekorator implementiert, welcher den Nutzer aus dem Request extrahiert und dem entsprechend annotierten Endpunkt zur Verfügung stellt.
Autorisierung:
Für die Rechteprüfung wird ein *Require Permission* Dekorator definiert, welcher die zu überprüfende *Action* und das *Subject* als Parameter übergeben bekommt und als Metadaten zur Verfügung stellt.
Die eigentliche Prüfung findet dann im *Permission Guard* statt, der das *CanActivate* Interface implementiert. Innerhalb der *canActivate*-Methode werden zunächst die vom Dekorator bereitgestellten Metadaten über einen Reflektor ausgelesen. Anschließend wird der Nutzer aus dem Request extrahiert und mithilfe der *Ability Factory* das Ability-Objekt des Nutzers erstellt. Auf Basis dessen überprüft die Anwendung, ob der Nutzer die notwendigen Rechte hat, um die Aktion auf dem gewünschten Ressourcentyp auszuführen. Falls ja, wird *true* zurückgegeben, andernfalls *false*, wodurch der Request mit dem HTTP-Status-Code 403 abgebrochen wird.

Die Registrierung der Guards erfolgt dabei über den von NestJS bereitgestellten *Use Guards* Dekorator am jeweiligen Endpunkt. Beim Permission Guard ist dabei zusätzlich die Annotation des *Require Permission* Dekorators erforderlich, um die zu prüfenden Berechtigungen zu definieren.
Durch die Zentralisierung von Authentifizierungs- und Autorisierungslogik wird eine klare Trennung zwischen Sicherheitslogik und Geschäftslogik erreicht, was sowohl die Konsistenz der Implementierung im Sinne der konzeptionellen Integrität, als auch die damit verbundene Wartbarkeit der Anwendung deutlich verbessert.

6.1.2 Umsetzung der Logging-Strategie

Bei der Umsetzung des, in 4.6.1 Logging beschriebenen, Logging-Konzepts wurde, wo möglich, eine zentrale Implementierung angestrebt. Dadurch sollen konsistente Log-Nachrichten sichergestellt, sowie die Wartbarkeit und der Entwicklungsaufwand möglichst gering gehalten werden.
Aufgrund dessen wurden mehrere Komponenten entwickelt, die zentrale Aspekte der Logging-Strategie, wie Logging von Datenbankoperationen, eingehenden und ausgehenden API-Requests, sowie von aufgetretenen Fehlern, übernehmen.

Alle weiteren Aspekte der beschriebenen Logging-Strategie müssen jeweils an den Stellen implementiert werden, an denen die Ereignisse auftreten, da unter anderem weiterer Kontext für die Log-Nachricht benötigt wird.

Logging von Datenbankoperationen

Eine implementierte Logger-Klasse, die das *TypeOrmLoggerInterface* implementiert, wodurch die von TypeORM bereitgestellten Logging Hooks genutzt werden, übernimmt das Logging von Datenbankoperationen. Der implementierte Logger wird dadurch automatisch bei allen relevanten Ereignissen aufgerufen und erstellt daraufhin entsprechende Log-Einträge mithilfe des NestJS-Loggers. Ausgeführte Datenbankabfragen, sowie Log-Events der Kategorie *INFO* gibt der Logger dabei als Debug-Logs aus. Andere Log-Typen werden auf die entsprechende Logging-Methode des NestJS-Loggers abgebildet.
Tritt während der Ausführung einer Datenbankabfrage ein Fehler auf, gibt die Anwendung zusätzlich zur Fehlermeldung auch die zugehörige Query als Error-Log aus, um zusätzlichen Kontext für die Analyse bereitzustellen.
Die Registrierung der implementierten Logger-Klasse findet im *DB Config Service* statt, sodass TypeORM diese automatisch nutzt.

Logging von eingehenden API-Requests

Das Logging von eingehenden API-Requests übernimmt ein zentraler NestJS-Interceptor. Dieser Interceptor wird automatisch für alle eingehenden API-Requests aufgerufen und erzeugt dabei Log-Einträge mit allen wesentlichen Informationen.
Dabei loggt er die aufgerufene URL, die ID des aufrufenden Nutzers, die übergebenen Request-Daten, sowie den HTTP-Response-Code und die Dauer der Bearbeitung des Requests.
Bevor der Request Body zum Log-Eintrag hinzugefügt wird, werden sensible Informationen mithilfe einer im Folgenden beschriebenen Methode anonymisiert, sodass keine sicherheitsrelevanten Daten, wie Passwörter oder API-Keys, in den Log-Dateien enthalten sind.

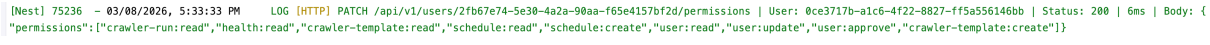

Abbildung 28: Logging von eingehenden API-Requests

Logging von ausgehenden API-Requests

Neben eingehenden API-Requests protokolliert die Anwendung auch ausgehende Anfragen. Hierfür wurde ein weiterer Interceptor implementiert.
Durch die Verwendung von unterschiedlichen Axios-Instanzen in verschiedenen Komponenten, muss der Interceptor bei Initialisierung der entsprechenden Komponente registriert und die genutzte Axios-Instanz übergeben werden.
Der Interceptor fängt anschließend alle ausgehenden Requests, sowie deren zugehörige Antworten ab und erzeugt entsprechende Log-Einträge. Der Log-Eintrag enthält dabei für

jede Anfrage unter anderem die Ziel-URL, sowie die Request-Daten. Auch hierbei werden sensible Daten mithilfe der im Folgenden beschriebenen Methode entfernt.
Für alle Antworten wird der HTTP-Response-Code, sowie die Dauer des Requests und im Fehlerfall die Fehlernachricht geloggt.

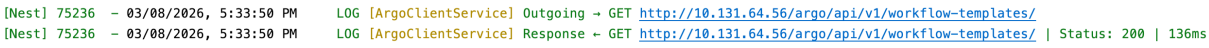

Abbildung 29: Logging von ausgehenden API-Requests

Umgang mit sensiblen Daten

Wie bereits beschrieben, wurde eine Methode implementiert, die sicherheitsrelevante Informationen aus Objekten entfernt.
Die Überprüfung der Objekte erfolgt dabei rekursiv anhand einer Liste von sensiblen Parameternamen, welche anonymisiert werden sollen. Wenn ein entsprechender Parameter erkannt wird, wird dieser durch einen Platzhalter ersetzt, sodass keinerlei sensible Informationen in den Logs enthalten sind.

Logging von Fehlern

Ein globaler Error Interceptor behandelt alle nicht abgefangenen Fehler bei der Abarbeitung von Abfragen.
Dabei wertet er zunächst den Statuscode aus und gibt bei clientseitigen Fehlern (4xx Bereich) nur Logs der Kategorie *warn* aus. Diese lassen sich oftmals auf fehlerhafte Anfragen des Nutzers zurückführen, wie beispielsweise die Abfrage eines bereits gelöschten Runs.
Alle anderen Fehler werden als schwerwiegende Fehler mit dem Log Level *error* geloggt. Hierbei enthält der Log-Eintrag zur Unterstützung der Analyse zusätzlich den Stacktrace, sowie die Kennung des anfragenden Nutzers.

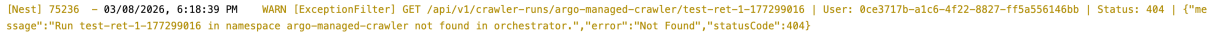

Abbildung 30: Logging von Fehlern

6.1.3 Targets

Um die Erreichbarkeit der Target-URLs der Crawler überprüfen zu können, müssen diese zunächst gespeichert werden. Targets repräsentieren dabei die, im Rahmen von Schedules, definierten Parameter, die sich auf die Ziel-URL von Crawlern beziehen. Deren Verfügbarkeit soll regelmäßig überprüft werden.
Die Persistierung der Targets erfolgt dabei in einer eigenen Datenbanktabelle. Zwischen Targets und Schedules besteht zudem eine relationale Verknüpfung, welche über eine eigene Datenbanktabelle abgebildet wird. Diese Tabelle ist notwendig, da eine Schedule mehrere Targets haben und ein Target aber auch in mehreren Schedules vorkommen kann.
Im Zuge der Erstellung einer Schedule überprüft der *Schedules Service*, anhand ihres Namens, alle Parameter, ob diese ein Target definieren. All diese Parameter werden dann an den *Targets Service* übergeben, welcher überprüft, ob die URL gültig ist, und ob diese bereits in der Datenbank existiert. Existiert die URL bereits, gibt die Methode das bestehende

Target-Objekt aus der Datenbank zurück. Andernfalls erfolgt die Anlage eines neuen Target-Objekts in der Datenbank, welches anschließend zurückgegeben wird.
Die gesammelten Targets der Schedule werden im *Schedules Service* anschließend bei Speicherung der Schedule in der Datenbank referenziert.
Auch bei Aktualisierung einer Schedule müssen alle Parameter erneut auf Target-URLs überprüft werden. Dabei vergleicht der *Schedules Service* die neu erkannten Targets mit den bisher der Schedule zugeordneten Targets. Treten dabei Änderungen auf, werden neue Targets über den *Targets Service* angelegt und entfernte Targets dem *Targets Service* zur Löschung übergeben. Eine Löschung erfolgt dort jedoch nur, falls diese keiner weiteren Schedule mehr zugeordnet sind. Dasselbe passiert auch bei der Löschung von Schedules.

6.1.4 Health

Für die Überprüfung des Gesundheitsstatuses muss zwischen dem Systemstatus und dem Status einzelner Target-URLs unterschieden werden.

