SCA Tool: API für automatisierte CI Workflows

MASTER THESIS

Emanuel Erben

Eingereicht am 25. Juli 2025

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

Betreuer:
Martin Wagner, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Technische Fakultät

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, 25. Juli 2025

Lizenz

Erlangen, 25. Juli 2025

Disclaimer

Das in dieser Arbeit gewählte generische Maskulinum bezieht sich zugleich auf die männliche, die weibliche und andere Geschlechtsidentitäten. Zur besseren Lesbarkeit wird auf die Verwendung anderer Sprachformen verzichtet. Alle Geschlechteridentitäten werden ausdrücklich mitgemeint, soweit die Aussagen dies erfordern.

Abstract

The goal of this thesis was to develop an automation that enables developers to seamlessly integrate SCA Tool into their development workflows. To this end, an OpenAPI-compliant CI/CD-Integration was implemented for both the GitHub and GitLab platforms. A user-friendly guide was created to support the configuration of the provided pipelines. For the safe communication with the backend, an authentication mechanism based on API keys was introduced.

A TypeScript script was developed to serve as the integration core, enriched with platform-specific extensions for improved usability. On GitHub, the script was used to implement a custom JavaScript-based GitHub-Action, which was published on the GitHub-Marketplace. For GitLab, a GitLab-Component was developed that loads the script and executes it within a Node.js container. This component is also publicly available via the GitLab CI/CD-Catalog. The project placed particular emphasis on the secure generation and handling of API keys, as well as on a maintainable and extensible architecture. Thanks to the modular structure of the integration script and the server components, the solution can be easily adapted to meet future requirements.

Zusammenfassung

Das Ziel der Arbeit war es, die Nutzung des SCA Tool über eine OpenAPI-konforme CI/CD-Anbindung der Plattformen GitHub und GitLab zu automatisieren. Dabei wurde eine serverseitige Authentifizierungsmethode auf Basis von API-Keys hinzugefügt. Ergänzend wurde eine benutzerfreundliche Anleitung zur Konfiguration der bereitgestellten CI/CD-Pipelines erstellt.

Für die Integration wurde ein TypeScript-Skript entwickelt, das durch plattformspezifische Anpassungen eine optimierte Nutzbarkeit bietet. Auf GitHub wurde dieses Skript in einer JavaScript-basierten GitHub-Action genutzt und im GitHub-Marketplace veröffentlicht. Für GitLab entstand eine dedizierte Komponente, die das Skript in einem Node.js-Container ausführt und die über den GitLab CI/CD-Catalog öffentlich verfügbar ist. Ein besonderer Fokus lag auf der sicheren Handhabung und Generierung der API-Keys sowie auf einer wartungsfreundlichen Architektur. Durch die Modularität des Integrationsskripts und der Serverkomponenten ist eine einfache Anpassung an zukünftige Anforderungen gewährleistet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Glossar

API	Ein Application Programming Interface (API) ist eine Programmierschnittstelle zu einem dahinterliegenden System, über das andere Programme vordefinierte Funktionalitäten nutzen und Daten hochladen oder abfragen können.
Integration	Im Rahmen dieser Arbeit ist mit Integration sowohl die GitHub-Action als auch die GitLab-Component gemeint.
Projekt	Ein Projekt ist vergleichbar mit einem Repository. Es enthält Konfiguration der Datenquelle und bündelt mehrere VerUnits zusammen. In der Datenbank ist es als `projects` zu finden.
Referenz	Eine Referenz oder auch Ref wird zur Strukturierung mehrerer Versionen eines Projektes genutzt. In der Datenbank sind Referenzen in der Tabelle `refs` zu finden. Eine Referenz kann dabei etwa ein Branch oder Tag sein.
REST-API	Eine Representational State Transfer (REST)-API ist eine spezielle Form einer API, die sich an bestimmte Architektureinschränkungen hält. Die wichtigste Eigenschaft ist dabei eine zustandslose Kommunikation, bei der jede Abfrage einen eigenen Kontext hat und nicht vom Kontext einer anderen Abfrage abhängig ist.

Version	Eine Version oder auch VerUnit stellt einen spezifischen Stand eines hochgeladenen Repositories zu einem gewissen Zeitpunkt dar. Sie ist einem Projekt zugeordnet und ist in der Datenbank in der Tabelle `ver_units` zu finden.

Akronyme

CI Continous Integration

CD Continuous Delivery beziehungsweise Continuous Deployment

SHA Secure Hashing Algorithm

HMAC Hash-based Message Authentication Code

API Application Programming Interface

REST Representational State Transfer

1 Einführung

In diesem Kapitel wird zunächst eine kurze Motivation für das Thema dieser Arbeit gegeben. Danach wird die Zielsetzung für das Projekt definiert und der Aufbau der Arbeit erklärt.

1.1 Motivation

Die Entwicklung von Softwareprodukten ist ein vielschichtiger Prozess, bei dem zahlreiche Aspekte berücksichtigt werden müssen. Die Aufrechterhaltung einer hohen Codequalität, die Sicherstellung einer robusten und skalierbaren Architektur, die zeitnahe Auslieferung benutzerzentrierter Funktionalitäten sowie die Gewährleistung der korrekten Implementierung unter oft straffen Zeitplänen sind nur einige der vielen Anforderungen.

Ein zentraler Aspekt zur Bewältigung dieser Komplexität und zur Beschleunigung des Entwicklungsprozesses ist die Integration von bestehenden Lösungen oder Bibliotheken. Dabei werden Abhängigkeiten, sogenannte Dependencies, in das eigene Projekt eingefügt. Diese externen Komponenten können die Entwicklung erheblich beschleunigen und vereinfachen, indem sie vorgefertigte Funktionalitäten bereitstellen und die Notwendigkeit einer eigenen Entwicklung von Standardlösungen, etwa Boilerplate-Code, reduzieren. Die Auswahl geeigneter, kompatibler Lösungen, die den gewünschten Funktionsumfang abdecken und sich nahtlos in die bestehende Systemarchitektur einfügen, stellt dabei bereits einen nicht zu unterschätzenden Aufwand dar.

Über den initialen Auswahlprozess hinaus geraten jedoch häufig zwei kritische Aspekte in den Hintergrund, deren Vernachlässigung weitreichende Konsequenzen nach sich ziehen kann: die Identifikation und Einhaltung von Lizenzbestimmungen sowie die Identifikation und das Management bekannter Sicherheitsschwachstellen in den eingebundenen Abhängigkeiten.

Die Komplexität wird dabei durch indirekte Abhängigkeiten erhöht. So können eingebundene Bibliotheken ihrerseits wiederum weitere Dependencies enthalten. Dies führt zur Entstehung eines komplexen, oft schwer durchschaubaren Abhän-

gigkeitsgraphen. Ein vollständiges Bild aller direkt und indirekt genutzten Komponenten, sowie deren jeweiliger Lizenzen und bekannter Schwachstellen manuell zu erfassen und aktuell zu halten, ist für viele moderne Softwareprojekte praktisch unmöglich geworden. Eine Untersuchung der Firma Snyk in Kooperation mit der Linux Foundation (Snyk, 2022) ergab, dass ein durchschnittliches Softwareprojekt 79 direkte Dependencies hat, wobei diese Zahl bei JavaScript-Projekten sogar bei 174 lag. Im Durchschnitt haben die von ihnen gescannten Projekte dabei 49 Sicherheitsschwachstellen (Vulnerabilities).

Hier setzt SCA Tool an. Es erstellt eine vollständige Liste aller direkt und indirekt genutzten Abhängigkeiten eines Softwareprojektes. Im Anschluss analysiert es die identifizierten Komponenten, um bekannte Vulnerabilities aufzudecken und die verwendeten Lizenzen aufzulisten. Dabei wird eine umfassende Übersicht generiert, die Entwicklerteams dabei unterstützt, bekannte Probleme frühzeitig zu erkennen und geeignete Maßnahmen zu deren Behebung einzuleiten.

Der Stand des SCA Tool vor dieser Arbeit erforderte, dass Nutzer einen Scan ihres Projektes jedes Mal manuell über einen Upload des Codes ihres Projektes, etwa einer Webanwendung, von GitHub auf der SCA Tool Webseite starten mussten, um es analysieren zu lassen. Auch wenn nur ein erneuter Scan gewünscht ist, muss dieser zunächst manuell konfiguriert und gestartet werden. Dies stellt einen zusätzlichen Aufwand dar, der die Arbeitsabläufe der Entwickler unterbricht. Zusätzlich wird jedes Mal nur eine Momentaufnahme erstellt. Wenn diese nicht regelmäßig erneuert wird, können Probleme längere Zeit unbemerkt bleiben.

Das frühe Erkennen von Vulnerabilities ist jedoch ein wichtiger Aspekt, um den Aufwand und die Kosten in der Entwicklung gering zu halten. Je später im Softwareentwicklungsprozess eine Schwachstelle entdeckt wird, umso teurer wird deren Beseitigung. So kann die Behebung einer Vulnerability, die während des laufenden Betriebs entdeckt wird, das 100-fache kosten, als wenn sie während der Design-Phase gefunden wird. Wird sie während des Testings entdeckt, reduzieren sich die Kosten auf das 15-fache und während der Implementierung sogar auf das 6,5-fache. Daher ist es wichtig, möglichst früh Schwachstellen oder Lizenzprobleme zu identifizieren, die ein Auswechseln und damit verbundenes Umbauen der Software erfordern. (Maurice Dawson et al., 2010)

Das Ziel dieser Arbeit ist es daher, den zuvor beschriebenen manuellen Prozess zu automatisieren und somit dem Nutzer regelmäßige frühe Feedbacks anbieten zu können. Zusätzlich wird die Effektivität und die Akzeptanz von SCA Tool erhöht, indem es nahtlos in die täglichen Arbeitsabläufe von Entwicklungsteams integriert werden kann. Es soll dabei möglich sein, den Code automatisch zu scannen, ohne dass der Nutzer manuell eingreifen muss. Dies soll durch die Integration des SCA Tool in die CI/CD-Pipelines von GitHub und GitLab geschehen, die automatisch nach vordefinierten Events, Tests oder auch Veröffentlichungen der Software starten können. Diese Automatisierung spart Zeit, reduziert Fehler-

quellen bei der Konfiguration und trägt dazu bei, die Sicherheit und Qualität von Softwareprojekten kontinuierlich zu überwachen und zu verbessern.

1.2 Zielsetzung und Aufbau der Arbeit

In dieser Arbeit wird die Integration von SCA Tool in bestehende Entwicklungs- und Deployment-Workflows beschrieben. Dabei soll eine Verbindung zwischen den Code-Hosting-Plattformen GitHub (mittels GitHub-Action) und GitLab (mittels GitLab-Component) mit dem SCA Tool über eine eigene API-Schnittstelle hergestellt werden. Über diese Verbindung sollen automatisiert neue Versionen der Codebasis von der externen Plattform hochgeladen und ein Scan gestartet werden können.

Dabei muss eine Authentifikationsmöglichkeit geschaffen werden, über die hochgeladene Daten eindeutig zugeordnet werden können. Außerdem soll der Benutzer die volle Kontrolle darüber haben, welche Daten mit SCA Tool geteilt werden. Dafür soll es möglich sein, den Umfang eines Repositories zu konfigurieren, indem Dateien und Ordner ausgeschlossen werden können. Zusätzlich soll die Integration eine Option bieten, einen Link zu den entsprechenden Scan-Ergebnissen bereitzustellen.

Die Konfiguration der Integration soll intuitiv gestaltet und durch die Benutzeroberfläche des SCA Tool unterstützt werden. Eine Schritt-für-Schritt-Anleitung soll den Benutzern helfen, die Integration einzurichten. Zusätzlich muss die Benutzeroberfläche so angepasst werden, dass zwischen mehreren Versionen eines Projektes navigiert werden kann.

Die Arbeit ist in 9 Kapitel unterteilt. Nach dieser initialen Motivation und Einleitung werden in Kapitel 2 die grundlegenden Technologien erklärt. In Kapitel 3 werden vergleichbare Systeme analysiert. Danach werden in Kapitel 4 die Anforderungen an die Software definiert. Kapitel 5 zeigt die Architektur des Systems und die dazugehörigen Komponenten auf. In Kapitel 6 werden anschließend die wichtigsten Designentscheidungen und Abläufe der erstellten Integration dargestellt, deren Implementierung in Kapitel 7 folgt. Abschließend werden in Kapitel 8 die zuvor festgelegten Anforderungen evaluiert, bevor in Kapitel 9 ein Fazit gezogen wird.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Technologien und Konzepte erklärt, die im Rahmen dieser Arbeit genutzt werden.

2.1 OpenAPI konforme REST-API

Ein Ziel dieser Arbeit ist es, die bestehende API zu einer OpenAPI-konformen REST-API umzuwandeln. OpenAPI hat sich als Standard zur Beschreibung von APIs etabliert. In einer Umfrage der Firma Smartbear mit 1500 Entwicklern gaben 82% der Befragten an, dass ihre Organisation den OpenAPI-Standard zur Definition von APIs nutzt. (SmartBear, 2020) OpenAPI beschreibt API-Methoden und Metadaten, was das Verständnis und die Portabilität von APIs über verschiedene Umgebungen hinweg verbessert. Dabei ermöglicht es, die Fähigkeiten einer API zu erfassen, ohne den Quellcode anschauen zu müssen. Ursprünglich als Swagger bekannt, ist OpenAPI heute ein herstellerneutraler Standard, der von der OpenAPI Initiative, einem Projekt der Linux Foundation, betrieben wird.

Typischerweise wird die Spezifikation in JSON oder YAML verfasst. Ein Beispiel für eine OpenAPI-Spezifikation ist in Listing 2.1 zu sehen.