Health Status des Systems

Der Systemstatus setzt sich aus dem Status des Orchestrators und dem des Kubernetes Clusters zusammen.
Zur Bestimmung des Orchestrator-Status fragt der *Health Service* über den *Orchestrator Client* einen Endpunkt des Orchestrator Servers ab. Tritt bei der Anfrage ein Fehler der HTTP-Kategorie 4xx auf, so wird der Systemstatus als **LIMITED** zurückgeliefert. Denn diese Fehlerkategorie beschreibt Client-Fehler, weswegen der Fehler wahrscheinlich im Crawler-Management liegt, jedoch vielleicht trotzdem einige Funktionen ausführbar sind. Tritt jedoch ein Fehler der Kategorie 5xx oder ein anderer unerwarteter Fehler auf, gibt die Methode den Status **DOWN** zurück. Bei erfolgreicher Anfrage wird der Status auf **AVAILABLE** gesetzt.
Der Status des Kubernetes Clusters wird mithilfe des *Kubernetes Clients* ermittelt. Zunächst fragt er hierfür alle Nodes des Clusters ab. Sollte kein Node verfügbar sein, gibt er den Status **DOWN** zurück.
Die Methode des *Kubernetes Clients* überprüft für alle gefundenen Nodes einzeln den Zustand. Befindet sich mindestens ein Node nicht im Status Ready oder tritt Memory, Disk oder PID Pressure auf, wird der Gesamtzustand des Clusters als **LIMITED** gewertet, da Ressourcenengpässe die Stabilität des Systems beeinträchtigen.
Anschließend erfolgt eine Überprüfung aller Pods des Crawler-Managements. Ist dabei ein Pod beendet oder nicht im Status Ready, gibt die Methode ebenfalls den Status **LIMITED** zurück, da, selbst wenn nicht alle Pods laufen, Teile des Systems oder auch alle, nur unter geringerer Last, weiterhin funktionieren.
Tritt jedoch bei einer der Anfragen an das Cluster ein Fehler auf, gibt die Methode den Status **DOWN** zurück.
Sind alle Überprüfungen erfolgreich, gibt die Methode den Status **AVAILABLE** zurück.

Health Status der Target-URLs

Der *Health Service* überprüft den Status der Target-URLs einzeln pro URL anhand ihrer Erreichbarkeit.

Hierfür liefert das *Targets-Modul* alle Targets, die mindestens eine nicht pausierte Schedule referenzieren, sodass ausschließlich aktuell relevante Targets überprüft werden.

Anschließend überprüft eine Hilfsmethode für jede URL, ob diese erreichbar ist. Hierbei wird versucht, eine Anfrage an die jeweilige URL zu senden. Tritt dabei ein Fehler der HTTP-Kategorie 5xx auf, gibt die Methode den Status **DOWN** für diese URL zurück. Tritt ein Fehler der Kategorie 4xx auf, wird der Status als **LIMITED** gewertet, da dies beispielsweise auch an fehlenden Zugriffsrechten liegen kann.

Bei erfolgreicher Anfrage gibt sie den Status **AVAILABLE** zurück.

Tritt ein anderer Fehler auf, wie beispielsweise eine Zeitüberschreitung der Anfrage, wird die Anfrage zeitversetzt wiederholt. Sollte auch dieser Versuch fehlschlagen, gibt die Methode den Status **DOWN** zurück.

6.1.5 Orchestrator-Synchronisation

Wie bereits im Kapitel Architekturentscheidungen beschrieben, wird ein regelmäßiger Synchronisationsvorgang zwischen Argo Workflows und dem Crawler-Management in Bezug auf die Schedules benötigt. Zum einen muss sichergestellt sein, dass die im Orchestrator vorhandenen Schedules mit den lokal persistierten Schedules übereinstimmen (4.7.4 Synchronisation von Schedules mit dem Orchestrator). Zum anderen muss regelmäßig geprüft werden, ob die, von einer Schedule bereitgestellten, Parameter weiterhin mit den, vom referenzierten Template, erwarteten Parametern übereinstimmen (4.7.3 Eigene Persistierung von Schedules).

Für beide dieser Funktionalitäten ist das *Orchestrator Sync-Modul* zuständig.

Dafür implementiert das Modul das *onModuleInit* und *onModuleDestroy* Interface.

Bei Initialisierung des Moduls werden zwei voneinander unabhängige Intervalle gestartet, eines zur Synchronisation der Schedules zwischen Orchestrator und Crawler-Management und eines zur Überprüfung von Inkonsistenzen zwischen Schedules und Templates. Bei Zerstörung des Moduls werden diese Intervalle beendet, um Ressourcenlecks zu vermeiden.

Schedule-Synchronisation

Für die Synchronisation der Schedules fragt die Methode zunächst alle Schedules vom Orchestrator über den *Orchestrator Client* ab und übergibt diese anschließend an den *Schedules Service*.

Der *Schedules Service* baut daraufhin eine Map-Datenstruktur aus allen lokal gespeicherten Schedules auf, bei welcher der Schlüssel aus Konkatenation von Name und Namespace besteht und die jeweilige Schedule als Wert definiert wird. Diese Vorgehensweise dient der

effizienten Verarbeitung, weil dadurch ein schneller Zugriff auf einzelne Schedules möglich ist, ohne der Notwendigkeit über alle Schedules iterieren zu müssen.
Anschließend überprüft eine Hilfsmethode für jede Schedule, ob diese in der lokalen Datenbank gespeichert ist. Ist dies nicht der Fall, wird sie in der Datenbank persistiert und ein Warn-Log geschrieben, da Schedules ausschließlich über die Crawler-Management Anwendung erstellt werden sollten. Existiert die Schedule bereits, werden die, zur Schedule lokal gespeicherten, Daten mit denen aus dem Orchestrator verglichen und gegebenenfalls aktualisiert. Auch schreibt die Methode im Falle von Änderungen ein Warn-Log, da Änderungen an der Schedule ausschließlich über das Crawler-Management-System durchgeführt werden sollten.
Daraufhin erfolgt eine Überprüfung der Target-Parameter. Neue Targets werden dabei an den *Target Service* zur Speicherung weitergegeben und entfernte Targets zur möglichen Löschung freigegeben.
Zuletzt wird die Schedule aus der lokalen Map entfernt und damit als synchronisiert markiert.
Nachdem alle vom Orchestrator gelieferten Schedules abgearbeitet sind, bleiben in der Map nur noch diejenigen Schedules übrig, die lokal existieren, jedoch im Orchestrator nicht mehr vorhanden sind. Diese werden aus der lokalen Datenbank gelöscht, da der Orchestrator im Falle von Inkonsistenzen als führendes System gilt.

Erkennung von Inkonsistenzen zwischen Schedule und Template

Die zweite periodisch ausgeführte Routine dient zur Erkennung von möglichen Inkonsistenzen zwischen einer Schedule und ihrem referenzierten Crawler Template.
Hierfür fragt der *Orchestrator Sync Service* zunächst alle lokal gespeicherten Schedules vom *Schedules Service ab*. Für jede Schedule überprüft eine Hilfsfunktion diese auf Inkonsistenzen und setzt oder entfernt entsprechende Fehlermeldungen.
Zur Kategorisierung von Fehlern wurden unterschiedliche Fehlertypen definiert. Aktuell existiert ein Fehlertyp für nicht existierende Templates und einer für Parameterinkonsistenzen.
Zunächst fragt die Hilfsmethode das referenzierte Template der Schedule vom Orchestrator über den *Orchestrator Client* ab. Sollte er dieses nicht finden, wird über den *Schedules Service* ein Fehler vom Typ **fehlendes Template** gesetzt.
Existiert das Template und ist ein Fehler vom Typ **fehlendes Template** gesetzt, wird dieser wieder entfernt, da das Template inzwischen existiert.
Anschließend überprüft die Methode, ob die Version des Templates mit der Version in der gespeicherten Schedule übereinstimmt. Falls ja, ist die Überprüfung für diese Schedule nicht notwendig und somit beendet, da sich das Template seit der letzten Prüfung nicht verändert hat. Sollten die Versionen jedoch nicht übereinstimmen, wird die im Orchestrator gespeicherte zugehörige Schedule über den *Orchestrator Client* abgerufen, da nur dort die Parameter gespeichert sind. Auf Basis dessen vergleicht nun eine Hilfsmethode die

Parameter der Schedule mit denen des Templates und setzt bei Inkonsistenzen entsprechende Fehlermeldungen vom Typ **Parameter Inkonsistenz** über den *Schedules Service* für die Schedule. Sollten jedoch keine Inkonsistenzen gefunden werden, werden alle Fehler dieses Typs, falls existent, von der Schedule entfernt.
Abschließend wird noch der Parameter in der Schedule für die letzte überprüfte Template-Version gesetzt.
Bei beiden Routinen wird sichergestellt, dass die vorherige Ausführung bereits beendet ist, bevor eine neue Ausführung startet.

6.1.6 Schedules

Schedule Engine

Um Schedules erstellen und verändern zu können, bedarf es spezifischer Logik, insbesondere bei Verwendung von Argo Workflows als Orchestrator.
Argo Workflows definiert Schedules als sogenannte Cron Workflows. Ein Cron Workflow referenziert ein Workflow beziehungsweise Crawler Template und definiert dabei neben der zeitlichen Ausführung auch beispielsweise Retry-Mechanismen und Parameter. Die Definition eines Cron Workflows erfolgt dabei im YAML-Format.
Zur Kapselung dieser Logik dient die entwickelte Argo Schedule Engine, die das Schedule Engine Interface mit Methoden zur Erstellung und Änderung von Schedules implementiert. Die Methode zur Erstellung einer Schedule erhält ein Objekt mit allen relevanten Daten der Schedule, sowie den erstellenden Nutzer zu Zwecken der Dokumentation. Daraufhin baut sie anhand der übergebenen Parameter mithilfe des Argo Cron Workflow Builders ein Schedule-Objekt und konvertiert dieses anschließend in ein YAML-Dokument. Dieses übergibt sie schließlich dem *Argo Client Service* zur Übertragung an den Argo Server.
Die Methode zur Änderung einer Schedule erwartet den Namen und den Namespace der Schedule, sowie die zu ändernden Parameter. Der bestehende Cron Workflow wird über den *Argo Client Service* geholt, entsprechend den übergebenen Parametern angepasst und erneut in ein YAML-Dokument konvertiert, und abschließend wieder dem *Argo Client Service* zur Weitergabe an den Argo Workflows Server übergeben.

Argo Cron Workflow Builder

Der *Argo Cron Workflow Builder* dient zur Konstruktion des komplexen Cron Workflow Objekts für Argo und abstrahiert dabei die komplexe Logik.
Zunächst setzt er hierbei einige grundlegende Felder des Objekts mit festen Werten, wie beispielsweise die API-Version von Argo oder den Typ des Objekts (Cron Workflow).
Verschiedene Methoden setzen anschließende spezifische Felder, wie Name, Namespace oder den Cron-Ausdruck inklusive Zeitzone.
Wo erforderlich werden hierbei auch entsprechende Validierungen durchgeführt, wie beispielsweise beim Setzen der Retry-Strategie, sodass keine negativen Zeitwerte gesetzt werden können.

Die Methode *build* liefert schließlich das vollständig konstruierte Objekt zurück und überprüft dabei, ob alle erforderlichen Felder gesetzt sind und löst andernfalls entsprechende Exceptions aus.

6.1.7 DB Config Service mit Migrationen

Wie bereits im Architekturkapitel im Abschnitt Architekturentscheidungen beschrieben, wird für Datenbankänderungen TypeORM Migrations verwendet. Eine Migrationsdatei, die sicherstellt, dass die Migrationen mit derselben Konfiguration erzeugt werden, welche die Anwendung auch selbst nutzt, dient dabei als Einstiegspunkt für die Migrationsskripte.
Zuerst erfolgt in der Migrationsdatei eine Validierung der erforderlichen Umgebungsvariablen. Anschließend wird ein Config Service mit den validierten Variablen erstellt. Mithilfe des *DB Config Service* wird ein DataSource-Objekt erstellt und für TypeORM exportiert.
Der eben schon beschriebene *DB Config Service* implementiert die *TypeOrmOptions Factory* und liefert dabei die konkreten *TypeORM DataSource* Optionen. Dabei werden die Zugangsdaten für die Datenbank aus dem, von der Migrationsdatei bereitgestellten, Config Service ausgelesen und zudem weitere Konfigurationsparameter, wie Logging-Optionen oder Migrationsverhalten, gesetzt.
Die Definition der zu berücksichtigenden Entities findet ebenfalls im Zuge dieser Konfiguration statt.
Wird also eine neue Datenbanktabelle benötigt und dafür somit ein Entity erstellt, muss dieses hier hinzugefügt werden. Anschließend kann der Migrationsbefehl ausgeführt werden, der eine entsprechende Migrationsdatei generiert. Nach händischer Überprüfung dieser, muss sie ebenfalls im *DB Config Service* hinzugefügt werden. Dadurch wird sie beim nächsten Start der Anwendung automatisch auf die Datenbank angewandt. Bereits ausgeführte Migrationen sind dabei in einer eigenen Datenbanktabelle dokumentiert.

6.1.8 Validierung von Umgebungsvariablen

Für einen stabilen Betrieb der Anwendung ist es wichtig, dass die benötigten Umgebungsvariablen vorhanden und korrekt sind.
Hierzu wird bei Start der Anwendung eine Validierungskomponente ausgeführt, die alle erforderlichen Umgebungsvariablen validiert, gegebenenfalls eine entsprechende Fehlermeldung erzeugt und in diesem Fall den Start der Anwendung daraufhin abbricht.
Dadurch lässt sich verhindern, dass Konfigurationsfehler in Bezug auf die Umgebungsvariablen erst zur Laufzeit an einzelnen Stellen auftreten. Denn dies würde die Nachvollziehbarkeit deutlich erschweren und zudem die Stabilität des Systems massiv beeinträchtigen. Stattdessen startet die Anwendung gar nicht erst, wodurch eine Fehlkonfiguration sofort erkennbar ist.

6.1.9 Argo Client

Argo Auth

Für die Authentifizierung des Crawler-Management-Backends gegenüber dem Argo Workflows Server ist eine spezielle Logik notwendig, da sich der Authentifizierungsmechanismus unterscheidet, je nachdem, ob das Backend innerhalb des Clusters oder zu Test- und Entwicklungszwecken lokal betrieben wird.
Nach dem Single Responsibility Principle (Martin, 2003, S. 95) sollte eine Klasse nur eine Verantwortlichkeit haben, weswegen die Authentifizierungslogik für Argo in eine eigene Klasse ausgelagert wurde.
Eine hierfür definierte Hilfsmethode bestimmt zunächst anhand der Umgebungsvariable *Stage*, ob die Anwendung im Cluster läuft.
Sollte die Anwendung im Cluster laufen, wird aus dem Dateisystem des Containers das injizierte Service Account Token ausgelesen. Der Anwendung wurde zuvor per Helm eine Rolle zugewiesen, welche die entsprechenden Rechte besitzt, um die notwendigen Daten vom Argo Workflows Server abzurufen.
Sollte die Anwendung lokal laufen, muss das Token über eine Umgebungsvariable bereitgestellt werden.
Das ermittelte Token wird anschließend bei jeder Anfrage an den Argo Workflows Server als Bearer Token im *Authorization Header* gesetzt, sodass sich der *Argo Client Service* vollständig auf die Geschäftslogik konzentrieren kann.

Argo Query Param Builder

API-Parameter folgen bei der Abfrage von Daten vom Argo Server besonderen Strukturen, weswegen diese Logik in eine extra Klasse nach dem Builder Pattern ausgelagert wurde.
Zunächst erfolgt hierfür eine Differenzierung zwischen normalen Parametern und Label-Parametern.
Die Definition normaler Parameter mit einem einzelnen Wert erfolgt in üblicher Query-Parameter-Form. Besitzt ein Parameter mehrere Werte, werden diese per Komma separiert und nicht, wie eigentlich üblich, durch mehrfaches Setzen desselben Parameters.
Label-Parameter werden mit dem Präfix listOptions.labelSelector definiert.
Ein einzelner Wert wird in der Form *parameter=wert* übergeben. Die Definition von mehreren Werten erfolgt per IN-Syntax.
Die folgende Tabelle verdeutlicht die Definition von Query-Parametern bei Anfragen an den Argo Workflows Server.

Ausgangssituation	Allgemeine Definition	Beispiel
Parameter mit einzelnem Wert	?{parameter}={wert}	?namePattern=test
Parameter mit mehreren Werten	?{parameter}= {wert1}, {wert2}	?fields=name,phase
Label-Parameter mit einzelnem Wert	?listOptions.labelSelector= {parameter}={wert}	?listOptions.labelSelector= cron-workflow=test
Label-Parameter mit mehreren Werten	?listOptions.labelSelector= {parameter} in ({wert1}, {wert2})	?listOptions.labelSelector= phase in (success, error)

Tabelle 3: Argo Workflows Query-Parameter Syntax

Um diese Funktionalität durch eine eigene Klasse zu kapseln, wird das Builder Pattern genutzt, welches dazu dient, Objekte inkrementell zu konstruieren und dabei die Details, wie das Objekt erstellt, repräsentiert und zusammengebaut wird, versteckt (Gamma, Helm, Johnson, & Vlissides, 1993, S. 13).
Ziel der Builder Klasse ist es hierbei einen Argo-kompatiblen Query-String zu bauen.
Intern verwaltet der Builder hierfür eine Map-Datenstruktur, bei der ein String als Schlüssel und ein Array aus Strings als Wert dient.
Zudem werden drei Methoden implementiert:
Die Methode *add* dient zum Hinzufügen normaler Parameter und nimmt hierfür den Parameternamen entgegen, sowie einen oder mehrere Werte. Dabei konvertiert sie die Werte zunächst zu Strings und fügt sie anschließend der Map hinzu. Existiert bereits ein Wert zum angegebenen Key bzw. Parameternamen, werden die Werte an diesen angehängt.
Eine weitere Methode ist die Methode *addLabelSelector*, welche die Funktionalität zum Hinzufügen von Label-Parametern bietet. Hierfür nimmt sie den Parameternamen, sowie beliebig viele Werte dafür entgegen. Anschließend unterscheidet sie, ob mehrere Werte übergeben wurden oder nur einer, und wendet entsprechend die oben beschriebene Syntax an. Zuletzt wird der zusammengesetzte Parameter-String aus Parameternamen und -werten in die Map unter dem Key *listOptions.labelSelector* gespeichert bzw. angehängt.
Die Methode *build* erzeugt schließlich den vollständigen Query String.
Durch diese Abstraktion wird verhindert, dass sich der *Argo Client Service* mit den syntaktischen Besonderheiten der Argo-API befassen muss, wodurch eine klare Trennung zwischen API-Logik und Geschäftslogik erreicht wird.

6.1.10 Authentifizierung gegenüber Kubernetes

Für die Authentifizierung des Crawler-Management-Backends gegenüber dem Kubernetes-API-Server ist eine spezielle Logik notwendig, da sich der Authentifizierungsmechanismus unterscheidet, je nachdem, ob das Backend innerhalb des Clusters oder zu Test- und Entwicklungszwecken lokal betrieben wird.
Die Authentifizierungslogik wurde hierbei, ähnlich wie bei der Authentifizierung gegenüber Argo, in eine eigene Klasse ausgelagert.
Zunächst überprüft eine Hilfsmethode anhand der Umgebungsvariable *Stage*, ob die Anwendung im Cluster läuft.
Sollte die Anwendung im Cluster laufen, wird die KubeConfig-Datei aus dem Cluster geladen, was eine Client-Bibliothek von Kubernetes übernimmt.
Bei lokaler Ausführung wird die KubeConfig-Datei vom, per Umgebungsvariable angegeben, Pfad gelesen. Sollte diese Variable nicht gesetzt sein, wird versucht sie von einem Default-Pfad zu lesen.
Anschließend überprüft die Methode durch Abfrage des zugehörigen Servers, ob die KubeConfig-Datei gültig ist.
Abschließend erstellt die Kubernetes Client-Bibliothek anhand der KubeConfig einen API-Client. Den API-Client kann der *Kubernetes Client Service* schließlich für Anfragen nutzen.