Listing 2.1: Beispielhafte OpenAPI Spezifikation

```yaml
openapi: 3.0.3
info:
  title: Beispiel API
  version: 1.0.0
servers:
  - url: https://api.example.com/v1
paths:
  /versions:
    get:
      summary: Retrieve a list of versions
      responses:
        '200':
          description: A list of versions
```

```
14            content:
15              application/json:
16                schema:
17                  type: array
18                  items:
19                    $ref: '#/components/schemas/Version'
20  components:
21    schemas:
22      Version:
23        type: object
24        properties:
25          id:
26            type: integer
27            format: int64
28          name:
29            type: string
30          url:
31            type: string
```

Eine Spezifikation besteht dabei aus mehreren Teilen. Als erstes kommt die Version des OpenAPI-Standards, die verwendet wird. Danach folgt die Beschreibung der API (info-Objekt), die unter anderem den Titel und die Version der API enthält. Diese zwei Angaben müssen laut Spezifikation verpflichtend anwesend sein. In Listing 2.1 folgt als nächstes eine Liste der Server. Durch diese Angabe können die verschiedenen Systeme aufgelistet werden, auf denen diese API genutzt werden kann. Außerdem kann darüber ein Basispfad für die Endpunkte festgelegt werden. In diesem Beispiel ist jeder Endpunkt mit dem Prefix 'v1' versehen. Im Anschluss folgt die Beschreibung der Endpunkte (paths), die die API zur Verfügung stellt. Dabei wird für jeden Endpunkt die HTTP-Methode (GET, POST, PUT, DELETE) angegeben, die verwendet werden kann, sowie eine Beschreibung des Endpunkts. Zusätzlich werden die Parameter und die Struktur der Anfrage (requestBody) und der Antworten (responses) beschrieben. Die Parameter können dabei sowohl in der URL, als Header, Query-Parameter oder Cookie übergeben werden. Die Rückgabewerte und Anfragekörper können in Form von Schemata referenziert werden, die im components-Bereich festgelegt werden. Ebenfalls können Sicherheitsanforderungen, wie Authentifizierungsmethoden, definiert werden. Seit Version 3.0 werden auch Callbacks und seit 3.1 Webhooks unterstützt, um asynchrone Operationen zu ermöglichen. Am Ende der Spezifikation kann ein components-Bereich wiederverwendbare Definitionen für Schemata, Parameter, Antworten etc. enthalten. Durch solch ein Schema ist die Struktur der Daten definiert. Diese Schemata bauen auf Konzepten der JSON Schema Spezifikation auf, um Datentypen und Validierungsregeln festzulegen. (OpenAPI Initiative, 2024)

Neben dem Vorteil einer standardisierten API-Beschreibung bietet OpenAPI eine gute Grundlage für die Dokumentation von Schnittstellen. Die Spezifikation kann nicht nur als Datei vorliegen, sondern auch in Form einer interaktiven Dokumentation bereitgestellt werden. Diese listet alle Endpunkte und ihre Beschreibungen übersichtlich auf. Sowohl Entwickler in einer IDE wie IntelliJ als auch Nutzer über ein bereitgestelltes Webinterface können so auf die Dokumentation zugreifen. Ein weiterer zentraler Vorteil liegt in der Möglichkeit zur automatischen Codegenerierung. Mithilfe von Tools wie Swagger Codegen[1] kann sowohl Client- als auch Backendcode für eine Vielzahl von Programmiersprachen - darunter Java, Python oder JavaScript - erzeugt werden. Dies reduziert den Entwicklungsaufwand und verringert gleichzeitig das Fehlerrisiko bei der Anbindung von Komponenten. Darüber hinaus erleichtert OpenAPI in Kombination mit einer Versionierung der API die Nachvollziehbarkeit von Änderungen. Durch die Änderung an einer zentralen Datei wird klar ersichtlich, welche Anpassungen vorgenommen wurden, ohne den zugrundeliegenden Programmcode analysieren zu müssen. Auch die Testbarkeit wird verbessert: Bereits vor der tatsächlichen Implementierung lassen sich mithilfe von Mock-Daten, Server oder Clients simulieren. Zudem ermöglichen präzise definierte Spezifikationen die automatische Generierung von Tests, die die Einhaltung des erwarteten Verhaltens der API überprüfen. (Kin Lane & Steef-Jan Wiggers, 2022)

2.2 CI/CD-Integration

GitHub Workflows und GitLab Pipelines sind die Integrationen von CI/CD in die jeweiligen Codehosting-Plattformen. Continous Integration (CI) mit Continuous Delivery beziehungsweise Continuous Deployment (CD) hat immer mehr an Bedeutung gewonnen. Automatisierte Abläufe, wie Testen, Kompilieren, Codequalitätschecks, die Veröffentlichung neuer Versionen, aber auch die Kommunikation mit externen Plattformen sind ein möglicher Teil einer solchen Pipeline. (Decan et al., 2022)

Im Folgenden werden die Grundlagen zur Erstellung einer CI/CD-Integration für die Plattformen GitHub und GitLab erklärt.

2.2.1 GitHub-Action

GitHub-Workflows werden in YAML-Dateien definiert und können auf verschiedene Ereignisse reagieren, wie z.B. das Erstellen eines Pull Requests oder das Pushen von Code in ein Repository. Sie werden auf sogenannten Runnern ausgeführt, die entweder von GitHub bereitgestellt oder selbst gehostet werden können. Ein Workflow besteht aus einem oder mehreren Jobs, die parallel oder sequenziell

[1] https://swagger.io/tools/swagger-codegen/

ausgeführt werden können. Ein Job besteht aus einem oder mehreren Schritten (Steps), die nacheinander ausgeführt werden. Ein Schritt kann entweder ein ausführbares Skript oder eine Action sein. (GitHub, 2025b)

Eine GitHub Action ist ein wiederverwendbarer Codebaustein, der in einem Workflow verwendet werden kann. Eine Action ist meist definiert, um komplexe, häufig ausgeführte Aufgaben abzubilden. Sie kann dabei vordefiniert oder selbst erstellt sein. Beispiele für vordefinierte Actions, die häufig genutzt werden, sind etwa das Abrufen des Repositories, Uploads von Artefakten des Workflows oder Setups für spezifische Programmiersprachen. (Decan et al., 2022, siehe Tabelle V)

Es gibt drei Möglichkeiten, um eine eigene Action zu erstellen. Diese sind Composite-Action, Docker-Container-Action und JavaScript-Action. (GitHub, 2025a)

Bei einer Composite-Action wird eine eigene GitHub-Action als Kombination bestehender Actions und Shellscripts erstellt. Das Ziel ist es, die mehrfache Definition immer wieder genutzter Schritte zu verhindern. Wenn etwa in einer Firma in jedem Projekt und in jedem Workflow die gleichen drei Schritte ausgeführt werden, so kann dies in einer eigenen Composite-Action definiert werden. So können Änderungen an diesen Standardschritten effizient umgesetzt werden, ohne jeden Workflow einzeln anzupassen. Um eine derartige Action zu erstellen, muss eine yaml-Datei erstellt werden. In dieser werden neben den In- und Outputs die Schritte der Action, wie bei einer normalen Workflow-Definition, definiert. So müssen lediglich die richtigen Actions gefunden und eventuell mit eigenen Shell Scripts angereichert werden. Dadurch ist diese Art der Erstellung einer eigenen Action auch sehr gut für Anfänger geeignet. (GitHub, 2025c)

Beim zweiten Ansatz wird Docker genutzt, wodurch die Action in einer vordefinierten Umgebung (Container) ausgeführt wird. Dafür muss neben dem Action Metadata File mit den Inputs und Outputs ein Dockerfile definiert werden. Dort wird das Basis-Image festgelegt und benötigte Dateien, wie etwa Skripte aus dem Action Repository hochgeladen. Zusätzlich wird ein Entrypoint-Skript spezifiziert, das ausgeführt wird, wenn der Container gestartet wird. Ein Vorteil dieses Ansatzes ist, dass durch die Nutzung von Docker spezifische Versionen von Betriebssystemen, Tools und Dependencies festgelegt werden können. Dies führt zu zuverlässig reproduzierbaren Ergebnissen. Durch das Laden und Bauen des Dockercontainers hat dieser Ansatz jedoch eine höhere Latenz als etwa eine JavaScript-Action. (GitHub, 2025a, 2025d)

Die dritte Option ist es, eine Action mit Hilfe von JavaScript zu programmieren. Dabei kann sowohl JavaScript als auch TypeScript als Programmiersprache genutzt werden, indem das TypeScript vor Release zu JavaScript transpiliert wird. Im Action Metadata File wird hier zusätzlich zu den In- und Outputs das JavaScript definiert, das aufgerufen werden muss. Neben der schnellen Ausführung bietet GitHub eigene npm-Packages zur einfacheren Nutzung der Workflow-

Umgebung an, die direkt im Code benutzt werden können. So stehen Interfaces für die direkte Nutzung von Workflow-Befehlen, Ein- und Ausgabevariablen oder auch des Programmrückgabecodes zur Verfügung. Zusätzlich wird auch ein authentifizierter REST-Client und Zugang zum Action-Context zur Verfügung gestellt. Dadurch entsteht eine starke Integration mit dem GitHub-System, wodurch Parameter wie der aktuelle Commit-Hash ausgelesen werden können oder mit GitHub kommuniziert werden kann, um etwa Kommentare oder Issues zu erstellen. (GitHub, 2025e)

Bei der Wahl der Art der Action spielt neben der Funktionalität und präferierten Programmiersprachen auch die Kompatibilität mit den Betriebssystemen der ausführenden Runner eine Rolle. Während JavaScript und Composite Actions auf allen von GitHub bereitgestellten Runnern laufen, werden Docker Actions offiziell nur auf Linux-Runnern unterstützt. (GitHub, 2025a)

Nachdem eine Action erstellt wurde, kann sie privat in einem eigenen Repository oder innerhalb einer Organisation genutzt werden. Um eine Action jedoch für alle Nutzer gut findbar und nutzbar zu machen, kann sie auf dem GitHub Marketplace veröffentlicht werden. Dafür muss die Action in einem öffentlichen Repository liegen, eine README-Datei mit einer Beschreibung der Action und eine Action.yml-Datei, die die Action beschreibt, bereitgestellt werden. Zusätzlich muss eine License-Datei erstellt werden, die die Lizenzbedingungen der Action beschreibt. Nach einer erfolgreichen Veröffentlichung wird die Action im GitHub Marketplace angezeigt und kann von anderen Nutzern verwendet werden. Jeder Release auf dem Marketplace ist mit einer Versionsnummer versehen. Dies ermöglicht es, dass Nutzer die Action zu einem passenden Zeitpunkt aktualisieren können und eine Änderung der Parameter nicht automatisch zu einem Fehler mit alten Konfigurationen führt. (GitHub, 2025f)

2.2.2 GitLab-Component

GitLab nutzt für automatisierte Abläufe sogenannte Pipelines, die in einer YAML-Datei definiert werden. Es gibt dabei verschiedene Arten von Pipelines. Neben den normalen Basic-Pipelines, die mit jedem Commit ausgeführt werden, gibt es beispielsweise Merge-Request-Pipelines, die nur bei einem Commit innerhalb eines Merge-Requests laufen. Ein weiteres nützliches Beispiel sind Merged-Request-Pipelines. Diese laufen, nachdem ein Merge-Request gemerged wurde. Alle Pipelines eines Repositories werden dabei jedoch in einer einzelnen YAML-Datei definiert. Durch Regeln wird definiert, welche Teile der Pipelines tatsächlich bei einer auslösenden Aktion ausgeführt werden. (GitLab, 2025c)

Ähnlich zu GitHub sind auch Pipelines unterteilt. Eine Pipeline besteht dabei aus unterschiedlichen Jobs, die in verschiedenen Stages unterteilt sind. Jobs innerhalb einer Stage werden parallel ausgeführt, während Stages sequenziell aus-

geführt werden. Ein Job der keiner spezifischen Stage zugeordnet wurde, wird standardmäßig der `test` Stage zugeordnet. (GitLab, 2025c)

Auch in GitLab können vordefinierte Elemente in der CI/CD Pipeline genutzt werden. Dabei können sowohl private als auch veröffentlichte GitLab Components genutzt werden. Ein Unterschied zu GitHub ist dabei, dass CI/CD Components theoretisch neben einzelnen Jobs oder Stages auch ganze Pipelines enthalten können. Eine Component wird dabei selbst wieder in einer YAML-Datei definiert, die vom Nutzer mit Hilfe eines Include-Befehls in seine eigene Pipeline-Definition integriert werden kann. Eine Component wird in einem eigenen Component-Repository erstellt. Es können jedoch mehrere Components pro Repository enthalten sein. Eine Component kann Eingabeparameter haben, die es dem Nutzer erlauben, dynamisch Daten für eine Konfiguration zu setzen. Components können außerdem im CI/CD-Catalog veröffentlicht werden, was neben einer einfacheren Entdeckbarkeit auch eine einfache Versionierung der veröffentlichten Komponente erlaubt. (GitLab, 2025a)

Ausgeführt werden die Jobs ebenfalls auf Runnern, die entweder von GitLab bereitgestellt werden oder selbst gehostet werden können. Ein Job hat dabei entweder eine `script`-Sektion, die eine Sammlung an ausführbaren Schritten enthalten kann, oder eine `trigger`-Sektion, die andere Pipelines starten kann. Sie werden dabei auf oberster Ebene der Pipeline-Definition aufgelistet und über den `stage`-Parameter einer Stage zugeordnet. Um Entwicklern die Möglichkeit der einfachen Nutzung verschiedener Technologien zu bieten, baut GitLab auf Docker. Dabei können Docker-Images angegeben werden, in denen die vordefinierten Befehle ausgeführt werden. (GitLab, 2025b)

In Bezug auf GitHub sind GitLab Stages mit GitHub Jobs vergleichbar und GitLab Jobs mit GitHub Steps, wobei ein GitHub Step entweder ein Shell-Script oder eine GitHub Action ist. Eine GitLab-Component kann zwar deutlich mehr als nur ein Job sein, in dieser Arbeit werden jedoch die Begriffe GitLab-Component und GitHub-Action genutzt, wenn es um die erstellten Komponenten geht, die ein Nutzer später in seiner eigenen CI/CD-Pipeline nutzen kann.

2.3 API-Key als Authentifizierungsmechanismus

Zu der Absicherung einer Anfrage an einen Server gehören zwei zentrale Schritte: Authentifizierung und Autorisierung. Authentifizierung bezeichnet den Prozess der Überprüfung der Identität eines Benutzers oder Systems. Dabei wird mit Hilfe eines Identifikators der Kommunikationspartner eindeutig identifiziert. Dazu kann sowohl ein geheimer Schlüssel als auch eine Kombination von Benutzername und Passwort genutzt werden. Ziel ist es, sicherzustellen, dass Kommunikationspartner ohne gültigen Zugang nicht mit dem System kommunizieren dürfen. Eine

Autorisierung einer Anfrage erfolgt nach der erfolgreichen Authentifizierung. Bei der Autorisierung wird dabei geprüft, ob ein Nutzer oder System auf Ressourcen zugreifen oder Aktionen ausführen darf. (Metz, 1999)

Die Autorisierung der Nutzer findet in SCA Tool erst in den Controllern der API innerhalb des Backends statt. Die Authentifikation der Anfragen der Integrationen soll analog zur Filterung der WebApp-Kommunikation bereits vorher in Form eines Security-Filters stattfinden. Dafür muss ein passender Authentifizierungsmechanismus für die Kommunikation mit der Integration geschaffen werden.