6.1.11 Crawler-Metriken

Der *Crawler Run Metrics Service* bildet die zentrale Komponente zur Verwaltung und Persistierung von Crawler-Metriken und -Events. Wie im Architekturkapitel beschrieben, werden diese Daten von den Crawlern an das Crawler-Management-Backend gesendet und dann an den *Crawler Run Metrics Service* übergeben.
Zu Persistierung der Daten wurden zwei Datenbanktabellen erstellt.
Eine Tabelle zur Speicherung der Metriken und eine zur Speicherung der Events.
Beide Tabellen verwenden dabei einen zusammengesetzten Schlüssel aus dem Namen und dem Namespace des Runs. Zudem wird, sofern vom Crawler mitgeliefert, eine Referenz zur Schedule des Runs gespeichert.
Für die Speicherung der Crawler-Metriken und -Events stellt der Service mehrere Methoden bereit. Dabei unterscheidet er zwischen Metriken aus dem Sammeln von Datensätzen und dem Download dieser, da sie in der Regel von unterschiedlichen Modulen des Crawlers gesendet werden (4.2 Kontextabgrenzung).
Die Persistierung der Metriken erfolgt idempotent. Das bedeutet, dass neu übermittelte Werte bereits gespeicherte Werte überschreiben, anstatt diese inkrementell zu erhöhen. Dies stellt sicher, dass Crawler, beispielsweise bei Netzwerkproblemen, Metriken erneut senden können, ohne diese zu verfälschen.
Auch zur Abfrage der gespeicherten Metriken und Events werden mehrere Methoden angeboten.

Erstens können die Metriken eines konkreten Runs direkt aus der Datenbank geladen werden. Zweitens wird eine Methode angeboten, mit der aggregierte Metriken aller Runs einer Schedule abgefragt werden können. Drittens existiert eine Methode, mit welcher die Metriken der Runs einer Schedule pro Tag aggregiert über einen angegebenen Zeitraum abgefragt werden können. Die Aggregationslogik wird bei beiden Methoden über eine optimierte Query direkt auf Datenbankebene effizient ausgeführt, ohne alle Runs zunächst in den Arbeitsspeicher des Containers laden zu müssen.
Zuletzt wird noch eine Methode zur Abfrage aller Events eines Runs angeboten.

6.2 Management-Frontend

6.2.1 Auth Context

Der *Auth Context* bildet die zentrale Schicht für Authentifizierung und Autorisierung im Frontend und umschließt dabei die gesamte Anwendung.
Der *Auth* Context kapselt sämtliche Logik für das Laden des aktuellen Benutzerprofils, das Setzen von Berechtigungen, sowie Prozesse für Login, Logout und zur Registrierung.
Ein globaler Zustand in Bezug auf Nutzerinformationen ist essenziell, um Konsistenz zwischen allen Komponenten zu garantieren und somit widersprüchliche Zustände zu verhindern.
Der Auth Context stellt dabei Methoden zum Login, Logout und zur Registrierung zur Verfügung und ruft dafür die entsprechenden Endpunkte im Backend auf, woraufhin der globale Nutzerzustand gesetzt wird.
Zur Bereitstellung dieses Zustands wird eine Kontextvariable definiert, welche beispielsweise den aktuellen Nutzer, den Ladezustand, sowie ein Ability-Objekt zur Verfügung stellt.
Die Informationen zum aktuellen Nutzer werden durch Aufruf des *auth/me* Endpunkts im Backend abgefragt. Nach erfolgreichem Login setzt das Backend ein HTTP Only Cookie, was im Browser des Nutzers gespeichert und automatisch bei allen Folgeanfragen mitgesendet wird (4.6.2 Authentifizierung und Autorisierung).
Das bereits erwähnte Ability-Objekt dient, wie auch bereits für das Backend beschrieben, zur Rechteprüfung. Es wird aus Autorisierungsregeln erzeugt, die im Backend definiert und als Teil der Nutzerdaten an das Frontend übertragen werden. Mithilfe dieses Objektes und der von CASL bereitgestellten Funktionen kann überprüft werden, ob ein Nutzer berechtigt ist, eine bestimmte Aktion auf einem bestimmten Ressourcentyp auszuführen. Dabei ersetzt die clientseitige Rechteprüfung jedoch in keinem Falle die serverseitige Rechteprüfung.

6.2.2 Guards

Auth Guard

Ein Großteil der Seiten der Anwendung soll ausschließlich für angemeldete Nutzer erreichbar sein. Würde jedoch jede Seite eigens prüfen, ob ein Nutzer angemeldet ist, führe dies zu Redundanz und damit verbundenem erhöhten Wartungsaufwand, zu potenziellen Inkonsistenzen, sowie der Verletzung der konzeptionellen Integrität.
Daher wurde ein zentraler *Auth Guard* implementiert, der als übergeordnetes Objekt dient und dabei die anzuzeigenden Seitenkomponenten umschließt.
Der *Auth Guard* ruft den globalen Zustand aus dem *Auth Context* ab und prüft, ob ein Nutzer eingeloggt ist und ob sich die Abfrage der Nutzerdaten noch im Ladezustand befindet. Solange die Daten des Nutzers noch laden, wird eine entsprechende Meldung angezeigt. Ist der Nutzer jedoch nicht angemeldet und der Ladevorgang bereits beendet, erfolgt eine Weiterleitung auf die Login-Seite und der Nutzer erhält somit keinen Zugriff auf diese Seite. Sollte ein Nutzer angemeldet sein, werden die übergebenen Kind-Objekte gerendert und die Seite entsprechend angezeigt.
Diese zentrale Lösung verbessert die Wartbarkeit erheblich, da Änderungen an der Authentifizierung nur an einer Stelle vorgenommen werden müssen. Zudem bietet diese Umsetzung dem Nutzer ein konsistentes Verhalten über alle Seiten der Anwendung hinweg.

Permission Guard

Ein ähnliches Vorgehen setzt der *Permission Guard* um, denn wie bereits beschrieben wird neben der Authentifizierung auch eine feingranulare Rechteverwaltung bereitgestellt. Nicht jeder Nutzer soll somit auf alle Seiten der Anwendung zugreifen können.
Um die Rechteprüfung ebenfalls zentral bereitzustellen, wurde der *Permission Guard* implementiert. Analog zum *Auth Guard* umschließt auch dieser die darzustellenden Komponenten, muss dabei jedoch stets selbst vom *Auth Guard* umschlossen werden, da ein angemeldeter Nutzer vorausgesetzt wird.
Der *Permission Guard* überprüft anhand des übergebenen Ability-Objekts und der übergebenen, geforderten Rechte, ob der Nutzer diese besitzt. Die Überprüfung findet mithilfe der *can* Methode von CASL statt.
Sollte ein Nutzer nicht alle nötigen Rechte besitzen, verweigert ihm der *Permission Guard* den Zugriff und zeigt an, welche Rechte ihm fehlen.
Besitzt der Nutzer jedoch die notwendigen Rechte, werden die eingeschlossenen Komponenten gerendert und der Nutzer bemerkt den Permission Guard nicht.

Can

Jedoch sollen nicht nur gesamte Seiten vor unautorisiertem Zugriff geschützt werden. So könnte ein Nutzer zwar die Rechte besitzen, Schedules zu sehen, jedoch nicht sie zu verändern. Hierfür ist der *Permission Guard* ungeeignet, da dieser sich auf gesamte Seiten bezieht. Für die gewünschte Funktionalität bietet CASL die *Can*-Komponente an. Diese

umschließt eine oder mehrere UI-Komponenten und erwartet das Ability-Objekt aus dem *Auth Context*, sowie die erforderlichen Rechte. Die Komponente wird nur dann gerendert, wenn der Nutzer die angegebenen Rechte besitzt.

Zusammenfassend lässt sich feststellen, dass durch Kombination aus *Auth Context*, *Auth Guard*, *Permission Guard* und die von CASL bereitgestellten Komponenten eine konsistente und zentralisierte Authentifizierungs- und Autorisierungslogik geschaffen wurde, wodurch sich Kernkomponenten auf die eigene Geschäftslogik konzentrieren können.

6.2.3 Utils

Im Utils-Verzeichnis des Frontends befinden sich diverse Hilfsfunktionalitäten. Einige davon werden im Folgenden exemplarisch beschrieben.

Cron Hilfsfunktionen

Da die Definition der Zeitpläne der Schedules im Cron-Format erfolgt, welches nur mit technischem Hintergrundwissen lesbar ist, werden zwei Hilfsfunktionen definiert.

Zum einen existiert eine Funktion, die aus einem Cron-Ausdruck ein menschlich lesbares Format erzeugt. Zur Validierung und Umwandlung werden hierfür zwei Bibliotheken verwendet.

Zum anderen wurde eine Funktion implementiert, die aus einem gegebenen Cron-Ausdruck und dessen Zeitzone die nächste Ausführung in Bezug auf die aktuelle Zeit und die Zeitzone des Nutzers berechnet.

Eine weitere Funktion berechnet zu einem gegebenen Cron-Ausdruck und zugehöriger Zeitzone mithilfe von zwei Bibliotheken einen Hinweistext, der die, durch den Cron-Ausdruck und dessen Zeitzone definierten, Zeiten in die lokale Zeit des Nutzers umrechnet. Beide Funktionen dienen somit dazu, dem Nutzer eine einfach lesbare Information anhand des Cron-Ausdrucks zu bieten und diese Funktionalität zentral an einer Stelle bereitzustellen, sodass redundante Implementierungen vermieden werden.

Zeitintervall Hilfsfunktion

Diverse zeitliche Informationen werden in Sekunden geliefert, was jedoch bei längeren Zeitintervallen zu schwer interpretierbaren Werten führt.