Grundsätzlich gibt es einige verschiedene Authentifizierungsmechanismen, wie etwa JWT (JSON Web Tokens), OAuth 2.0, Cookie-basierte Authentifizierung oder API-Keys. Alle haben unterschiedliche Vor- und Nachteile. Es wurde eine Authentifizierung mittels API-Key gewählt. Da das Integrationsskript auf unterschiedlichen Maschinen ausgeführt werden kann, ist eine maschinenunabhängige Lösung notwendig. Außerdem wurden die Möglichkeiten auf String-basierte Methoden verringert, da zur Speicherung der Zugangsdaten verborgene private Variablen in GitHub beziehungsweise GitLab genutzt werden. Dadurch soll verhindert werden, dass diese direkt im Quellcode liegen, was vor allem für Open-Source-Projekte wichtig ist. (Gupta et al., 2024)

API-Keys sind eindeutige Identifikatoren, die typischerweise dazu verwendet werden, den Zugriff auf eine Programmierschnittstelle zu authentifizieren und zu autorisieren. Ihre genaue Verwendung kann dabei aber sehr unterschiedlich sein. Dabei können API-Keys zum Logging der Nutzung, zur Beschränkung der Nutzung von Schnittstellen, als passwortähnliche Zugangssteuerung oder als Teil einer aufwendigen kryptographischen Operation zur Authentifizierung verwendet werden. Ein API-Key kann sowohl als Header, Parameter oder Cookie einer Anfrage mitgegeben werden. (Farrell, 2009; Kornienko et al., 2021)

Die Sicherheit von API-Keys allein ist jedoch begrenzt. Gupta et al. (Gupta et al., 2024) empfehlen eine sichere Speicherung über Mechanismen wie Umgebungsvariablen. Gerade bei Open-Source-Projekten kann es sonst vorkommen, dass API-Keys durch eine Eintragung im Quellcode veröffentlicht werden. Auch das OWASP Cheat Sheet (OWASP, 2025) warnt im Bezug auf API-Keys, dass diese durch die Nutzung von Third-Party-Clients leicht kompromittiert werden können. OWASP empfiehlt deshalb, zusätzliche Parameter hinzuzuziehen, um Nutzerdaten zu schützen und auf eine sichere Übertragung zu achten.

2.4 Hashing Methoden

Hashing an sich bezeichnet grundlegend eine Funktion, die einen Input mit beliebiger Länge in einen String mit immer der gleichen Länge umwandelt. Dabei kann es verschiedene Anwendungsfälle und damit verbundene Anforderungen für

die Hashing-Funktionen geben. Die zwei Hauptkategorien sind einerseits das datenorientierte Hashing, Methoden zum schnellen Vergleichen und Sammeln von Daten, und andererseits sicherheitsorientiertes Hashing, Methoden zur Verifikation oder Validierung mittels eines Hashes. Die zweite Art wird auch in diesem Projekt benötigt, um die API-Keys sicher abzuspeichern und vergleichen zu können. (Chi & Zhu, 2017)

Um gespeicherte Passwörter sicher zu speichern, wird in SCA Tool aktuell bcrypt zur Erstellung der Hashes genutzt. Hashing-Funktionen wie bcrypt, Argon2 oder PBKDF2 zeichnen sich dadurch aus, dass durch viele Iterationen von Hashing das Erstellen eines Hashes ressourcenaufwendig ist und dadurch einige Zeit kostet. Dadurch können etwa Brute-Force-Attacken verhindert werden. Außerdem nutzt sie in ihren Standard-Konfigurationen sogenannte `Salts`. Diese sind zufällige Strings, die jedem erstellten Passwort angehängt werden. Das bedeutet, dass zwei Nutzer mit demselben Passwort unterschiedliche Hashes erhalten. So kann bei veröffentlichten Datenbanken durch das Entschlüsseln eines Passworts nicht ein anderes auch entschlüsselt werden. Jedoch führt dieser Vorteil dazu, dass zum Abgleichen einer Eingabe das Passwort nicht einfach erneut gehasht und verglichen werden kann, da für jeden Hash ein unterschiedlicher Salt genutzt wird. Sowohl bcrypt als auch Argon2 bieten daher eine Funktion an, die überprüft,ob ein neuer String mit dem ursprünglichen übereinstimmt. Das bedeutet, dass damit erstellte Hashes nicht genutzt werden können, um nach dem spezifischen Hash-Wert in der Datenbank zu suchen. Dafür müsste jeder einzelne Eintrag abgerufen und mittels der Funktion verglichen werden. (Singh & Raj, 2022; Sriramya & Karthika, 2015)

Daher muss eine andere Methodik genutzt werden, die reproduzierbare Hashes erzeugt. Im Rahmen dieser Arbeit ist die Entscheidung auf den HMAC-SHA256 gefallen. Das liegt daran, dass unter anderem der populäre Hashing-Algorithmus PBKDF2 als Grundlage den HMAC-SHA256 nutzt. (Singh & Raj, 2022, Kapitel 7) SHA256 steht dabei für Secure Hashing Algorithm (SHA) mit einer resultierenden Hashlänge von 256 Bit. Es ist eine kryptographische Einweg-Hash-Funktion, aus der der Input nicht zurückgerechnet werden kann. Zur zusätzlichen Sicherheit wird der Hash-based Message Authentication Code (HMAC) genutzt. Dabei wird der zu hashende Teil mit einem Secret Key kombiniert. Dadurch kann überprüft werden, ob der Hash von einem System erstellt wurde, das den Secret Key kennt. (Manisha Rana et al., 2024)

3 Stand der Technik

Im folgenden Kapitel werden Systeme vorgestellt, die entweder ähnliche Funktionalität wie SCA Tool oder eine ähnliche Art von Integration bieten. Besonderer Fokus liegt dabei darauf, wie diese eine Integration mit GitHub und GitLab implementieren. Dafür wurden die Integrationen des OSS Review Toolkit und AppSweep der Firma Guardsquare analysiert.

3.1 OSS Review Toolkit

Das OSS Review Toolkit (ORT) ist eine Sammlung von Tools, die zur Automatisierung von Software-Compliance-Prüfungen entwickelt wurden. Es ist ein Projekt der Linux Foundation und bietet eine umfassende Lösung für die Software Composition Analysis (SCA), Lizenz-Compliance, Schwachstellenmanagement und mehr, um die Risiken in der Softwareentwicklung zu managen und zu analysieren. Es kann sowohl als Bibliothek zur Nutzung im eigenen Code mittels eines Commandline-Aufrufs für die skriptgesteuerte Verwendung als auch über seine CI-Integrationen eingesetzt werden. Neben einer GitHub Action wird auch eine GitLab-Pipeline zur Verfügung gestellt. (OSS Review Toolkit, 2025)

Die GitHub Action[1] ist eine Composite Action, das bedeutet eine Action, zusammengesetzt aus anderen Actions oder Commandline-Befehlen. Die komplette Logik der Action liegt daher in der action.yaml Datei. Nach der Definition der möglichen Inputs und Outputs erfolgt die eigentliche Beschreibung der Action. Neben einer Analyse der verschiedenen Inputs, dem Sammeln von Konfigurationsdateien sowie dem Neuordnen von Environment-Variablen führt sie je nach Konfiguration die unterschiedlichen Komponenten von ORT aus. Hervorzuheben ist dabei, dass zur Ausführung das ORT Docker Image geladen und in den einzelnen Schritten mit den verschiedenen Parametern der einzelnen Services aufgerufen wird. Optional werden die Ergebnisse der verschiedenen Aufrufe als Artefakte der Action hochgeladen. Dadurch kann der Nutzer sie im Anschluss herunterladen und analysieren, ohne diese Dateien tatsächlich im Repository liegen zu

[1] https://github.com/oss-review-toolkit/ort-ci-github-action

haben. Zusätzlich kann die Action final noch konfiguriert werden, den Lauf als fehlgeschlagen zu deklarieren, wenn festgelegte Thresholds bei den Ergebnissen überschritten werden. Dadurch gilt der Lauf als gescheitert und der Nutzer wird darüber informiert, dass Änderungen notwendig sind.

Die GitLab Pipeline[2] nutzt ebenfalls das erwähnte Docker-Image. Eine Besonderheit ist dabei, dass die Pipeline von Anfang an das ORT Docker-Image nutzt und alles innerhalb dieses Images ausgeführt wird. Die Action hingegen ruft für jeden Service einzeln 'docker run' auf, wodurch für jeden Aufruf ein Docker-Container gestartet wird. Außerdem werden für GitLab Schritte wie das Caching selbst implementiert, da die in der GitHub Action genutzten Funktionen auf GitLab nicht zur Verfügung stehen. Ein weiterer Unterschied ist, dass die entstandenen Artefakte nicht automatisch hochgeladen werden, sondern der Nutzer diese in der Job-Definition festlegen muss. Nur markierte Artefakte sind nach dem Ende eines Durchlaufs verfügbar.

Obwohl SCA Tool intern ebenfalls ORT verwendet, wurde bewusst auf die Nutzung der bestehenden ORT-Integration verzichtet, um eine technologische Unabhängigkeit sicherzustellen. Durch den Upload der Dateien und die serverseitige Analyse bietet SCA Tool eine eigenständige und umfangreichere Möglichkeit zur Untersuchung von Softwareabhängigkeiten. Zudem können Analysen auf diese Weise zu einem beliebigen Zeitpunkt reproduzierbar erneut durchgeführt werden.

3.2 AppSweep

Während die SCA Tool Integration aktuell für npm-Packages ausgelegt ist, bietet Guardsquare mit AppSweep eine ähnliche Lösung für mobile Applikationen. In einem ersten Schritt führt das Tool eine statische Analyse auf dem hochgeladenen Code der iOS- oder Android-Applikation durch. Dabei wird eine Übersicht über alle genutzten Abhängigkeiten erstellt und zu jeder Dependency bekannte Schwachstellen und ihr Schweregrad angegeben. Zusätzlich zu der vergleichbaren Funktionalität von SCA Tool bietet AppSweep noch einen Code-Scan, um eventuell gefährliche Funktionsaufrufe und Konfigurationen zu finden und bewertet diese unter anderem anhand der OWASP-Einschätzungen. (Guadrsquare, 2025a)

Um den Anwendungscode für den Scan bereitzustellen, bietet AppSweep einige Möglichkeiten. Einerseits kann die gebündelte Applikation auf der Webseite hochgeladen werden. Auf der anderen Seite stehen dem Nutzer verschiedene Möglichkeiten der Integration mit Programmen und Plattformen zur Verfügung. Dazu gehören unter anderem Programme wie Xcode, Command-Line-Tools wie Fastlane oder Plattformen wie Jenkins, Azure oder GitHub.

[2] https://github.com/oss-review-toolkit/ort-ci-gitlab

Bei GitHub existiert dabei sowohl die Möglichkeit der Nutzung einer GitHub Action innerhalb eines Workflows, um den Code an AppSweep zu übermitteln, als auch eine GitHub-App. Diese ermöglicht es unter anderem, im Anschluss an die Analyse, die Ergebnisse des Scans direkt auf GitHub zur Verfügung zu stellen. Diese benötigt jedoch dementsprechend einige umfangreiche Berechtigungen, um eigenständig Inhalte auf GitHub hinzuzufügen. (Guadrsquare, 2025b)

Eine eigene GitLab-Component hat AppsSweep nicht. Jedoch kann in GitLab auch das Command-Line-Tool genutzt werden und somit eine eigene Integration vom Nutzer erstellt werden.

Die AppSweep GitHub Action[3] ist eine Composite Action. Dabei wird zunächst ein Installations-Shellscript[4] heruntergeladen, das dann das eigentliche AppSweep CLI Tool herunterlädt. Nach erfolgreichem Installieren wird dann der Code mit Hilfe dieses CLI Tools hochgeladen. Bei einer Analyse des Installationsskripts fällt auf, dass die unterstützten Plattformen auf Linux und MacOS beschränkt sind. Diese Einschränkung ist jedoch nicht in der Beschreibung der Action oder dem Eintrag des GitHub Marketplace zu entnehmen. Wie in Kapitel 2.2.1 beschrieben, ist eine Composite Action grundsätzlich auf allen Plattformen ausführbar. Dadurch können Nutzer ohne Ausprobieren oder genauere Analyse des Skriptes nicht erkennen, dass eine Ausführung unter Umständen zu Problemen führen könnte.

[3]https://github.com/Guardsquare/appsweep-action
[4]https://platform.guardsquare.com/cli/install.sh

4 Requirements

Dieses Kapitel listet die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde System auf. In Kapitel 8 wird analysiert, ob und inwieweit diese Anforderungen erfüllt wurden. Die Anforderungen sind thematisch gruppiert und nach ihrer Priorität sortiert. Auch wird durch die Nutzung von „muss“ und „soll“ die Notwendigkeit der Anforderung spezifiziert.

4.1 Funktionale Anforderungen

Funktionale Anforderungen beschreiben, welche Funktionen das System und seine Komponenten erfüllen müssen, um die gewünschten Aufgaben zu unterstützen.

Allgemein

F-1 Das System muss eine REST-API zur Kommunikation mit einer Integration bereitstellen.

F-2 Ein Externes System muss über eine API einen Scan erstellen und starten können, um eine Automatisierung über externe Tools zu ermöglichen.

F-3 Der Nutzer muss ein Projekt so konfigurieren können, dass dieses mit einer Integration verbunden ist und die benötigten Daten zur Nutzung gespeichert werden können.

F-4 Das System muss eine URL zu den Ergebnissen bereitstellen, auf der die Ergebnisse eines über die API gestarteten Scans eingesehen werden können.

F-5 Das System muss den zeitlichen Verlauf von gescannten Versionen darstellen können, um die Scanreihenfolge von hochgeladenen Versionen nachzuvollziehen.

F-6 Das System muss Versionen zu Referenzen wie etwa Branches oder Tags zuordnen, damit eine effiziente Suche nach einem bestimmten Scan-Ergebnis möglich ist.

F-7 Der Nutzer muss zwischen hochgeladenen Versionen eines Projektes auswählen können, um die verschiedenen Versionen zu vergleichen.

F-8 Das System muss bei Fehlern abbrechen und dem Nutzer nach Möglichkeit konkrete Hinweise zur Fehlerbehebung geben.

F-9 Der Nutzer muss über die Web-Oberfläche eine Anleitung zur Einrichtung und Verwendung der Integration erhalten, um eine korrekte Konfiguration und somit korrekte Nutzung zu garantieren.

F-10 Die Benutzeroberfläche muss die benötigte Integrations-Konfiguration bereitstellen, damit der Nutzer diese lediglich in sein Repository einfügen muss.

F-11 Der Nutzer soll direkte Links zu externen Plattformen zur Vervollständigung der Konfiguration erhalten.

Integration

F-12 Die Integration muss mit der SCA Tool API interagieren, um Dateien hochzuladen und einen Scan zu starten.