Eine zentrale Hilfsfunktion löst dieses Problem, indem sie eine gegebene Anzahl an Sekunden in eine benutzerfreundliche Darstellung konvertiert. Abhängig von der Größenordnung werden die Werte in Sekunden, Minuten, Stunden oder Tagen dargestellt.

7 Evaluation

In diesem Kapitel wird evaluiert, welche der definierten Anforderungen umgesetzt wurden, und welche Herausforderungen sich im Zusammenhang mit der Wahl von Argo Workflows als Orchestrierungs-Framework ergeben haben.

7.1 Anforderungsevaluation

Wie bereits im Kapitel 3 Anforderungen beschrieben, beziehen sich die vorgestellten Anforderungen auf ein Crawler-Management-System für JValue und somit nur teilweise auf den Umfang dieser Arbeit. Als Anforderungen für diese Arbeit können die Anforderungen der Priorität **A** als Minimal Viable Product angesehen werden, die der Priorität **B** als Stretch Goals und die der Priorität **C** als Stretch Goals mit geringerer Priorität.

7.1.1 Anforderungen der Priorität A

Anforderungen der Priorität A wurden als Anforderungen definiert, die umgesetzt werden müssen, um ein für das JValue Projekt gut nutzbares Crawler-Management-System bereitzustellen.

A1.1: *Die Anwendung muss* ***vollständig automatisiert*** *mittels CI/CD in die aktuell im Projekt genutzte Kubernetes-Umgebung deployt werden können.* ***(wurde erfüllt)***
Für das automatisierte Deployment in die bereitgestellte Kubernetes-Umgebung wurde ein Helm Chart sowie eine GitHub Action entwickelt, die auf einem Self-Hosted Runner ausgeführt wird, wodurch ein automatisierter Deployment-Prozess sichergestellt wurde (4.3.4 Deployment).

A1.2: *Die Anwendung muss* ***nach Abstürzen automatisch*** *ohne Datenverlust und ohne manuellen Eingriff* ***neu starten****, um die Zuverlässigkeit des Systems zu garantieren.* ***(wurde erfüllt)***
Um Ausfallsicherheit zu gewährleisten, wurde ein Disaster Recovery Konzept erstellt (4.6.4 Disaster Recovery) und entsprechend umgesetzt.

A1.3: *Die Anwendung muss* ***unabhängig von Implementierungsdetails und interner Architektur der Crawler*** *sein, sodass Anpassungen an den Crawlern keine Anpassungen im Management-System erfordern.* ***(wurde erfüllt)***
Die Unabhängigkeit der Crawler-Implementierung und Architektur wird über generische Schnittstellen sichergestellt (4.3.6 Unabhängigkeit des Systems vom Crawler-Aufbau).

A1.4: *Die Anwendung muss so entworfen sein, dass ein* ***Wechsel des Orchestrierungs-Frameworks keine vollständige Neuentwicklung****, sondern lediglich die Entwicklung eines, zur Schnittstelle des neuen Orchestrierungs-Frameworks passenden, Clients erfordert, um ein Vendor Lock-in zu verhindern.* ***(wurde erfüllt)***
Die Unabhängigkeit vom Orchestrierungs-Framework wurde durch eine Orchestrator-Abstraktionsschicht umgesetzt, welche sämtliche orchestrator-spezifische Logik kapselt (4.6.3 Orchestrator-Abstraktionsschicht).

A1.5: *Die Anwendung muss* ***ausführliches Logging*** *bereitstellen, um Fehler nachvollziehbar analysieren zu können.* ***(wurde erfüllt)***
Hierfür wurde eine explizite Logging-Strategie entworfen und umgesetzt (4.6.1 Logging).

A2.1: *Die Anwendung muss durch Authentifizierung und Autorisierung gegen unautorisierten Zugriff geschützt sein.* ***(wurde erfüllt)***
Für Authentifizierung und Autorisierung wurde ein Sicherheitskonzept entworfen und implementiert (4.6.2 Authentifizierung und Autorisierung).

A2.2: *Die Anwendung muss ein* ***Nutzermanagement mit mindestens zwei Rollen*** *(Leserechte/Schreibrechte) bieten, um die Rechte von bestimmten Nutzern einschränken zu können.* ***(wurde ersetzt)***
Das Nutzermanagement auf der Basis von Rollen wurde durch feingranulare Rechteverwaltung, wie in Anforderung C2.1 beschrieben, ersetzt.

A3.1: *Die* ***Workflow-Schablonen*** *müssen über* ***Git versioniert*** *werden, um die Nachverfolgbarkeit zu gewährleisten.* ***(wurde erfüllt)***
Für die Versionierung von Workflow-Schablonen über Git wurde ein eigenes Repository erstellt, in welchem die Workflow-Schablonen verwaltet und über eigens entwickelte GitHub Actions deployt werden können.

A3.2: *Die Anwendung muss die Möglichkeit bieten,* ***parametrisierte Runs für Workflow-Schablonen manuell zu starten****, um einmalige Ausführungen zu ermöglichen.* ***(wurde erfüllt)***
Auf der Crawler Templates Übersichtseite können Nutzer für jedes Template manuelle, parametrisierte Runs starten (5.7 Crawler Templates Seite).

A3.3: *Die Anwendung muss eine Funktion bieten,* ***Schedules für Workflow-Schablonen zu definieren****, inklusive unterschiedlicher Parametrisierung je Schedule, um die regelmäßige automatisierte Ausführung von Workflow-Schablonen zu ermöglichen und dadurch die Aktualität der Datensätze zu gewährleisten.* ***(wurde erfüllt)***

Die Erstellung von Schedules inklusive Parametrisierung erfolgt ebenfalls über die Templates Übersichtsseite (5.7 Crawler Templates Seite).

A3.4: *Die Anwendung muss eine Funktion bieten, die es ermöglicht,* ***Schedules pausieren, fortsetzen und löschen*** *zu können.* ***(wurde erfüllt)***
Über die Schedule Detailseite (5.8.2 Schedule Detailseite) können Schedules pausiert, fortgesetzt und gelöscht werden.

A3.5: *Die Anwendung muss* ***konfigurierbare Retry-Mechanismen für Schedules*** *unterstützen, um die Fehler Resilienz zu verbessern.* ***(wurde erfüllt)***
Bei der Erstellung und Bearbeitung von Schedules kann eine umfangreiche Retry-Strategie definiert werden (5.7 Crawler Templates Seite).

A4.1: *Die Anwendung muss* ***alle ausgeführten Runs anzeigen****, um dem Nutzer die Möglichkeit zu bieten, die Ausführung der Runs nachzuverfolgen und Informationen darüber zu erhalten.* ***(wurde erfüllt)***
Auf der Crawler Runs Übersichtsseite werden alle ausgeführten Runs mit den wichtigsten Informationen dazu aufgeführt (5.6.1 Crawler Runs Übersichtsseite).

A4.2: *Die Anwendung muss eine* ***Filterung der Runs*** *nach folgenden Parametern anbieten, sodass die Suche nach bestimmten Runs erleichtert wird:* ***(wurde erfüllt)***

- *Workflow-Schablone*
- *Start- und Enddatum*
- *Schedule*
- *Aktueller Status*

Die Filterung der Runs nach den genannten Parametern wird auf der Crawler Runs Übersichtsseite ermöglicht (5.6.1 Crawler Runs Übersichtsseite).

A5.1: *Die Anwendung muss* ***Statistiken pro Run*** *anzeigen, sodass Runs entsprechend dieser analysiert werden können:* ***(wurde erfüllt)***

- Anzahl der gesammelten Datensätze
- Anzahl der Cache Hits
- Dauer des Runs

Die geforderten Statistiken werden auf der Crawler Run Detailseite gemeinsam mit weiteren Metriken angezeigt (5.6.2 Crawler Run Detailseite).

A5.2: *Die Anwendung muss* ***aggregierte Statistiken pro Schedule*** *anzeigen, um die Effektivität und Effizienz der Schedule zu analysieren und anhand dessen notwendige Änderungen zu erkennen:* ***(wurde erfüllt)***

- *Gesamtanzahl gesammelter Datensätze*
- *Anzahl gesammelter Datensätze im Zeitverlauf*
- *Anzahl neu gesammelter Datensätze im Zeitverlauf*
- *Verhältnis neu gesammelter Datensätze zu insgesamt gesammelter Datensätze im Zeitverlauf*

Aggregierte Statistiken, sowie zeitliche Verläufe pro Schedule werden auf der Schedule Detailseite visualisiert (5.8.2 Schedule Detailseite).

A6.1: *Die* ***Steuerung der Crawler muss vollständig*** *über die* ***grafische Nutzeroberfläche*** *möglich und somit für nicht-technische Nutzer einfach nutzbar sein, da die Steuerung der Crawler nicht zwangsläufig im Aufgabenbereich der Entwickler liegt.* ***(wurde erfüllt)***
Sämtliche Funktionalitäten werden über die Benutzeroberfläche angeboten. Lediglich die Generierung von API-Keys, welche ohnehin nur von Entwicklern durchgeführt werden sollte, wird ausschließlich über die API angeboten (5 Design).

7.1.2 Anforderungen der Priorität B

Anforderungen der Priorität B wurden als Anforderungen definiert, welche umgesetzt werden sollten, jedoch für die Grundfunktionalität nicht notwendig sind und somit in Bezug auf die Arbeit Stretch Goals darstellen.

B1.1: *Die Anwendung soll einen* ***einfachen Health Status*** *mit Informationen zum Cluster und Orchestrator anzeigen, um unmittelbar einen Überblick über mögliche Probleme zu erhalten.* ***(wurde erfüllt)***
Die Anwendung bietet eine Health Status Seite mit Informationen zum Gesundheitsstatus von Cluster und Orchestrator an (5.5 Health Status Seite und 6.1.4 Health)

B3.1: *Die Anwendung soll es ermöglichen,* ***Zeitpläne, Retry-Mechanismen und Parameter bestehender Schedules ändern zu können****, um beispielsweise notwendige Änderungen aufgrund von Anpassungen am Crawler einfach umsetzen zu können.* ***(wurde erfüllt)***
Über die Schedule Übersichtsseite können der Cron-Ausdruck, die Retry-Strategie, sowie die Parameter einer bestehenden Schedule geändert werden (5.8.2 Schedule Detailseite).

B3.2: *Die Anwendung soll die Möglichkeit bieten,* ***Schedules für die Ausführung zu priorisieren****, um die Ausführung von wichtigeren Schedules sicherzustellen.* ***(wurde nicht erfüllt, aber architektonisch konzipiert)***
Es gibt in der aktuellen Version des Crawler-Management-Systems keine Möglichkeit zur Priorisierung von Schedules. Die Umsetzung dieser Anforderung wurde jedoch bei der

Planung der Architektur mit einbezogen, sodass eine spätere Umsetzung ohne grundlegende Änderungen möglich ist.

B3.3: *Die Anwendung soll* ***Breaking Changes in Workflow-Schablonen erkennen und eine Warnung dafür anzeigen****, sodass Nutzer darauf hingewiesen werden, dass die Ausführungen der Schedule ohne Anpassung wahrscheinlich fehlschlagen werden.* ***(wurde erfüllt)***
Alle Schedules werden regelmäßig gegen ihre referenzierten Workflow-Schablonen validiert, wobei gegebenenfalls entsprechende Fehlermeldungen generiert werden (5.8.2 Schedule Detailseite und 6.1.5 Orchestrator-Synchronisation).

B3.4: *Die Anwendung soll für jede* ***Schedule den nächsten geplanten Run anzeigen****, um jederzeit zu wissen, wann die Datensätze wieder aktualisiert werden.* ***(wurde erfüllt)***
Sowohl auf der Schedule Übersichtsseite als auch auf der Schedule Detailseite wird der nächste geplante Run angezeigt (5.8.1 Schedule Übersichtsseite und 5.8.2 Schedule Detailseite.).

B4.1: *Die Anwendung soll den* ***Fortschritt laufender Runs*** *anzeigen, um diese überwachen und dadurch mögliche Deadlocks erkennen zu können.* ***(wurde erfüllt)***
Der Fortschritt laufender Runs wird auf der Crawler Run Detailseite angezeigt (5.6.2 Crawler Run Detailseite).

B4.2: *Die Anwendung soll* ***Logs der Runs*** *anzeigen, um eine schnelle erste Analyse bei Fehlern zu ermöglichen.* ***(wurde nicht erfüllt, aber architektonisch konzipiert)***
In der aktuellen Version des Crawler-Management-Systems wurde aus Kapazitätsgründen keine Anzeige von Logs der Crawler-Ausführungen implementiert. Die Umsetzung dieser Anforderung wurde jedoch bei der Konzeptionierung der Architektur miteinbezogen.

B4.3: *Die Anwendung soll* ***Nachrichten der Crawler, klassifiziert nach Dringlichkeit,*** *anzeigen können, sodass die Crawler explizit auf mögliche Probleme hinweisen können und stets erkennbar ist, in welcher Phase der Crawler sich derzeit befindet.* ***(wurde erfüllt)***
Nachrichten der Crawler klassifiziert nach Dringlichkeit werden auf der Crawler Run Detailseite dargestellt (5.6.2 Crawler Run Detailseite).

7.1.3 Anforderungen der Priorität C

Anforderungen der Priorität C stellen Anforderungen dar, welche umgesetzt werden können, um die Nutzbarkeit der Anwendung zu verbessern. Hierbei wurden alle Anforderungen in der Architektur mitkonzipiert und umfangreiche Strategien für deren Umsetzung entworfen (4 Architektur). Im Folgenden werden nur die umgesetzten Anforderungen vorgestellt.

C1.3: *Die Anwendung kann* ***automatisch die Erreichbarkeit von Ziel-URLs überprüfen und Warnungen anzeigen****, um auf wahrscheinliche Probleme bei der Ausführung von Schedules hinzuweisen.* ***(wurde erfüllt)***
Die Erreichbarkeit von Ziel-URLs wird automatisch überprüft und auf der Health Status Übersichtsseite angezeigt (5.5 Health Status Seite und 6.1.3 Targets).

C2.1: *Die Anwendung kann eine* ***feingranulare Rechteverwaltung*** *bieten, um Nutzer auf die Funktionen einzuschränken, für die sie befugt sind.* ***(wurde erfüllt)***
Die Anwendung implementiert eine feingranulare Rechteverwaltung, welche Nutzer explizit auf bestimmte Aktionen auf bestimmten Ressourcentypen einschränkt
(4.6.2 Authentifizierung und Autorisierung).

7.2 Evaluation von Argo Workflows als Orchestrierungs-Framework

In 2.4 Auswahl des Orchestrierungs-Frameworks wurde die Entscheidung getroffen, Argo Workflows als Orchestrierungs-Framework für die Ausführung der Crawler zu verwenden.
Während der Konzeption und Implementierung des Crawler-Management-Systems zeigten sich jedoch einige Einschränkungen, der von Argo bereitgestellten, Funktionalitäten, insbesondere in Bezug auf die API.
Diese werden im Folgenden beschrieben und abschließend bewertet.

7.2.1 Referenzierung von Workflow Templates bei Runs

Bei der Abfrage von Runs über die API wird grundsätzlich das zugehörige Workflow Template referenziert, welches ausgeführt wurde. Diese Referenz wird jedoch nur mitgeliefert, solange das entsprechende Workflow Template im Cluster existiert.
Wird ein Workflow Template gelöscht, enthält die API-Antwort des Runs keine Referenz auf das Workflow Template mehr. Dies ist für die Analyse vergangener Runs sehr problematisch, da im Fehlerfall nicht mehr nachvollzogen werden kann, welcher Ablauf durch das Workflow Template definiert wurde.
Dieses Problem wurde jedoch durch einen Workaround umgangen. In jedem Workflow Template wird nun definiert, dass bei Erstellung eines Runs, automatisch ein Label mit dem Namen des Workflow Templates gesetzt wird. So kann auch nach dessen Löschung der Name des Templates nachvollzogen werden. In Kombination mit der Versionsverwaltung der Workflow Templates im Git-Repository ist somit weiterhin die ausgeführte Logik des Runs nachvollziehbar.

7.2.2 Breaking Change bei Archivierung von Runs

Wie bereits beschrieben, bietet Argo Workflows die Möglichkeit an, Runs zu archivieren, wodurch Runs, auch nach Löschung im Kubernetes Cluster, weiterhin über die Argo API abrufbar sind.

Zu Beginn der Implementierung wurde diese Funktion noch deaktiviert. Als die Funktion jedoch später aktiviert wurde, traten Fehler bei der Anzeige von Runs auf. Ursache dafür war, dass sich der Aufbau des Response-Objekts für Runs maßgeblich verändert hatte und damit von der offiziellen API-Dokumentation abweicht. Dies führte dazu, dass die bestehende Logik zur Abfrage und Verarbeitung von Runs nicht mehr funktionierte und stark angepasst werden musste, was zusätzlichen Entwicklungsaufwand und einen Ausfall der Development-Umgebung zur Folge hatte.

7.2.3 Inkonsistenzen in der Art der Pagination

Die API von Argo Workflows bietet Pagination eigentlich auf Basis von Tokens an. Dabei wird bei Abfrage einer Liste von Ressourcen ein sogenanntes *Continuation Token* zurückgeliefert, welches bei einer nachfolgenden Anfrage wieder mitgegeben werden muss, um die nächste Seite zu erhalten.

Im Laufe der Implementierung zeigte sich jedoch, dass dieses Verhalten nicht konsistent über alle API-Endpunkte mit Pagination in dieser Weise definiert ist.

Bei Abfrage der Workflow Templates wird das erwartete Token zurückgegeben, bei Abfrage der Runs hingegen wird ein *boolean* Wert zurückgeliefert, welcher angibt, ob noch mehr Elemente existieren.

Durch diese Inkonsistenz wird der Implementierungs- und Wartungsaufwand erheblich erhöht, da keine einheitlichen Pagination-Komponenten verwendet werden können.

7.2.4 Pagination nur ohne Sortierung

Der Endpunkt zur Abfrage von Runs bietet zwar grundsätzlich die Möglichkeit zur Filterung der Runs. Jedoch steht diese Funktion nur zur Verfügung, wenn gleichzeitig keine Pagination genutzt wird.

Da im produktiven System jedoch eine sehr große Anzahl an Runs existiert, ist es nicht möglich, alle Runs in einem Request abzurufen, weswegen eine seitenweise Abfrage zwingend erforderlich ist. Daher kann die Sortierungsfunktion der Argo API nicht sinnvoll genutzt und somit keine Sortierfunktion angeboten werden.

Zwar wäre es theoretisch möglich, das Problem zu umgehen, indem direkt auf die, von Argo Workflows verwendete, Datenbank zugegriffen und eine eigene Abfragelogik implementiert werden würde, jedoch führe dies zu erheblichen Nachteilen. Dies würde zu einer starken Kopplung an die interne Datenbankstruktur von Argo führen, welche sich in zukünftigen Versionen ändern könnte. Somit würde der Wartungsaufwand erheblich steigen.

Trotz der beschriebenen Einschränkungen bietet Argo Workflows eine Vielzahl an Funktionen, welche die Implementierung des Crawler-Management-Systems erheblich erleichterten. Insbesondere die direkte Integration in Kubernetes und die Workflow-Orchestrierung reduzierten den Entwicklungsaufwand erheblich.
Die meisten der beschriebenen Einschränkungen konnten durch Workarounds umgangen werden. Zwar bleibt die fehlende Sortierung, jedoch stellt diese keine Kernfunktion des Systems dar.
Hinsichtlich Stabilität und Performance hat sich Argo bisher als zuverlässig und schnell erwiesen, denn während des Beobachtungszeitraums von fünf Monaten kam es zu keinem Systemausfall und API-Anfragen wurden stets mit geringer Latenz beantwortet.
Somit kann die Entscheidung, Argo Workflows als Orchestrierungs-Framework zu verwenden, insgesamt als sinnvoll und positiv bewertet werden.