F-13 Die Integration muss mit GitHub Workflows und GitLab CI/CD kompatibel sein.

F-14 Die Integration muss mit den unterschiedlichen SCA Tool Systemen (Testing/Production) nutzbar sein.

F-15 Die Integration muss bestimmte Pfade und Dateien, die der Nutzer angibt, von der Suche ausschließen, um den Nutzer entscheiden zu lassen, welche Dateien hochgeladen werden dürfen und welche privat bleiben sollen.

F-16 Die Integration muss verschiedene Dateitypen finden, die durch den Nutzer angegeben werden können.

F-17 Die Integration muss bei Fehlern abbrechen und dem Nutzer nach Möglichkeit konkrete Hinweise zur Fehlerbehebung geben.

F-18 Die Integration muss neben Dateien auch Metadaten, wie etwa Repository-Name, Referenz oder Commit-Hash übermitteln, damit das System die notwendigen Daten hat, um den Upload zu verarbeiten.

F-19 Die Integration muss die Antwort des Systems dem Nutzer zugänglich machen.

F-20 Die Integration muss sicherstellen, dass alle erforderlichen Dateien vor dem Upload vorhanden sind.

F-21 Die Integration muss Scans abbrechen, falls Dateien eine maximale Dateigröße von 100MB überschreiten, um den Server vor fehlerhaften und zu großen Dateien zu schützen.

F-22 Die Integration muss als Github-Action im GitHub-Marketplace und als GitLab-Component im Gitlab-CI/CD-Catalog zur Nutzung durch Nutzer bereitgestellt werden.

F-23 Die Integration soll in der Lage sein, neben der Ausgabe der Ergebnisse im Log des Ablaufs, den Link als Kommentar in einem Pull-Request bzw. Merge-Request zu posten.

API-Key

F-24 Das System muss eine Authentifizierung mittels API-Key ermöglichen, damit eine Integration die Möglichkeit hat, sich mittels eines Schlüssels am System zu authentifizieren.

F-25 Das System muss einen sicheren, nicht reproduzierbaren API-Key generieren, damit dieser nicht von Dritten rekonstruiert werden kann.

F-26 Ein API-Key muss ein Ablaufdatum besitzen, um einen regelmäßigen Wechsel zur Erhöhung der Sicherheit zu fördern.

F-27 Das System muss einen API-Key eindeutig einer Organisation zuordnen, damit gesendete Daten entsprechend gespeichert werden können.

F-28 Das System muss API-Keys auf Gültigkeit (Format, Ablaufdatum, Zugehörigkeit) prüfen, um Manipulation zu verhindern.

F-29 Das System muss die Möglichkeit bieten, API-Keys zu invalidieren, um sowohl versehentlich veröffentlichte als auch ungenutzte Keys zu sichern.

F-30 Der Nutzer muss die Möglichkeit haben, alle mit ihm verbundenen API-Keys zu verwalten, um neue Keys zu erzeugen und bestehende zu prüfen.

F-31 Die Benutzeroberfläche muss den Nutzer zur sicheren Speicherung des API-Keys auf GitHub/GitLab anleiten, um versehentliche Veröffentlichungen zu verhindern.

4.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben die Qualitätsmerkmale und Randbedingungen des Systems.

Allgemein

NF-1 Die API-Definition muss OpenAPI 3.0.3 konform sein.

NF-2 Die Integration muss mit den gängigen Betriebsystemen kompatibel sein. Also MacOS, Windows und Linux.

NF-3 Die Integration muss über eine Dokumentation verfügen, die ein Beispiel einer Konfiguration für Nutzer und Hinweise an weitere Entwickler beinhaltet.

NF-4 Die Integration und die damit verbundenen Endpunkte sollen versioniert sein, um Nutzern die Umstellung auf eine neue Version in einem angemessenen Zeitraum zu ermöglichen.

Benutzerfreundlichkeit

NF-5 Die Benutzeroberfläche muss eine Hierarchie von Projekt zu Referenz, zu Version darstellen.

NF-6 Die Benutzeroberfläche muss bei gängigen Problemen während der Konfiguration, Hilfestellung und mögliche Lösungen anzeigen.

NF-7 Die Benutzeroberfläche muss eine Navigation für Versionen bieten, die es dem Nutzer erlaubt innerhalb von 2 Klicks zu einer vorherigen Version zu gelangen.

NF-8 Die Konfigurationsseite soll für Nutzer mit unterschiedlichen Wissensständen die benötigten Informationen übersichtlich zur Verfügung stellen.

Sicherheit

NF-9 Das System muss sicherstellen, dass API-Keys nur zur Authentifizierung für die Kommunikation mit einer Integration zugelassen sind.

NF-10 Der API-Key muss über eine sichere HTTPS Verbindung übertragen werden.

NF-11 Der API-Key muss verschlüsselt in der Datenbank gespeichert werden.

NF-12 Die Integration soll dem Prinzip der Datenminimierung folgen, indem es nur notwendige Daten vom Nutzer verlangt.

Codequalität

NF-13 Der Code muss definierten Formatierungsrichtlinien entsprechen.

NF-14 Die Integration muss modular aufgebaut und erweiterbar sein, damit spezifische Funktionalität leicht geändert und hinzugefügt werden kann.

NF-15 Der Schnittstellen Code und Definitionen von Web- und Integrations-API sollen dem Single-Responsibility-Prinzip folgend voneinander getrennt sein.

5 Architektur

In diesem Kapitel wird die Architektur der Integration im Zusammenhang mit SCA Tool erläutert. Zuerst wird das System und die API-Architektur betrachtet, bevor im Anschluss auf die involvierten Entitäten und ihre Verbindung eingegangen wird.

5.1 SCA Tool System und API-Architektur

SCA Tool besteht im Wesentlichen aus drei Hauptkomponenten: dem Backend, dem Web-Client und der neu entwickelten CI-Integration.

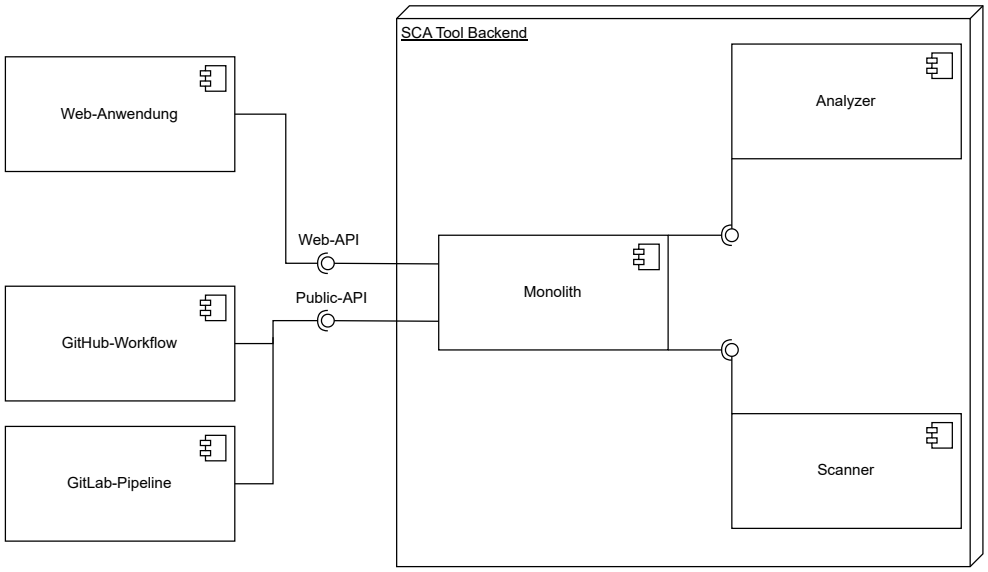


Abbildung 5.1: Vereinfachtes Komponentenmodell

Abbildung 5.1 veranschaulicht dieses System und ihre Interaktion in vereinfachter Form. Das Backend selbst besteht aus drei zentralen Komponenten. Der Analyzer extrahiert aus den hochgeladenen Dateien die genutzten Abhängigkeiten. Im An-

schluss analysiert der Scanner die Lizenzen und bekannten Schwachstellen der gefundenen Abhängigkeiten. Die zentrale Rolle innerhalb des Backends übernimmt der Monolith, der mit allen Komponenten des Systems kommuniziert und diese verbindet. Er plant und startet Analyseprozesse, verarbeitet Anfragen sowohl von Web-Clients als auch von CI-Integrationen und kommuniziert mit der Datenbank sowie weiteren Server-Komponenten, die zur Vereinfachung nicht in dieser Abbildung abgebildet wurden. Der Monolith ist in Java unter der Verwendung des Spring-Boot-Frameworks implementiert.

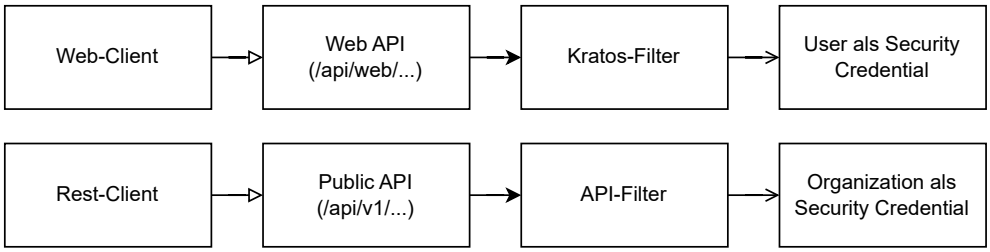


Abbildung 5.2: Übersicht Client zu Authorisierungsinformation

Für die Kommunikation mit dem Backend gibt es zwei separate Schnittstellen. In Abbildung 5.2 ist dargestellt, welche Clients mit welchen APIs kommunizieren, welche Filter zur Authentifizierung der Anfragen verwendet werden und welche Informationen zur weiteren Autorisierung in den Controllern gesetzt werden. Die erste Schnittstelle ist die Web-API. Über diese kommuniziert die in Angular mit TypeScript geschriebene Web-Anwendung mit dem Backend. Die zugehörigen Endpunkte der Webanwendung sind dabei mit dem Prefix `/api/web/` versehen. Anfragen an diese Endpunkte werden durch den sogenannten Kratos-Filter geprüft. Dieser validiert eingehende Anfragen anhand eines Cookies mit einer Kratos-Session. Bei erfolgreicher Authentifizierung wird der Benutzer als sogenanntes Security-Credential zur weiteren Autorisierung gesetzt.

Die zweite Schnittstelle ist die Public-API, die die Kommunikation mit den Integrationen ermöglicht. In Abbildung 5.1 ist die Integration als zwei Komponenten dargestellt - einem GitHub-Workflow und einer GitLab-Pipeline - da beide mit dem Backend über das Integrationsskript interagieren. Dabei unterscheiden sich die Implementierungen lediglich in einigen wenigen plattformspezifischen Unterschieden. Die Hauptlogik ist jedoch bei beiden derselbe zugrunde liegende TypeScript-Code. Anfragen zum Upload einer Version laufen über die Public-API, deren Endpunkte mit dem Prefix `/api/v1` versehen sind. Sie werden durch den sogenannten API-Filter authentifiziert. Dabei wird der mitgesendete API-Key auf Gültigkeit geprüft. Für die anschließende Autorisierung wird die zugehörige Organisation als Security-Credential gesetzt. Dadurch kann eindeutig festgestellt werden, für welche Projekte ein API-Key berechtigt ist, Daten hochzuladen.

Die Trennung der Schnittstellen zwischen Web-Client und Backend sowie Integration und Backend verfolgt zwei zentrale Ziele. Erstens unterstützt sie die Einhaltung des Single-Responsibility-Prinzips der APIs, da jede API für einen klar abgegrenzten Funktionsbereich zuständig ist. Das erleichtert das Testen, Verstehen und Warten der einzelnen Schnittstellen und verhindert unbeabsichtigte Auswirkungen auf andere Funktionalität bei Änderungen. (Ramachandrappa, 2024) Zweitens vereinfacht die Trennung die Versionierung der API der Integration: Unterschiedliche API-Präfixe, die auf Hauptebene der Spezifikation festgelegt werden, ermöglichen es, neue Versionen ohne großen Aufwand einzuführen. Auch kann durch parallel veröffentlichte Versionen die Abwärtskompatibilität mit alten Konfigurationen der Integration gewahrt bleiben. (Harry Peter, 2025) Zudem lassen sich die zugehörigen Controller in getrennte Packages auslagern, was die Wartbarkeit des Codes weiter verbessert.

5.2 Entitäten

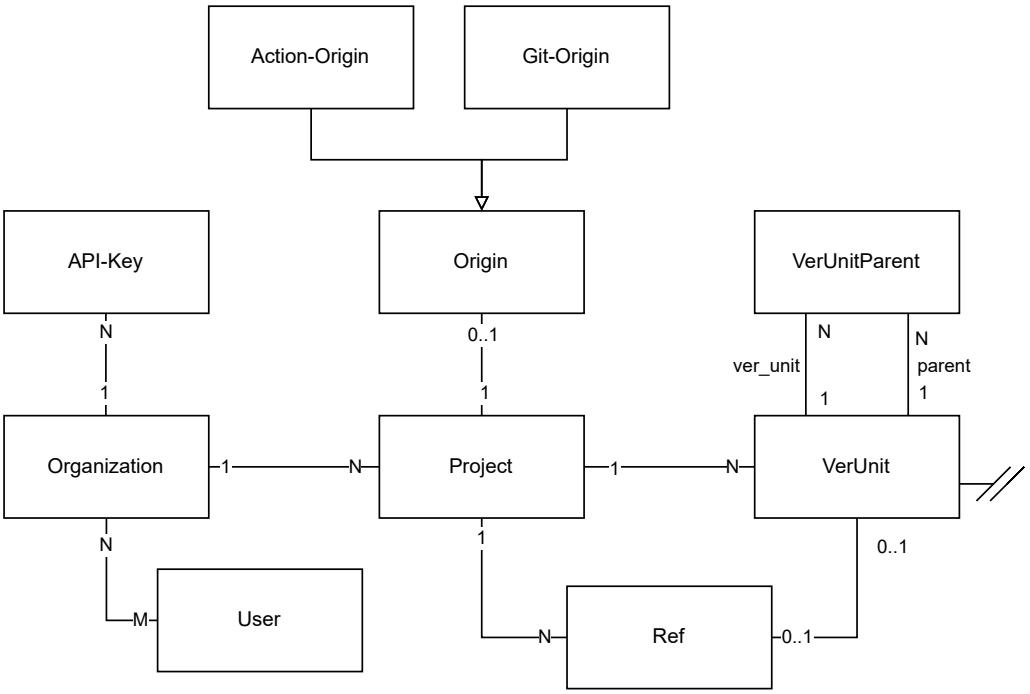


Abbildung 5.3: Ausschnitt aus einem vereinfachten Entitätenmodell des SCA Tool

In Abbildung 5.3 ist ein Ausschnitt aus einem vereinfachten Entitätenmodell dargestellt, das den für diese Arbeit relevanten Teil zeigt. Dargestellt sind die Beziehungen zwischen Projekten, Versionen, Referenzobjekten, Organisationen

und API-Keys. Dieses Modell bildet die Grundlage für die Integration und die dazugehörige Verwaltung der hochgeladenen Versionen. Ein Auszug aus dem entsprechenden Datenbankschema ist in Anhang B zu finden.

Die zentrale Entität dieser Arbeit ist die `VerUnit`. Sie repräsentiert eine spezifische, hochgeladene Version eines Repositories, die entweder manuell über die Weboberfläche oder automatisiert über die neu entwickelte Integration hochgeladen wurde. Mit der `VerUnit` sind sämtliche Analyseergebnisse des SCA Tool verknüpft. Dieser Teil ist in der Abbildung jedoch nicht dargestellt, da er für die vorliegende Arbeit nicht relevant ist.

Jede `VerUnit` ist genau einem `Project` zugeordnet. Ein Projekt ist mit maximal einer `Origin` verbunden, die die benötigte Konfiguration bestimmt. Der `Origin`-Type gibt an, aus welcher Quelle die zu analysierenden Daten stammen. Diese Unterscheidung ist für die interne Verarbeitung wichtig, da je nach Quelle die Daten unter Umständen anders verarbeitet oder angezeigt werden sollen. In Abbildung 5.3 sind die Typen `Git-Origin` und `Action-Origin` dargestellt.

Ein Projekt hat eine `Origin` des Typs `Git-Origin`, wenn ein GitHub-Repository durch das SCA Tool geklont und anschließend ein Scan gestartet werden soll. Dafür muss der Nutzer unter anderem die URL des Repositories sowie – im Falle eines privaten Repositories – die benötigten Zugangsdaten angeben, damit das SCA Tool eine lokale Kopie erzeugen und analysieren kann.

Die `Action-Origin` stellt die neu hinzugefügte Möglichkeit dar, mittels einer CI-Integration für GitHub und GitLab automatisiert die erforderlichen Daten an das SCA Tool zu übermitteln. Im Gegensatz zur `Git-Origin` wird dabei lediglich die Repository-URL benötigt.

Je nach Konfiguration der Integration können `VerUnits` für unterschiedliche Branches, Tags oder Pull Requests erzeugt werden. So kann etwa ein GitHub-Workflow so konfiguriert sein, dass bei jedem neuen Commit auf dem main-Branch die Action ausgelöst wird. Die Information über die interne Referenz auf den externen Plattformen wird in SCA Tool durch sogenannte `Refs` abgebildet.

In einer `Ref` wird immer nur die aktuellste Version verlinkt. Um dennoch alle Versionen, die zu einer `Ref` gehören, finden zu können, wird eine Vorgängerreferenz in einem Eintrag in der `VerUnit-Parent` Tabelle gespeichert, wenn eine Version für eine bestehende Ref hochgeladen wird. Durch dieses Konstrukt können `VerUnits` mit der gleichen `Ref` gruppiert werden, um in der Benutzeroberfläche zwischen Versionen einer Ref wechseln zu können.

Der letzte dargestellte Teil der Abbildung 5.3 betrifft die Zuordnung von Projekten zu Organisationen und Nutzern. Jedes Projekt ist genau einer `Organization` zugeordnet. Einer Organisation können mehrere `User` angehören und ein `User` kann Mitglied mehrerer Organisationen sein. `API-Keys`, die zur Authentifizierung

der Integrationen genutzt werden, sind stets genau einer Organisation zugeordnet - nicht jedoch einem einzelnen Nutzer. Dadurch bleibt eine Integration auch dann funktionsfähig, wenn ein Nutzer die Organisation verlässt.

6 Design

In diesem Kapitel wird das Design der einzelnen Komponenten beschrieben, die zur Einbindung der Integrationen erforderlich sind. Zunächst wird das in TypeScript implementierte Integrationsscript und dessen Ablauf im Rahmen einer GitHub Action vorgestellt. Anschließend wird der Upload-Prozess auf Serverseite beschrieben. Zum Abschluss des Kapitels wird noch kurz auf den Prozess der Erstellung eines API-Keys sowie der Konfiguration eines Projektes im SCA Tool eingegangen.

6.1 Integrationsscript

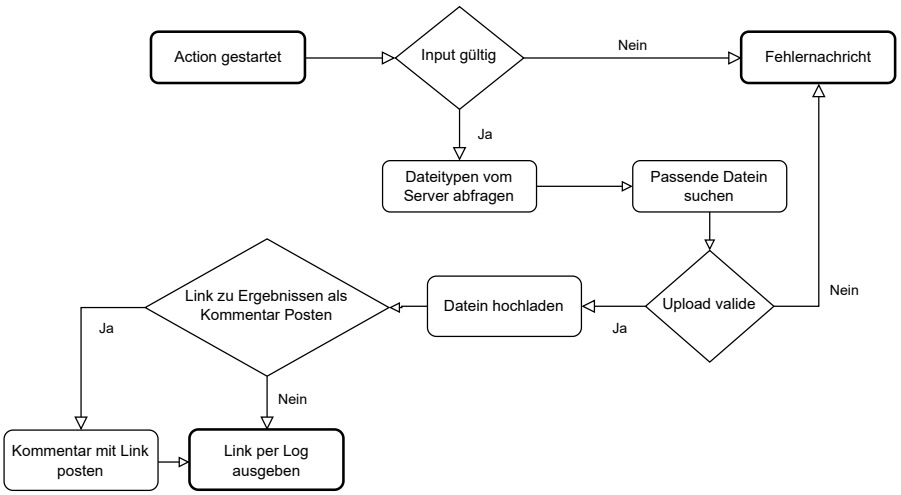


Abbildung 6.1: Ablauf der GitHub-Action

Abbildung 6.1 zeigt den Ablauf des in einer GitHub Action ausgeführten Integrationsscripts in Form eines vereinfachten Ablaufdiagramms. Wird die Action als Teil eines Workflows – etwa durch einen Push in einem Pull-Request – ausgelöst, so beginnt der Prozess mit der Prüfung der Eingabeparameter. Diese beinhalten

unter anderem den API-Key, die Projekt-ID sowie optional weitere Konfigurationswerte. Dabei werden die Eingaben auch auf Gültigkeit geprüft. Dazu zählt die Überprüfung, ob der API-Key abgelaufen ist, oder ob die eingegebene Liste an ausgeschlossenen Dateien zu einer gültigen Liste umgewandelt werden kann. Treten an dieser Stelle Fehler auf, wird eine entsprechende Fehlermeldung ausgegeben und das Script abgebrochen.

Im nächsten Schritt fragt das Script die unterstützten Dateikombinationen vom Backend ab. Auf diese Weise kann der Server steuern, welche Technologien erkannt und verarbeitet werden können, ohne dass die GitHub-Action selbst angepasst werden muss. Dies erlaubt eine flexible Erweiterbarkeit der unterstützten Technologien auf Serverseite.

Nach erfolgreicher Antwort sucht das Script im Repository nach den vom Backend spezifizierten Dateitypen. Dabei werden durch den Nutzer ausgeschlossene Dateien, Ordner oder Pfade ignoriert. Vor dem Upload werden die gefundenen Dateien überprüft. Dafür wird sichergestellt, dass mindestens eine Datei gefunden wurde. Auch wird kontrolliert, dass mindestens eine vom Backend angegebene Kombination der benötigten Dateien vorhanden ist. Beispielsweise benötigt SCA Tool im Falle einer `package.json` Datei zwingend auch eine `package-lock.json`, um eine Analyse erfolgreich durchführen zu können. Diese Anforderungen sind ebenfalls in der vom Backend erhaltenen Antwort enthalten. Zum Schutz des Servers vor großen Dateien wird zudem sichergestellt, dass keine Datei größer als 100 MB ist.

Sind alle Voraussetzungen erfüllt, wird ein Upload-Request an das Backend gesendet. Dieser enthält neben den eigentlichen Dateien und deren Pfaden innerhalb des Repositories auch den API-Key, den Repository-Namen und die Projekt-ID zur Identifikation und eindeutigen Zuordnung. Zusätzlich werden der Commit-Hash sowie die Ref zur Strukturierung der Versionen innerhalb von SCA Tool mitgeschickt. Wie in Kapitel 5.2 bereits erläutert, ist die Ref der Branch, Tag oder Pull-Request, in dessen Kontext die Action ausgelöst wurde.

Abbildung 6.2 stellt die Interaktion zwischen GitHub Action, dem GitHub-System und SCA Tool als Sequenzdiagramm dar. Nach dem Start des Workflows und der darin enthaltenen Action sendet das Integrationsscript eine nicht authentifizierte Anfrage zur Abfrage der Dateitypen an das Backend. Anschließend erfolgt - nach der beschriebenen internen Verarbeitung - der Upload der relevanten Daten auf den Server, dessen Verarbeitung in der folgenden Sektion erläutert wird. Als Antwort erhält die Action unter anderem den Link zur neu erstellten VerUnit. Über diesen kann der Nutzer in der Webanwendung die Analyseergebnisse einsehen und mit vorherigen Versionen vergleichen.

Bei der Nutzung der erstellten GitHub-Action kann optional noch ein Kommentar in GitHub in einem Pull-Request gepostet werden, der den Link zur entsprechen-

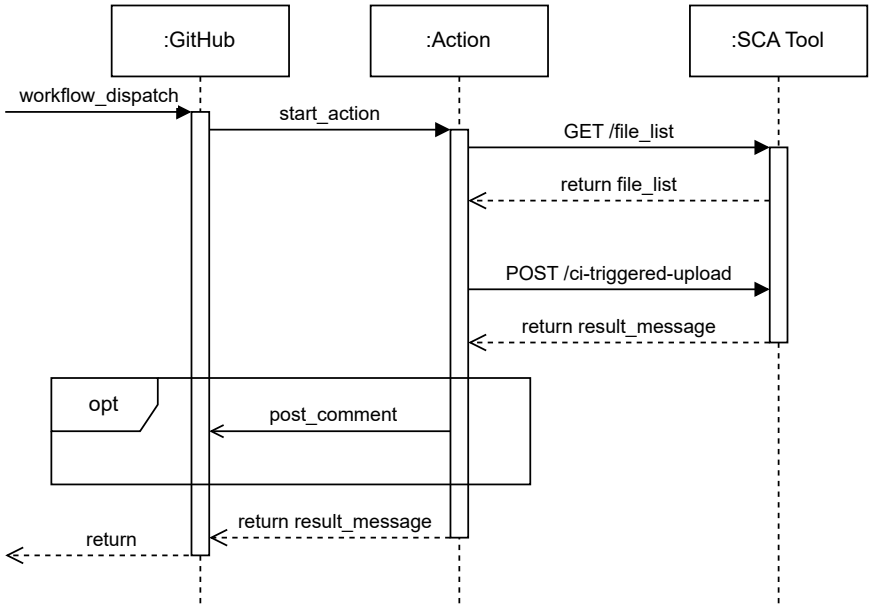


Abbildung 6.2: Sequenz eines GitHub-Workflows

den VerUnit enthält. Dies ist sowohl in Abbildung 6.1 als auch in 6.2 dargestellt. Ob ein Kommentar gepostet wird, hängt davon ab, ob ein GitHub-Token als Parameter in der Konfiguration der Action gesetzt wurde und ob der aktuelle Kontext ein Pull-Request ist. In diesem Fall wird die GitHub API verwendet, um eine Nachricht mit dem Link und den hochgeladenen Dateien zu posten. So kann der Link direkt im Pull-Request eingesehen werden, ohne dass der Nutzer den Ausführungslog der Action aufrufen muss. Tritt während des Prozesses ein Fehler auf, wird kein Kommentar erstellt. Unabhängig von der Kommentarfunktion wird sowohl bei der GitHub Action als auch dem GitLab Component alle Informationen im Log ausgegeben beziehungsweise bei einem Fehler die Ausführung als fehlgeschlagen markiert und dementsprechend auf der Plattform angezeigt.

6.2 Ablauf auf Serverseite

Die erste Kommunikation mit dem Server erfolgt über eine GET-Anfrage zur Abfrage der unterstützten Kombinationen von Dateitypen. Hierbei handelt es sich um eine einfache, nicht authentifizierte Anfrage, bei der eine Liste von Dateitypenkombinationen zurückgesendet wird. Da bei dieser Anfrage keine sensiblen oder privaten Daten übertragen werden, ist kein API-Key erforderlich und es erfolgt keine Filterung durch die Security-Konfiguration.

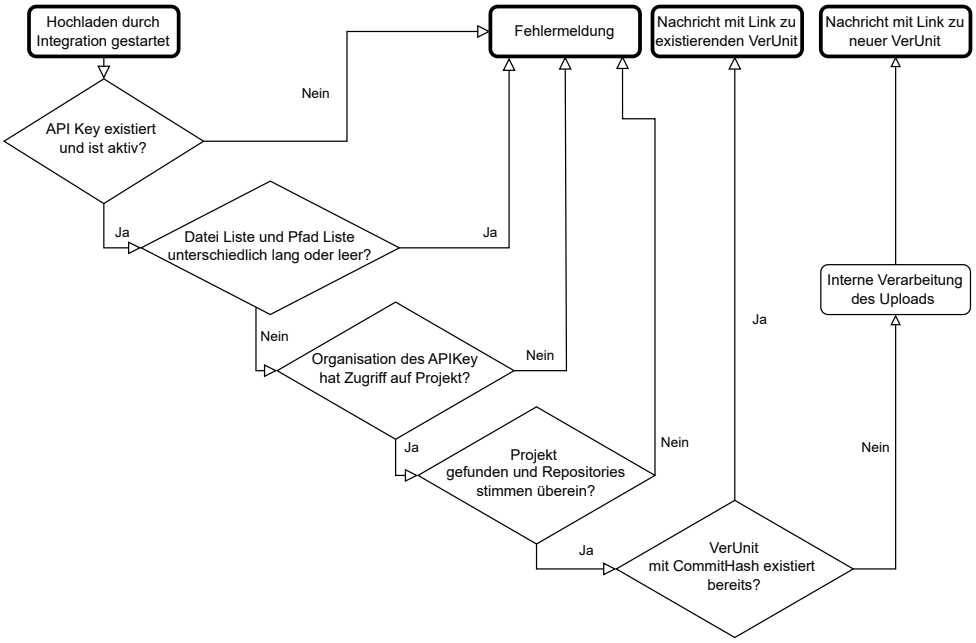


Abbildung 6.3: Ablauf der Integration innerhalb des Backends

In Abbildung 6.3 ist die zweite und zentrale Interaktion dargestellt. Dabei wird mit Hilfe eines POST-Requests die Dateien zum Scannen an den Server geschickt. Zunächst wird die Anfrage durch den API-Filter geprüft. Dabei wird sichergestellt, dass der übermittelte API-Key noch gültig ist und ein entsprechender Eintrag in der Datenbank existiert. Zudem wird verifiziert, dass das mitgesendete Ablaufdatum mit dem in der Datenbank gespeicherten Wert übereinstimmt, um Manipulation zu verhindern.