8 Möglichkeiten zur Weiterentwicklung

Dieses Kapitel stellt mögliche Weiterentwicklungen des Crawler-Management-Systems, sowie weitere Entwicklungsmöglichkeiten im Gesamtkontext des Projektes vor. Hierbei sollen Ansätze erläutert werden, welche im Zuge dieser Arbeit nicht umgesetzt wurden, jedoch einen Mehrwert für das Projekt darstellen würden.

8.1 Weiterentwicklung des Crawler-Management-Systems

In Bezug auf das Crawler-Management-System stellen insbesondere nicht umgesetzte Anforderungen weitere Entwicklungsmöglichkeiten dar.
Diese wurden jedoch bereits im Anforderungskapitel benannt und im Architekturkapitel umfangreich konzeptioniert, weswegen sie hier nicht erneut beschrieben werden.
Stattdessen folgt eine Darstellung von Ideen, die im Laufe der Entwicklung entstanden sind und einen Mehrwert für das Crawler-Management-System darstellen würden.

8.1.1 Workflow zur Beantragung von Zugriffsrechten

Wie bereits beschrieben, bietet das Crawler-Management-System eine feingranulare Rechteverwaltung an. Hierbei erhalten die Nutzer initial nur sehr eingeschränkte Berechtigungen. Weitere Zugriffsrechte können nur entsprechend autorisierte Nutzer vergeben.
Nutzer können ausschließlich die Funktionen und Inhalte verwenden, für die sie autorisiert sind. Stellt ein Nutzer fest, dass ihm für eine bestimmte Aktion die erforderlichen Rechte fehlen, muss er sich im aktuellen System an einen Nutzer mit entsprechenden Nutzeradministrationsrechten wenden. Dieser öffnet anschließend die Nutzermanagement Seite und vergibt die benötigten Rechte manuell.
Dieser Prozess weist jedoch mehrere Nachteile auf. Zum einen entsteht zusätzlicher manueller Aufwand, da die Kommunikation außerhalb des Systems erfolgen muss und zum anderen besteht das Risiko von Missverständnissen und Informationsverlusten.
Um diesen Prozess zu vereinfachen, würde es Sinn machen eine Funktion zur Beantragung von Zugriffsrechten einzuführen. So hätte der Nutzer beispielsweise die Möglichkeit, über eine explizite Seite Rechte anzufordern. Ergänzend wäre es möglich auf Seiten oder bei Funktionen, für die ein Nutzer nicht die erforderlichen Berechtigungen besitzt, einen passenden Hinweis, mit der Möglichkeit zur direkten Beantragung der entsprechenden Rechte, anzuzeigen.
Die Admin Übersichtsseite könnte daraufhin für alle Nutzer mit entsprechenden Rechten einen Bereich anzeigen, in welchem alle unbeantworteten Anfragen von Nutzern angezeigt werden. Dort ließen sich die Anfragen prüfen und anschließend genehmigen oder ablehnen. Durch diesen integrierten Genehmigungsprozess würde der manuelle Kommunikationsweg entfallen und das Risiko von Missverständnissen ließe sich minimieren.

8.1.2 Profilseite

Die aktuelle Version des Systems bietet dem Nutzer keine Möglichkeit zur Änderung des Nutzernamens oder des Passworts. Aus Sicherheitsgründen sollte diese Funktion jedoch bereitgestellt werden, um beispielsweise nach Passwort-Leaks das Passwort ändern zu können.

Zudem werden für Funktionen, wie beispielswiese Alarmierungen, zusätzliche Informationen des Nutzers, wie E-Mail-Adresse oder Slack-Nutzername benötigt.

Dies wäre über eine Profilseite möglich, über welche der Nutzer sowohl private Informationen, wie beispielsweise die Konfiguration der Alarmierungswege, als auch öffentlich sichtbare Informationen, wie Profilfoto oder Rolle im Unternehmen, verwalten könnte.

Die Detailseite einer Schedule zeigt dann zum Beispiel die öffentlich sichtbaren Informationen, um nachvollziehen zu können, wer die Schedule erstellt hat. Im aktuellen System wird die ID des Nutzers, der eine Schedule erstellt hat, bereits gespeichert, sodass die eben beschriebene Funktionalität auch bei älteren Schedules angezeigt werden könnte.

8.1.3 Erweiterung der Rechteverwaltung um organisationsbasierte Zugriffskontrolle

Wie bereits in 4.6.2 Authentifizierung und Autorisierung beschrieben, findet die Rechteverwaltung der Anwendung derzeit ausschließlich auf Aktionen und Ressourcentypen statt. Dadurch kann zwar festgelegt werden, dass ein Nutzer beispielsweise Schedules erstellen, aber nicht löschen darf, jedoch gelten diese Berechtigungen global auf allen Ressourcen des jeweiligen Ressourcentyps.

Eine mögliche Weiterentwicklung des Systems wäre daher die Einführung von organisationsbasierter Rechteverwaltung, sodass Ressourcen bestimmten Organisationseinheiten zuweisbar sind.

Die Beschränkung der Rechte von Nutzern würde dadurch nicht nur auf dem Ressourcentyp, sondern auch auf konkreten Organisationseinheiten basieren.

Dadurch könnten mehrere Teams mit derselben Instanz des Crawler-Management-Systems arbeiten, ohne dabei Zugriff auf die Ressourcen anderer Teams zu erhalten.

Eine mögliche Umsetzung dieser Funktionalität ist über die, auch jetzt schon verwendete, CASL-Bibliothek gut möglich. Dafür müssten die Ressourcentypen um ein Attribut zur Zuordnung einer Organisationseinheit erweitert werden. Die Autorisierung könnte anschließend weiterhin über CASL erfolgen, indem die Regeln um eine entsprechende Bedingung zur Überprüfung der Organisationseinheit ergänzt würden. (CASL, o. D.)

Durch die eben beschriebene Erweiterung wäre die Anwendung für Szenarien mit mehreren unabhängigen Teams deutlich geeigneter, da keine separaten Instanzen betrieben werden müssten. Gleichzeitig bliebe die aktuelle Architektur bestehen, da CASL die Definition von Zugriffsbedingungen unterstützt.

8.1.4 Benutzeroberfläche für Verwaltung von API-Keys

Die Verwaltung von API-Keys findet in der aktuellen Version des Systems lediglich über die bereitgestellte API statt. Diese Entscheidung wurde getroffen, da die Erstellung und Verwaltung von API-Keys ausschließlich von Entwicklern bzw. Administratoren mit technischem Hintergrund durchgeführt werden sollte.
Dennoch würde eine grafische Benutzeroberfläche zur Verwaltung und Erstellung von API-Keys eine komfortable Erweiterung darstellen.
Über eine entsprechende Seite könnten die Metadaten aller API-Keys, wie beispielsweise das letzte Datum der Nutzung, angezeigt sowie die Löschung von API-Keys ermöglicht werden.
Da die dafür notwendigen Endpunkte im Backend bereits implementiert sind, beschränkt sich der Implementierungsaufwand auf die notwendigen Frontend-Komponenten und wäre damit mit geringem Entwicklungsaufwand realisierbar.

8.1.5 Übersicht über alle Datenquellen mit Aktualitätsinformationen

Eine weitere Möglichkeit zur Erweiterung des Crawler-Management-Systems bestände in der Darstellung von Informationen zur Aktualität der Datensätze.
Dadurch könnte schnell erkannt werden, wie aktuell die jeweiligen Datensätze sind, und ob die Aktualisierung nicht regelmäßig genug erfolgt.
Eine Anzeige von Informationen zum Speicherort der Datensätze sowie generierter API-URLs zur Abfrage der Daten könnte ebenfalls bereitgestellt werden.

8.1.6 Automatische Fehleranalyse von Runs

Derzeit muss die Analyse von fehlgeschlagenen Runs manuell durch einen Entwickler erfolgen. Hierfür werden die Logs und gesendeten Events der Crawler genutzt.
Eine mögliche langfristige Erweiterung der Anwendung könnte daher die automatische Analyse von Fehlern auf Basis der Logs und Events der Crawler mithilfe von künstlicher Intelligenz sein.
Dabei müsste die künstliche Intelligenz wiederkehrende Fehlermuster erkennen und entsprechende Lösungsstrategien vorschlagen.
Da diese Lösung jedoch zusätzliche Infrastruktur benötigt sowie erheblichen Implementierungsaufwand erfordert, ist die Umsetzung in nächster Zeit nicht absehbar.

8.2 Entwicklungsmöglichkeiten im Gesamtkontext des JValue-Biomed-Projekts

Abgesehen von den möglichen Erweiterungen des, im Rahmen dieser Arbeit entwickelten, Crawler-Management-Systems, ergeben sich auch im Gesamtkontext des JValue-Biomed-Projekts weitere Entwicklungsmöglichkeiten, welche im Folgenden kurz vorgestellt werden.

8.2.1 Entwicklung weiterer Crawler

Die Umsetzung des Crawler-Management-Systems sowie die Entwicklung des ersten funktionsfähigen Crawlers schuf die Grundlage für die breite Entwicklung weiterer Crawler. Dadurch ist die Erstellung zusätzlicher Crawler möglich, die für weitere Dateiformate oder Serverarten geeignet oder speziell konfigurierbar zur Suche nach bestimmten Datensätzen konzipiert sind.

Durch die vorhandene Infrastruktur, sowie das Crawler-Management-System und den bereits existierenden Crawler, können neue Crawler vergleichsweise einfach erstellt und integriert werden.

8.2.2 Entwicklung eines MCP-Servers zur Abfrage der Daten

Neben der Speicherung der Datensätze, stellt auch deren Bereitstellung, für zum Beispiel KI-Systeme, einen wichtigen Bestandteil des Gesamtprojekts dar.

Über einen sogenannten Model Context Protocol Server (MCP) könnten die gesammelten Datensätze strukturiert abgefragt werden.