Die weitere Verarbeitung erfolgt im zuständigen Controller der Schnittstelle und den verbundenen Services. Zuerst wird geprüft, ob mindestens eine Datei übertragen und ob für jede Datei ein Pfad mitgesendet wurde. Anschließend wird validiert, ob die mit dem API-Key verknüpfte Organisation Zugriff auf das angegebene Projekt hat. Auch die Repository-URL wird mit dem Projekt in SCA Tool abgeglichen. Schlägt eine dieser Prüfungen fehl, wird eine Fehlermeldung zurückgegeben und der Vorgang abgebrochen. Die Validierungen sollen sicherstellen, dass nur berechtigte Nutzer Integrationen für ein bestimmtes Projekt konfigurieren und Daten hochladen können.

Bevor eine neue `VerUnit` erstellt wird, prüft das System, ob für die Kombination aus Projekt und Commit-Hash bereits eine Version existiert. Ist dies der Fall, wird keine neue Analyse durchgeführt, sondern der Link zur bestehenden Version zurückgegeben. Andernfalls wird eine neue Version angelegt und der Upload verarbeitet. Diese Verarbeitung ist in Abbildung 6.4 dargestellt.

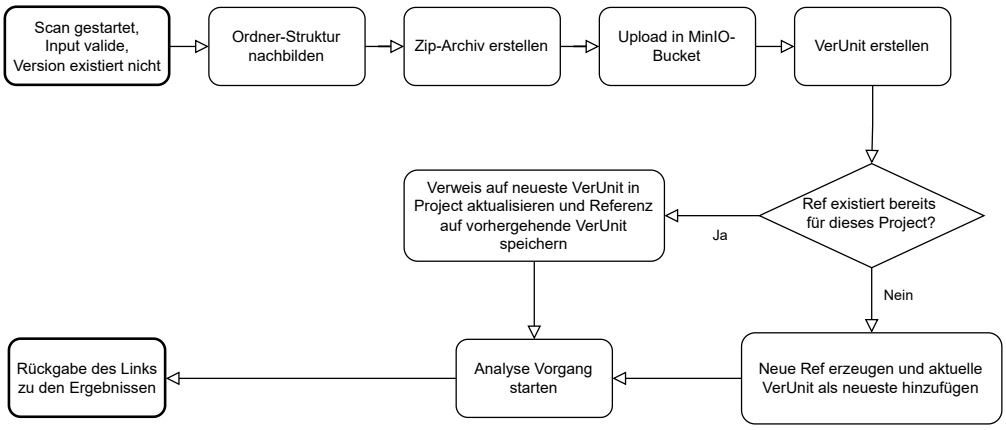


Abbildung 6.4: Ablauf der internen Verarbeitung des Uploads

Bei der weiteren Verarbeitung wird zunächst die Ordnerstruktur des Repositories rekonstruiert, sodass alle Dateien in ihrem ursprünglichen Ordner erscheinen. Dies ist erforderlich, da manche Analyseprozesse funktionierende relative Pfade benötigen. Die strukturierte Dateiablage wird anschließend in ein ZIP-Archiv gepackt und in den MinIO-Bucket des Servers hochgeladen, von wo aus der Scanner darauf zugreifen kann.

Nachdem die `VerUnit` in der Datenbank angelegt wurde, erfolgt anschließend die Zuordnung zu einer `Ref`. Falls die übermittelte `Ref` für das Projekt bereits existiert, wird diese aktualisiert, indem die neue `VerUnit` als neueste Version eingetragen wird. Zudem wird ein Eintrag in der VerUnit-Parent Tabelle erstellt, sodass ein chronologischer Verlauf der Versionen nachverfolgt werden kann. Falls die `Ref` noch nicht existiert, wird sie neu erstellt und mit der `VerUnit` verknüpft. Abschließend wird der Analyseprozess gestartet und der Link zur Detailseite der `VerUnit` zurückgesendet. Ein eingeloggter Nutzer mit entsprechender Berechtigung kann damit direkt auf die Ergebnisse zugreifen.

6.3 API-Key Management

Zur Authentifizierung der Integration im Backend wird, ein API-Key verwendet. Zur Verwaltung dieser Schlüssel wurde innerhalb der Organisationseinstellungen eine dedizierte Verwaltungsseite implementiert. Auf dieser Seite können API-Keys erstellt, eingesehen und deaktiviert werden. Eine beispielhafte Benutzeroberfläche mit aktiven und deaktivierten API-Keys ist in Anhang A, Abbildung A.5 dargestellt.

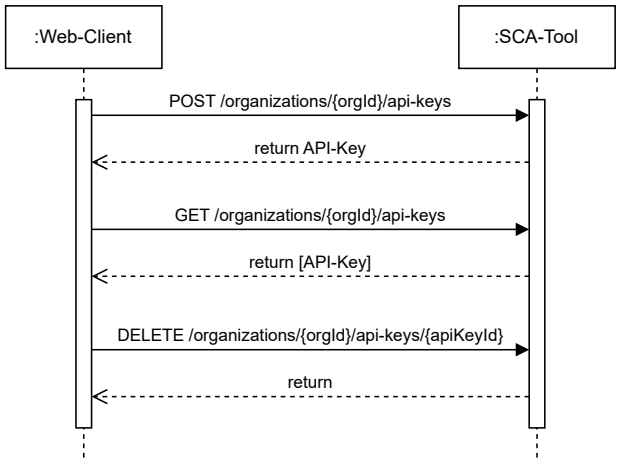


Abbildung 6.5: Sequenzdiagramm der unterschiedlichen Requests zur Verwaltung der API-Keys

Abbildung 6.5 zeigt ein Sequenzdiagramm, das die drei möglichen Operationen zwischen Web-Anwendung und Backend im Zusammenhang mit dem API-Key-Management darstellt.

Erstellung eines API-Keys:

Ein API-Key wird über den Endpunkt `/organization/{orgId}/api-keys` mittels eines `POST`-Requests erzeugt. Dabei werden der vom Nutzer optional angegebene Name sowie ein Ablaufzeitpunkt des Keys an das Backend übermittelt. Der Name dient der einfacheren Identifikation des Keys durch den Nutzer. Das Ablaufdatum legt fest, wie lange der Schlüssel gültig sein soll. Standardmäßig ist ein neuer API-Key für ein Jahr aktiv. Dieser Zeitraum kann jedoch vor der Erstellung in der Benutzeroberfläche angepasst werden. Als Antwort auf den Request erhält der Client das zugehörige API-Key-Objekt samt Klartextwert des Schlüssels. Dies ist die einzige Möglichkeit, den Key in Klartext vom Server zu erhalten. Der Server speichert ausschließlich die gehashte Variante des API-Keys und kann den ursprünglichen Wert daher nicht rekonstruieren.

Abrufen vorhandener API-Keys:

Beim Öffnen der Verwaltungsseite wird ein `GET`-Request an denselben Endpunkt wie zuvor gesendet. Die Antwort enthält eine Liste aller aktiven sowie deaktivierten oder abgelaufenen API-Keys, die noch nicht gelöscht wurden. Die zurückgegebenen API-Key-Objekte enthalten nur die notwendigen Informationen. Aus

Sicherheitsgründen wird der Hash des Schlüssels nicht mitgesendet. Zur eindeutigen Identifikation eines Eintrages dient daher ein separates ID-Feld.

Deaktivierung eines API-Keys:

Zum Deaktivieren eines Schlüssels wird ein `DELETE`-Request mit der zugehörigen Key-ID an den Endpunkt `/organizations/{orgId}/api-keys/{apiKeyId}` gesendet. Der Server markiert den entsprechenden API-Key als inaktiv und bestätigt die erfolgreiche Deaktivierung. Der inaktive API-Key bleibt anschließend für einen Zeitraum von mindestens 30 Tagen sichtbar. Dies dient dazu, dem Nutzer die Möglichkeit zu geben, betroffene Integrationen zu identifizieren und entsprechend anzupassen. Der Key ist jedoch nicht mehr zur Authentifizierung nutzbar. Nach Ablauf der 30 Tage wird der API-Key beim nächsten Durchlauf des entsprechenden Jobs endgültig gelöscht.

7 Implementierung

In diesem Kapitel werden Besonderheiten der Implementierung der in Kapitel 5 beschriebenen Architektur und des in Kapitel 6 dargestellten Designs erläutert.

7.1 OpenAPI

Zum Start der Arbeit waren alle API-Definitionen in den verschiedenen Controllern in der Spring-Applikation verteilt. Deshalb wurden diese Definitionen zunächst dem OpenAPI 3.0.3 Standard entsprechend angepasst. Als Tool wurde SpringDoc-OpenAPI[1] genutzt. Neben der Generierung der OpenAPI-Spezifikation aus dem Java-Spring-Code stellt es ebenfalls die Swagger-Benutzeroberfläche zur Analyse der API bereit. Dabei werden alle verfügbaren Schnittstellen aufgelistet, dokumentiert und eine Umgebung bereitgestellt, in der die einzelnen API-Schnittstellen ausprobiert werden können.

Nach der initialen Einrichtung konnte auf eine unvollständige Ansicht der API-Schnittstellen und die API-Definition zugegriffen werden. Die dabei erstellte Spezifikation erfüllte zwar den OpenAPI-Standard, enthielt aber keine genauen Informationen zu möglichen Antworten oder Beschreibungen der einzelnen Parameter. Um diese zu vervollständigen, gibt es zwei Möglichkeiten. Entweder kann die OpenAPI-Definition von Hand erstellt und verlinkt werden, oder die entsprechenden Controller und dazugehörigen Modelle werden mittels Annotationen genauer spezifiziert. Um auch für zukünftige Änderungen immer eine aktuelle Spezifikation erhalten zu können, wurde der Code annotiert.

Die wichtigsten Annotationen für Datenmodelle sind:

- @NotNull: Ein Attribut das bei jeder Anfrage gesetzt sein muss.
- @Id: Ein Attribut das ein Objekt eindeutig identifiziert.

[1] https://springdoc.org/

und für API-Endpunkt-Definition:

- @Operation: Textuelle Beschreibung des Endpunkts. Dabei wird zwischen einer Zusammenfassung und einer ausführlichen Beschriebung unterschieden.
- @ApiRespone: Definition der möglichen Response-Codes, deren Bedeutung und die Struktur der Antwort.
- @Parameter: Für jeden Parameter des Endpunkts kann eine textuelle Beschreibung und eine Angabe über die Notwendigkeit des Parameters festgelegt werden.

Die verwendeten Annotationen konnten nicht nur zur Erzeugung einer OpenAPI-Spezifikation genutzt werden, sondern dienten auch einer Dokumentation des Codes. Die resultierende OpenAPI-konforme API-Definition wurde im Anschluss von anderen Entwicklern des SCA Tool Projektes verwendet, um eine typsichere Nutzung der API sowohl in der Webanwendung als auch im Backend sicherzustellen. Durch diese Verwendung der Spezifikation hat sich der Stand des Projektes und die Verwendung der Annotation entscheidend geändert. Dabei werden mit Hilfe des OpenAPI-Generators[2] aus der erstellten OpenAPI-Spezifikation automatisch die Interfaces der API für beide Kommunikationsseiten generiert. Im Backendcode können diese generierten Interfaces beispielsweise direkt in den Controllern der API verwendet werden. Dadurch wird sichergestellt, dass der Controller exakt die Schnittstelle implementiert, die auch von der Webanwendung erwartet und in der Spezifikation definiert wurde. In der Webanwendung dient die OpenAPI-Spezifikation insbesondere dazu, die verwendeten Objekt-Schemata zu generieren, die zur Typisierung der Schnittstellenobjekte verwendet werden. Im Zuge dieser Umstellung wurden Swagger sowie die zuvor beschriebenen Annotationen aus dem Projekt entfernt.

Die OpenAPI-Spezifikation stellt nun die primäre Quelle für alle relevanten Details der Schnittstellen dar - statt umgekehrt. Damit folgt SCA Tool dem API-first-Ansatz, bei dem die Schnittstellendefinition eines Features vor dessen Implementierung erfolgt. Dieses Prinzip wurde konsequent in der weiteren Entwicklung der benötigten Endpunkte und der Public-API angewendet. Wie in Kapitel 5.1 erklärt, ist die Public-API für die Kommunikation mit der Integration verantwortlich. Sie ist in einer separaten Spezifikation beschrieben, mit der in einem eigenen Generator-Durchlauf die notwendigen Klassen für den Integrations-Controller erzeugt werden.

[2] https://openapi-generator.tech/

7.2 API-Key

Neben der bereits existierenden Web-API wurde eine Public-API erstellt. Diese ist für die Kommunikation mit der GitHub Action, der Gitlab-Component und potenziellen weiteren Integrationen zuständig.

Zur sicheren Kommunikation mit SCA Tool nutzt die Web-App eine cookiebasierte Authentifizierung gegenüber dem Backend. Dabei wird ein Cookie mit einem Session-Token als Header mit jedem Request an das Backend geschickt. Diesen erhält der Client bei der Anmeldung über die Benutzeroberfläche. Wie in Kapitel 2.3 erwähnt, ist diese Authentifizierungsweise jedoch nicht mit der Integration nutzbar, da dies ein fester Token sein muss, der einmalig vom Nutzer gesetzt wird und der ohne direkten Zugriff auf den ausführenden Runner funktionieren muss. Daher wurde eine Authentifikation mittels API-Keys implementiert. Der API-Key wird dabei bei der Kommunikation mit der Public-API zur Authentifizierung mitgesendet und durch einen Filter, wie in Kapitel 5.1 erklärt, gefiltert.

Der API-Key kann in der SCA Tool Web-Oberfläche erzeugt werden und für die Nutzung kopiert werden. Das geht sowohl in der Konfigurationsseite eines Projekts, als auch in den Organisationseinstellungen. Eine vertrauliche Handhabung des API-Keys ist entscheidend, da durch Missbrauch fremde Daten hochgeladen werden könnten.

7.2.1 Generierung

Der generierte API-Key besteht aus drei Teilen.

1. Ein Wortmuster um die einzelnen Teile des Keys einfach extrahieren zu können. Hierfür wurde am Anfang des API Keys `sca` und in der Mitte `tool` hinzugefügt.
2. Ein Ablaufsdatum des API-Keys zur Prüfung der Gültigkeit.
3. Der Eigentliche Key-Teil. Eine zufällig generierte Zeichenfolge.

Ein Beispiel für einen generierten API-Key ist:

`sca2026-06-14toolDo2zos6y6ftjnI7_C8RtF1k7iyaBB9iLVSQWTMP48xc`

Wobei der 14.06.2026 das Ablaufdatum ist und

`Do2zos6y6ftjnI7_C8RtF1k7iyaBB9iLVSQWTMP48xc`

der eigentliche Schlüssel.

Zur Generierung des Schlüssels wird ein 32-Byte SecureRandom erzeugt, der im Anschluss mithilfe von Base64 zu einem nutzbaren String umgewandelt wird.

SecureRandom ist ein kryptographisch sicherer Zufallszahlengenerator. (Oracle, 2025) Generierte Bytes können nicht reproduziert werden, wodurch generierte Schlüssel zwar wirklich zufällig sind, jedoch kann eine Doppelung von generierten Werten nicht ausgeschlossen werden. Diese ist jedoch aufgrund der Länge von 32 Byte und der Menge der möglichen Werte unwahrscheinlich, da es insgesamt 2^{256} mögliche Werte gibt. Dennoch wurde zur Sicherheit der APIKey als Attribut mit einzigartigen Werten in der Datenbank gekennzeichnet. Dadurch verhindert die Datenbank, dass ein API-Key mehrfach vergeben werden kann. Dies gilt auch für den gehashten Wert des API-Keys.

7.2.2 Hashing des API-Keys

Da über den API-Key eine Authentifizierung auf Seite des Servers stattfindet, muss dieser für den Abgleich in der Datenbank gespeichert werden. Für den Fall, dass ein Angreifer Zugriff auf die Datenbank erhält, ist es wichtig, dass API-Keys, genauso wie Nutzer-Passwörter, nicht in Klartext gespeichert sind. Daher wird dieser in der Datenbank gehasht gespeichert. Hierfür wird der API-Key mit dem in Kapitel 2.4 vorgestellten HMAC-SHA-256-Algorithmus gehasht und in der Datenbank gespeichert. Durch die genutzte Hashing-Mehtode wird sichergestellt, dass zwei gleiche Keys denselben Hash ergeben. Dies ermöglicht uns einerseits den Vergleich der Hashes zum Finden des API-Key-Eintrags in der Datenbank. Auf der anderen Seite wird durch die Einzigartigkeit der Einträge in der Datenbank verhindert, dass ein Key mehrmals vergeben werden kann.

Für den Fall, dass die Datenbank versehentlich veröffentlicht wird, besteht die Gefahr, dass die darin gespeicherten Hash-Werte rekonstruiert werden können. Eine mögliche Angriffsform ist die Brute-Force-Attacke, bei der sämtliche Eingabewerte systematisch durchprobiert werden. Durch die Länge des erzeugten Keys ist dies jedoch sehr aufwendig. Darüber hinaus existieren optimierte Methoden wie sogenannte Rainbow-Tabellen, in denen vorberechnete Hash-Werte für häufig verwendete Zeichenfolgen gespeichert sind. Ein Angreifer vergleicht dabei die Hashes der Datenbank mit den Einträgen in der Tabelle, um dadurch Rückschlüsse auf die ursprünglichen Werte zu ziehen. Da der Key aus zufälligen Zeichenfolgen besteht, sind die Möglichkeiten hierfür bereits eingeschränkt. Um solchen Angriffen zusätzlich entgegenzuwirken, wird in der in Kapitel 2.4 vorgestellten Hashing-Methode der HMAC-Algorithmus eingesetzt. Dabei werden die Einträge gegen vorberechnete Tabellen geschützt, da Versuche ohne Kenntnis des geheimen Schlüssels wirkungslos sind. Ein Angreifer müsste für jeden möglichen geheimen Schlüssel jeweils alle potenziellen Eingabewerte hashen, was die Komplexität der Attacke exponentiell erhöht. (Bhorkar, 2017)

7.2.3 Ablaufdatum

Das Ablaufdatum eines API-Keys stellt eine zusätzliche Sicherheitsmaßnahme dar. Zum einen verlieren auch versehentlich veröffentlichte Schlüssel nach dem angegebenen Datum automatisch ihre Gültigkeit, sodass ein Missbrauch zeitlich beschränkt werden kann. Zum anderen wird der Nutzer dadurch dazu angehalten, regelmäßig neue Schlüssel zu generieren und dabei bestehende, möglicherweise nicht mehr benötigte API-Keys zu überprüfen und zu deaktivieren.

Das Ablaufdatum wird als Teil des übermittelten Tokens an den Server mitgesendet. Dadurch wird eine missbräuchliche Nutzung erschwert, selbst wenn der zufällig generierte Teil des Schlüssels erraten oder kompromittiert wurde. Denn das im Token enthaltene Ablaufdatum muss exakt mit dem in der Datenbank gespeicherten Datum übereinstimmen, um als gültig zu gelten. Somit erhöht sich die Komplexität des geheimen Teils um 8 zusätzliche Zeichen, was die Wahrscheinlichkeit eines erfolgreichen Angriffs weiter reduziert.

Ein weiterer Vorteil besteht darin, dass CI/CD-Abläufe mit abgelaufenen Tokens bereits auf der Seite der Integration beendet werden können, bevor eine Kommunikation mit dem Server stattfindet. Zudem kann die Integration den Nutzer proaktiv über das bevorstehende Ablaufdatum eines verwendeten Tokens informieren, damit dieser rechtzeitig einen neuen Schlüssel erzeugen kann.

7.2.4 Weitere Sicherheitsmaßnahmen

Wie in Kapitel 6.3 beschrieben, ist die einzige Möglichkeit, einen erzeugten API-Key zu erhalten, als Antwort auf die Erstellung. Dies ist nicht nur eine technische Notwendigkeit, da der Server den Klartext des Keys nicht speichert, sondern dient auch dem Schutz vor potenziellen Zugriffen über nicht abgemeldete Sitzungen oder Session Hijacking. Somit kann der API-Key ausschließlich von der Person eingesehen werden, die ihn erstellt hat. Dies schließt jedoch nicht aus, dass der Key nachträglich unsicher gespeichert wird. Daher wurde der Konfigurationsseite eine Anleitung hinzugefügt, wie der API-Key sicher als CI/CD-Variable in den jeweiligen Plattformen hinterlegt werden kann.

Darüber hinaus wird der API-Key lediglich für die Authentifizierung genutzt. Wie in Kapitel 2.3 erwähnt, sollten API-Keys mit weiteren Aspekten kombiniert werden, um Nutzerdaten zu sichern. Obwohl im aktuellen Projektstatus mit dem API-Key ausschließlich Uploads erlaubt sind und keine Daten aus dem System abgerufen werden können, muss dennoch verhindert werden, dass fremde Dateien in ein Projekt eingeschleust werden. Deshalb wird zur vollständigen Autorisierung jeder Anfrage innerhalb des Controllers die Anfrage mittels der Projekt-ID und dem Repository-Namen zusätzlich abgesichert. Nur wenn diese Merkmale konsistent sind, wird der Upload akzeptiert. Somit ergibt sich ein mehrstufiges Absicherungsmodell, das aus einem gültigen API-Key, der korrekten Projekt-ID,

dem passenden Ablaufdatum sowie dem zugehörigen Repository-Namen besteht. Diese Kombination erschwert unbefugte Zugriffe erheblich und schützt vor versehentlichen oder gezielten Fehlkonfigurationen einer Integration.

7.3 SCA Tool System

Bei der Implementierung der Integration auf Seite des SCA Tool wurden die neuen Entitäten `APIKey`, `Ref`, `ActionOrigin` und `VerUnitParent` erstellt. Außerdem wurden die notwendigen Änderungen am Frontend und der dazugehörigen API gemacht. Dabei wurde vor allem die Navigierbarkeit zwischen `VerUnits` und die Konfigurationsanleitung der Integration implementiert. Neben der Anpassung der Sicherheitskonfiguration zur Absicherung der Kommunikation mittels API-Key wurde der Controller für die Integration und dessen API erstellt. Im Folgenden sollen einige Besonderheiten der Implementierung dieser Sachen hervorgehoben werden.

7.3.1 Wiederkehrende Jobs zur Verwaltung der API-Keys

Wie bereits mehrfach erwähnt, enthält jeder API-Key auch ein festgelegtes Ablaufdatum. Darüber hinaus kann ein API-Key manuell deaktiviert werden. Zur besseren Identifikation ungültiger Schlüssel wurde daher ein zusätzliches Boolean-Feld `is_active` eingeführt. Dieses Kennzeichen gibt an, ob ein API-Key derzeit aktiv und zur Authentifizierung zugelassen ist. Ebenfalls wurde ein Datumsfeld hinzugefügt, das den Zeitpunkt der letzten erfolgreichen Authentifizierung mit dem jeweiligen API-Key speichert. Diese beiden zusätzlichen Felder dienen nicht nur der Anzeige und Verwaltung, sondern sind auch Grundlage für zwei wiederkehrende Hintergrundprozesse (Jobs). Der erste Job läuft täglich um Mitternacht und markiert automatisch alle API-Keys als inaktiv, deren Ablaufdatum überschritten wurde. Der zweite Job wird jeweils zum Monatsanfang ausgeführt. Er löscht alle API-Keys, die sowohl als inaktiv markiert sind als auch vor mehr als 30 Tagen deaktiviert wurden. Durch diese Mechanismen werden abgelaufene Schlüssel zunächst deaktiviert und nach einem definierten Zeitraum automatisch gelöscht.

7.3.2 Rekursive SQL-Abfragen

Die `VerUnitParent` ist eine Supportklasse, die zur Abbildung des zeitlichen Verlaufs von Versionen dient. Wie in Kapitel 5.2 beschrieben, werden hierfür Schlüsselpaare aus zwei `VerUnits` gebildet. Ein Eintrag kennzeichnet jeweils eine Version als Vorgänger (`parent`) einer anderen. Über diese Beziehung kann für jede `Ref` eine chronologische Liste aller älteren `VerUnits` aufgebaut werden.

Eine zentrale Herausforderung bei dieser Modellierung ist die effiziente Abfrage mehrerer vorhergehender oder nachfolgender Versionen einer bestimmten `VerUnit`. Dies betrifft insbesondere die Ermittlung der neuesten oder ältesten Version innerhalb einer Versionskette. Die dafür benötigten SQL-Abfragen erfordern rekursive Strukturen. Um beispielsweise die drei vorhergehenden Versionen einer bestimmten `VerUnit` zu finden, muss ausgehend vom aktuellen Eintrag, wiederholt die jeweils verknüpfte `parent_id` abgefragt werden. Ein Ansatz zur Umsetzung besteht darin, diese Abfragen sequenziell in der Anwendung auszuführen, indem jeweils ein Eintrag geladen und auf dessen Elternbeziehung geprüft wird, bis die gewünschte Anzahl an Vorgängerversionen gefunden ist. Alternativ kann dieses Problem durch den Einsatz einer rekursiven SQL-Abfrage als Teil der Common Table Expressions (CTE) von SQL[3] gelöst werden. Letzteres bietet insbesondere bei längeren Versionsketten Vorteile in Bezug auf Performance und Wartbarkeit der Datenbanklogik.

Listing 7.1: SQL Abfrage zum Finden der x vorherigen VerUnits.

```
@Query( value = """
    WITH RECURSIVE VerUnitHierarchy AS (
        SELECT ver_unit_id, parent_id, 1 AS level
        FROM ver_unit_parent
        WHERE ver_unit_id = :verUnitId

        UNION ALL

        SELECT vup.ver_unit_id, vup.parent_id, vuh.level + 1
        FROM ver_unit_parent vup
        INNER JOIN VerUnitHierarchy vuh ON vup.ver_unit_id =
            vuh.parent_id
            WHERE vuh.level < :maxLevel
    )
    SELECT vu.*
    FROM VerUnitHierarchy vuh
    JOIN ver_units vu ON vu.id = vuh.parent_id
    ORDER BY vuh.level""",
    nativeQuery = true
)
List<VerUnitEntity> findPreviousVerUnits(@Param("verUnitId")
    UUID verUnitId, @Param("maxLevel") int maxLevel);
```

[3] https://www.postgresql.org/docs/current/queries-with.html#QUERIES-WITH-RECURSIVE

Die in Listing 7.1 dargestellte rekursive SQL-Abfrage nutzt die spezifischen Möglichkeiten von PostgreSQL zur Definition nativer Queries. Sie dient dazu, alle vorhergehenden `VerUnits` bis zu einer maximalen Tiefe `maxLevel` zu ermitteln.

Im ersten Teil der Abfrage wird die Ausgangs-`VerUnit` mit ihrem `parent`-Eintrag und einem initialen Level-Wert von 1 selektiert. Im rekursiven Teil werden iterativ die jeweils übergeordneten `VerUnits` ermittelt, wobei das Level bei jedem Schritt inkrementiert wird. Die Rekursion wird gestoppt, sobald das definierte maximale Level erreicht ist.

Im äußeren Teil der Abfrage werden die ermittelten Vorgänger-IDs mit den vollständigen Einträgen aus der Tabelle `ver_units` verknüpft. Abschließend erfolgt eine Sortierung anhand der Hierarchieebene `level`.

Ein ähnliches Prinzip wird in der Abfrage der aktuellsten Version einer beliebigen `VerUnit` verwendet. Diese wird benötigt, um die `Ref` einer `VerUnit` finden zu können. Sie ist in Listing 7.2 zu sehen.

Listing 7.2: SQL Abfrage zum Finden der aktuellsten VerUnit.

```
@Query(value = """
    WITH RECURSIVE chain(ver_unit_id, parent_id, depth) AS (
        SELECT ver_unit_id, parent_id, 1 as depth
        FROM ver_unit_parent
        WHERE parent_id = :startId

        UNION ALL

        SELECT v.ver_unit_id, v.parent_id, c.depth + 1
        FROM ver_unit_parent v
        INNER JOIN chain c ON v.parent_id = c.ver_unit_id
    )
    SELECT ver_unit_id
    FROM chain
    ORDER BY depth DESC
    LIMIT 1;""",
    nativeQuery = true
)
Optional<UUID> findLatestChildVerUnit(@Param("startId") UUID
    startId);
```

Der Unterschied zur vorherigen Abfrage ist hierbei das entfernte Abbruchkriterium der Tiefe im rekursiven Teil. Dadurch wird die Kette solange von einem `parent` zur `VerUnit` abgeschritten, bis sie endet. Am Ende wird nur die tiefste, also neueste Version ausgewählt. Sollte keine gefunden werden, ist die Ausgangsversion bereits die neueste Version.