Ein MCP-Server dient als standardisierte Schnittstelle zur Abfrage von Daten für KI-Systeme. Dadurch könnten beispielsweise Large Language Models gezielt relevante Datensätze abrufen und in ihre Verarbeitung miteinbeziehen.

Die Entwicklung eines solchen MCP-Servers würde es ermöglichen die, von den Crawlern, gesammelten Daten einfach für KI-Systeme bereitzustellen. Zudem gäbe es durch die standardisierte Schnittstelle eine wiederverwendbare Möglichkeit zum Abruf der Daten.

8.2.3 Bereinigung der Datensätze

Neben dem Umfang der zur Verfügung stehenden Datensätze, sowie deren Aktualität stellt auch die Qualität der Datensätze eine wichtige Voraussetzung für qualitativ hochwertige KI-Systeme dar. Fehlerhafte, inkonsistente oder verzerrte Datensätze führen dabei direkt zu unzuverlässigen KI-Modellen. Laut Mohammed et. al (2025) haben insbesondere die Qualitätsmerkmale Vollständigkeit, Feature-Genauigkeit und Zielgenauigkeit einen hohen Einfluss auf die Qualität des KI-Systems.

Daher könnte die systematische Bereinigung und damit verbundene Qualitätssicherung der gesammelten Datensätze eine weitere Entwicklung im Kontext des JValue-Biomed-Projekts darstellen.

Dafür müsste ein System entwickelt werden, welches in regelmäßigen Abständen die Datensätze überprüft und ferner anpasst, um beispielsweise Duplikate oder fehlerhafte Datensätze zu entfernen sowie fehlende Metadaten zu ergänzen.
Eine solche Verbesserung der Datengrundlage würde für KI-Systeme, wie auch für andere Anwendungen einen erheblichen Mehrwert darstellen, da diese stark von zuverlässigen und konsistenten Datensätzen abhängen.

9 Schlussfolgerung

Zentrales Ziel dieser Arbeit war die Konzeption und Entwicklung eines webbasierten Crawler-Management-Systems für das JValue-Biomed-Projekt. Das System sollte dabei auf einem bestehenden Orchestrierungs-Framework aufbauen, ohne eng an dieses gekoppelt zu sein. Dadurch sollte eine strukturierte Orchestrierung verteilter Crawler, sowie die Bereitstellung detaillierter domänenspezifischer Metriken und Auswertungen ermöglicht werden.

Zu Beginn der Arbeit wurden hierfür zunächst Anforderungen an das zugrundeliegende Orchestrierungs-Framework gestellt. Auf Basis dieser folgte ein Vergleich und eine Bewertung verschiedener Orchestrierungs-Frameworks. Die Entscheidung fiel dabei auf Argo Workflows, da dieses Framework die definierten Anforderungen am besten erfüllte.

Darauf aufbauend wurde die begründete Entscheidung getroffen, die Nutzeroberfläche, die bereits für die Darstellung der Metriken und Auswertungen geplant wurde, um Funktionalitäten zur Orchestrierung der Crawler zu erweitern. Anstatt die von Argo bereitgestellte Benutzeroberfläche zu verwenden, wurden nur die umfangreichen Argo-Backendfunktionalitäten genutzt, wobei das Crawler-Management-Backend als Abstraktionsschicht hin zum Crawler-Management-Frontend dient.

Die darauffolgend definierten Anforderungen an das zu entwickelnde Crawler-Management-System dienten als Grundlage für eine umfangreiche Systemarchitektur, welche alle definierten Anforderungen berücksichtigt und daher die Grundlage für eine Umsetzung dieser bildete. Die Architektur folgt dabei der Struktur der arc42-Architekturdokumentation.

Im Anschluss folgte ein kurzes Kapitel über das Frontend-Design der Anwendung sowie ein Implementierungskapitel, in dem besondere Aspekte der Implementierung vorgestellt wurden.

Im anschließenden Rückblick auf die definierten Anforderungen zeigte sich, dass alle 17 Anforderungen der Priorität A erfüllt wurden, sowie 6 von 8 Anforderungen der Priorität B und 2 von 14 Anforderungen der Priorität C. Diese Arbeit übertraf das für sie definierte **Minimal Viable Product** somit bei Weitem. Zudem erfolgte noch die architektonisch umfangreiche Ausarbeitung aller insgesamt 39 definierten Anforderungen.

Abschließend wurden Weiterentwicklungsmöglichkeiten des Crawler-Management-Systems sowie des Gesamtprojekts vorgestellt.

Insgesamt wurde im Rahmen dieser Arbeit ein webbasiertes Crawler-Management-System konzipiert und entwickelt, welches auf Argo Workflows aufbaut, jedoch nicht eng an dieses gekoppelt ist. Das System bildet damit die Grundlage für die Orchestrierung, Überwachung und Auswertung von Crawlern im JValue-Biomed-Projekt und unterstützt somit den effizienten und zuverlässigen Einsatz von Crawlern. Eine solche Datenbasis stellt einen zentralen Faktor für die Qualität von KI-Systemen dar.

Literaturverzeichnis

Apache Airflow. (29. 08 2025 - a). *Apache Airflow Documentation*. Von https://airflow.apache.org/docs/apache-airflow/3.1.8/ abgerufen

Apache Airflow. (21. 03 2025 - b). *Apache Airflow GitHub License*. Von https://github.com/apache/airflow/blob/main/LICENSE abgerufen

Apache Software Foundation. (kein Datum). *Apache Airflow Project*. Von https://projects.apache.org/project.html?airflow (Zuletzt aufgerufen am 01.03.2026) abgerufen

Argo Workflows. (06. 04 2018). *Argo Workflows GitHub License*. Von https://github.com/argoproj/argo-workflows/blob/main/LICENSE abgerufen

Argo Workflows. (o. D.). *Argo Workflows Documentation*. Von https://argo-workflows.readthedocs.io/en/latest/ (Zuletzt aufgerufen am 03.03.2026) abgerufen

Astronomer. (o. D. - a). *Airflow Concepts - Rerun Airflow DAGs and tasks*. Von https://www.astronomer.io/docs/learn/rerunning-dags (Zuletzt aufgerufen am 01.03.2026) abgerufen

Astronomer. (o. D. - b). *Apache Airflow Resilience: Guide to running highly available data pipelines*. Von https://www.astronomer.io/airflow/resilience/ (Zuletzt aufgerufen am 01.03.2026) abgerufen

auth0. (o. D.). *Auth0 - Authentifizierung und Autorisierung im Vergleich.* Von https://auth0.com/de/intro-to-iam/authentication-vs-authorization (Zuletzt aufgerufen am 06.03.2026) abgerufen

Azure. (06. 03 2026). *Azure Architecture Center - Publisher-Subscriber pattern* . Von https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber abgerufen

CASL. (o. D.). *CASL - Documentation Guide.* Von https://casl.js.org/v6/en/guide/conditions-in-depth (Zuletzt aufgerufen am 01.03.2026) abgerufen

Chaurasia, A. (22. 01 2026). *Airflow Survey 2025*. Von https://airflow.apache.org/blog/airflow-survey-2025/ abgerufen

Cloud Native Computing Foundation. (o. D.). *Project Argo*. Von https://www.cncf.io/projects/argo/ (Zuletzt aufgerufen am 02.03.2026) abgerufen

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design Patterns: Abstraction and Reuse of Object-Oriented Design. *ECOOP'93 - Object-Oriented Programming, 7th European Conference.* Kaiserslautern, Deutschland.

Google. (02 2017). *Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed.* Von https://think.storage.googleapis.com/docs/mobile-page-speed-new-industry-benchmarks.pdf abgerufen

Kosinski, M., Clark, B., & Scapicchio, M. (2026). *IBM - Was ist Authentifizierung?* Von https://www.ibm.com/de-de/think/cybersecurity#605511093 abgerufen

Kubernetes. (o. D.). *Kubernetes Documentation*. Von https://kubernetes.io/docs (Zuletzt aufgerufen am 01.03.2026) abgerufen

Martin, R. C. (2003). *Agile Software Development.* Prentice Hall PTR.

MDN. (08. 10 2025). *Mozilla Developer Network - Using HTTP cookies* . Von https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies abgerufen

Meyer, B. (1997). *Object-oriented Software Construction, 2nd Edition.* Prentice Hall International.

Mohammed, S., Budach, L., Feuerpfeil, M., Ihde, N., Nathansen, A., Noack, N., . . . Harmouch, H. (2025). The effects of data quality on machine learning performance on tabular data. *Information Systems - Volume 132*.

Open Source Initiative. (22. 03 2007). *The Open Source Definition*. Von https://opensource.org/osd abgerufen

Parmar, D., Gupta, P., Chouhan, C., & Saran, H. (2023). Data-centric AI: Prioritizing Data Quality Over Model Complexity. *International Journal of Innovative Research in Computer and Communication Engineering. 11*.

Prefect. (08. 03 2022). *Prefect GitHub Lizenz*. Von https://github.com/PrefectHQ/prefect/blob/main/LICENSE abgerufen

Prefect. (o. D. - a). *Prefect Customers*. Von https://www.prefect.io/customers (Zuletzt aufgerufen am 01.03.2026) abgerufen

Prefect. (o. D. - b). *Prefect Documentation*. Von https://docs.prefect.io/v3 abgerufen

Rangoola, M. K. (20. 07 2023). *AIRFLOW MONITORING: MASTERING SLAS, DAGS, & OBSERVABILITY*. Von https://www.astronomer.io/blog/expert-tips-for-monitoring-the-health-and-slas-of-your-apache-airflow-dags abgerufen

Starke, G., & Hruschka, P. (o. D.). *arc42 Documentation*. Von https://docs.arc42.org/home/ (Zuletzt aufgerufen am 01.03.2026) abgerufen

The OWASP Foundation. (2025). *A01:2025 Broken Access Control.* Von https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/ abgerufen

Valavi, E., Hestness, J., Ardalani, N., & Iansiti, M. (2021). Time and the value of data. *Harvard Business School Strategy Unit Working Paper No. 21-016*.