7.3.3 Konfigurationsseite einer Integration

Ein zentraler Aspekt der benutzerfreundlichen Nutzung der CI/CD-Integration ist eine leicht verständliche und geführte Einrichtung für Nutzer mit unterschiedlichem Vorwissen. Hierzu wurde eine dedizierte Konfigurationsseite für eine Action-Origin entwickelt. Wählt der Nutzer als Projekttyp die Option `Integrate with CI`, kann er zunächst die Repository-URL eines GitHub- oder GitLab-Repositories eingeben. Basierend auf dieser Eingabe wird der nächste Konfigurationsschritt automatisch an die jeweilige Plattform angepasst. Die zugehörigen Konfigurationsseiten sind in Anhang A, Abbildung A.3 (GitHub) und A.4 (GitLab) dargestellt. Ein manuelles Umschalten zwischen den Plattformen ist ebenfalls möglich, falls etwa durch die Nutzung einer eigenen Enterprise-Instanz die automatische Erkennung fehlschlägt.

Der zweite Konfigurationsschritt gliedert sich in zwei Teile: Zum einen die optionale Erstellung eines API-Keys und zum anderen die Generierung und Vervollständigung der benötigten Konfigurationen für die gewählte CI/CD-Plattform.

Zur Erstellung eines API-Keys wurde die zuvor implementierte Komponente aus den Organisationseinstellungen wiederverwendet. Zusätzlich wird dem Nutzer ein Link angezeigt, der aus der Repository-URL abgeleitet wird und für die sichere Speicherung des API-Keys auf der Zielplattform verwendet werden kann. Dies hilft dem Nutzer nicht nur bei der sicheren Ablage des API-Keys, sondern ermöglicht gleichzeitig eine Überprüfung der URL-Eingabe und liefert Hinweise zur korrekten Konfiguration.

Der zweite Teil des Konfigurationsschritts umfasst die Erstellung und Vervollständigung der Konfiguration der Integration für die CI/CD-Plattform. Dabei kann zunächst die Liste der auszuschließenden Dateien, Ordner oder auch Dateitypen in ein Eingabefeld eingegeben werden. Anschließend kann die resultierende Konfiguration entweder direkt kopiert oder alternativ als Datei heruntergeladen und in das entsprechende Repository hochgeladen werden. Die Benennung der Konfigurationsdatei richtet sich dabei nach der jeweiligen Plattform. So wird eine korrekte Einrichtung der Integration sichergestellt und der Upload-Prozess kann automatisiert erfolgen. Im Fall von GitHub kann der Nutzer optional wählen, ob automatisch ein Kommentar mit einem Link zum Upload gepostet werden soll.

Standardmäßig ist die erstellte Konfiguration so definiert, dass Uploads nur im Rahmen aktiver Pull-Requests ausgelöst werden. Dies kann jedoch vom Nutzer jederzeit geändert werden, um optimal auf den Entwicklungsablauf angepasst zu werden.

7.4 GitHub-Action

Aus den in Kapitel 2.2.1 vorgestellten Möglichkeiten, eine eigene GitHub Action zu erstellen, wurde sich für die JavaScript-Variante entschieden. Die Implementierung wurde in TypeScript angefertigt, die vor der Veröffentlichung zu JavaScript transpiliert und mit den benötigten Abhängigkeiten gebündelt wurde.

Die Wahl fiel auf TypeScript, da diese Sprache bereits für die Benutzeroberfläche des SCA Tool verwendet wird und entsprechend Erfahrung im Team vorhanden war. Darüber hinaus ermöglicht eine JavaScript-basierte Action eine breite Kompatibilität mit allen GitHub-Runner-Betriebsystemen, wodurch andere genutzte Actions nicht eingeschränkt werden. Der erstellte Code konnte außerdem mit geringem Anpassungsaufwand für die GitLab-Component wiederverwendet werden.

Mit Hilfe des von GitHub bereitgestellten Packages konnten neben Funktionen zur besseren Kommunikation mit dem CI/CD-System auch kontextspezifische Informationen wie Repository, Commit oder Ref sicher ausgelesen werden. Diese Metadaten sind in der GitHub-Dokumentation[4] dokumentiert.

Eine wichtiger Parameter ist dabei die `Ref`, die je nach Ausführungskontext unterschiedliche Inhalte besitzt. Dies ist durch die Konfiguration, wann eine Action ausgeführt wird, festgelegt. Bei einer Ausführung aufgrund eines Tag-Events ist sie beispielsweise `refs/tags/v1.0.0`, bei einem Pull Request `refs/pull/x/merge`, wobei `x` die Pull-Request-ID darstellt. Diese Referenz wird im SCA Tool genutzt, um hochgeladene VerUnits eindeutig zu gruppieren.

Die Nutzung des erstellten Integrationsscript in der erstellten Action ist durch die Art der Action ohne Probleme möglich. Dafür wird in einer `action.yaml` Datei die In- und Outputs und die transpilierte JavaScript-Datei referenziert.

Listing 7.3: Einbindung der GitHub-Action in bestehenden Workflow

```yaml
- name: Run SCA Tool Scan
    uses: scatool/sca-tool-action@v0.4.0
    with:
        api_key: ${{ secrets.SCA_TOOL_API_KEY }}
        project_id: 'aaaa-aaaa-aaaa-aaaa'
```

Listing 7.3 zeigt ein Beispiel einer Konfiguration, die ein Nutzer benötigt, um die Action in einem bestehenden Workflow hinzuzufügen. Dabei müssen lediglich der Name der Action und die Eingabeparameter spezifiziert sein. Alle dafür benötigten Daten kann der Nutzer auf der Konfigurationsseite des Projektes finden oder die generierte Konfiguration kopieren.

[4] https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/accessing-contextual-information-about-workflow-runs#github-context

7.5 GitLab-Component

Wie in Kapitel 2.2.2 beschrieben, setzt GitLab auf die Nutzung von Docker als Ausführungsumgebung. Die Pipeline definiert dazu ein Image, in dem die festgelegten Befehle ausgeführt werden.

Zur Reduktion von Komplexität und Verbesserung der Wartbarkeit wurde das in TypeScript erstellte Integrationsscript wiederverwendet. Dieses wird innerhalb eines `node`-Containers ausgeführt.

Die wesentliche Anpassung betraf die Ersetzung der GitHub-spezifischen Funktionen durch äquivalente Mechanismen für GitLab. Dazu zählt insbesondere die Nutzung vordefinierter GitLab-Variablen[5], über die Repository-Name, Referenzname und Commit-SHA abgefragt werden.

Auf eine Kommentarfunktionalität wurde in GitLab bewusst verzichtet. Im Gegensatz zu GitHub, wo der für Actions bereitgestellte Token bereits über die benötigten Rechte verfügt, müsste ein GitLab-Nutzer einen separaten Token mit umfassenden Schreibrechten erzeugen, was ein zusätzliches Sicherheitsrisiko darstellen würde.

Listing 7.4: GitLab-Component Konfiguration

```yaml
spec:
  inputs:
    api_url:
      default: "https://app.scatool.com/api/v1/"
      description: "The base URL of the SCA Tool API."
    project_id:
      description: "The unique project-ID in SCA Tool."
    excluded_paths:
      description: "Comma-separated list of paths in glob-
         format to exclude from the scan."
    api_key:
      description: "The API key for authenticating with the
         SCA Tool API."
    stage:
      default: "test"
      description: "The stage in which the scan will run."

---
sca_tool_scan:
  stage: $[[ inputs.stage]]
  image: node:latest
  variables:
```

[5] https://docs.gitlab.com/ci/variables/predefined_variables/

```
21      API_URL: $[[ inputs.api_url ]]
22      PROJECT_ID: $[[ inputs.project_id ]]
23      EXCLUDED_PATHS: $[[ inputs.excluded_paths ]]
24      API_KEY: $[[ inputs.api_key ]]
25      GITLAB_TOKEN: $[[ inputs.gitlab_token ]]
26    script:
27      - echo "Copying action scripts..."
28      - git clone --depth=1 https://gitlab.com/scatool/gitlab-
            job.git scatool-gitlab
29      - cp -r scatool-gitlab/dist ./sca-tool-script
30      - rm -rf scatool-gitlab
31      - echo "Running SCA Tool Scan..."
32      - node sca-tool-script/index.js --api_url=$API_URL --
            excluded_paths="$EXCLUDED_PATHS" --api_key=$API_KEY
            --project_id=$PROJECT_ID --gitlab_token=$GITLAB_TOKEN
             | tee sca_tool_scan.log
```

Listing 7.4 zeigt die Definition der erstellten GitLab-Component. Sie besteht aus der Auflistung der Inputs und der Job-Konfiguration, inklusive der innerhalb des Docker-Containers ausgeführten Befehle. Im Gegensatz zu GitHub enthält diese einen zusätzlichen Parameter mit dem Namen `stage`. Wie in Kapitel 2.2.2 erklärt, nutzt GitLab sogenannte Stages, um den Ausführungszeitpunkt innerhalb der Pipelines flexibel zu steuern. So kann eine eigene Stage angegeben werden, wodurch der erstellte Job zu dem durch den Nutzer festgelegten Punkt ausgeführt wird. Wird kein Wert angegeben, erfolgt die Ausführung in der Standard-Stage `test`.

Ein wesentlicher Unterschied zu GitHub ist, dass GitLab-Components nicht die Möglichkeit bieten, neben der Konfiguration andere Dateien zur Nutzung während der Ausführung zu bündeln. Um dies zu umgehen, kann ein eigenes Docker-Image erstellt werden, das bereits die benötigten Dateien enthält. Alternativ kann das zum Release kompilierte Skript über einen alternativen Weg heruntergeladen werden. Wie in Listing 7.4 in Zeilen 28 bis 30 zu sehen ist, wird für die erstellte Component das Repository geklont, wobei mithilfe des `depth`-Parameters nur der aktuellste Commit geladen wird. Anschließend werden alle nicht relevanten hinzugefügten Dateien entfernt, um versehentlich zusätzliche hochgeladene Dateien zu verhindern. Zum Schluss wird das Skript per Node.js ausgeführt, wobei die Konfigurationswerte als Parameter übergeben werden. Die Ausgaben des Skriptes erscheinen dann im Log der Pipeline, wo sie durch den Nutzer eingesehen werden können.

Listing 7.5: Einbindung der GitLab-Component in bestehende CI/CD-Pipeline

```yaml
include:
  - component: $CI_SERVER_FQDN/scatool/gitlab-job/scatool-gitlab@v0.4.0
    inputs:
      project_id: 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa'
      api_key: $SCATOOL_API_KEY
```

Listing 7.5 zeigt ein minimales Beispiel der Einbindung der GitLab-Komponente in eine bestehende Pipeline eines GitLab-Repositories. So muss lediglich die Projekt-ID eingefügt werden und der API-Key in die CI/CD-Variablen des Repositories hinzugefügt werden. Im Falle eines Updates der Komponente muss auf beiden Plattformen im besten Fall lediglich die Versionsnummer angepasst werden.

7.6 Integrationsscript

Bei der Entwicklung des Integrationsscripts wurde besonderer Wert auf Modularität gelegt. Dabei existiert ein zentrales Hauptskript, von dem aus die in Kapitel 6.1 beschriebenen Funktionen aufgerufen werden. Dazu zählt etwa die Validierung der zu übertragenden Dateien. Durch diese Modularität können leicht einzelne Teile getauscht oder verbessert werden, ohne andere Funktionalitäten dadurch zu beeinflussen. Die klare Trennung von plattformabhängigem und -unabhängigem Code ermöglicht es, Änderungen unkompliziert zwischen den Integrationen zu übertragen – unabhängig von der konkreten Implementierung der Zielplattform.

Während die Kommunikation mit dem Produktionsserver stets sicher mittels `https` abläuft, wird dies von den Testservern nicht unterstützt. Das Skript soll jedoch sowohl mit den Testservern als auch dem Produktionsserver kommunizieren können. Daher muss zwischen den Systemen unterschieden werden und dementsprechend das `http`- beziehungsweise das `https`-Protokoll verwendet werden. Dafür wird die angegebene URL des Zielservers analysiert. Sollte der Nutzer keinen angeben, wird standardmäßig der Produktionsserver mit dem sicheren `https`-Protokoll genutzt.

Zur Dateisuche wurde das npm-Package `fast-glob`[6] eingesetzt. Dabei kann mit Hilfe von Mustern sowohl die gesuchten Dateien als auch die ignorierten Dateien und Ordner festgelegt werden. Außerdem kann angegeben werden, dass nur Dateien als Ergebnisse zurückgegeben werden sollen. Die Muster werden dabei im `glob`-Format angegeben. Dies erlaubt es, spezifische Dateitypen, Pfade oder auch gezielt einzelne Dateien zu erfassen oder auszuschließen. Beispielsweise schließt das Muster `*/**.md` alle Markdown-Dateien, einschließlich der in Unterordnern

[6] https://github.com/mrmlnc/fast-glob

enthaltenen, ein. Die Verwendung dieses Formats entspricht dabei etablierten Standards, da etwa auch `.gitignore`-Dateien dieses Format nutzen.

8 Evaluation

In diesem Kapitel werden die in Kapitel 4 definierten Anforderungen evaluiert. Dabei wird jede Anforderung entweder als nicht erfüllt, teilweise erfüllt oder erfüllt bewertet.

8.1 Funktionale Anforderungen

Allgemein

F-1 Diese Anforderung wurde erfüllt. Das System hat eine separate API, die die Schnittstellen für die Kommunikation mit der Integration bereitstellt. Dies ist einerseits in Abbildung 5.1 dargestellt oder im Code als public.yaml zu finden.

F-2 Diese Anforderung wurde erfüllt. Die Public-API hat einen Endpunkt mit dem Namen `v1/integration/ci-triggered-upload` über den ein externes System die benötigten Dateien hochladen und einen Scan starten kann.

F-3 Diese Anforderung wurde erfüllt. Mittels des eingeführten Origin Typ Action-Origin, wird ein Projekt zur Nutzung durch eine Integration markiert. Dadurch kann die Repository-URL gesetzt werden, die benötigt wird, um Uploads über die API zu verarbeiten.

F-4 Diese Anforderung wurde erfüllt. Als Antwort auf den Upload-Request, gibt das System bei Erfolg den Link zu der spezifischen VerUnit in der dazugehörigen SCA Tool Webapplikation zurück. Über diesen kann der Nutzer auf die Ergebnisse zugreifen.

F-5 Diese Anforderung wurde erfüllt. Durch die Einführung der VerUnit-Parent Tabelle kann der zeitliche Verlauf in der Datenbank abgebildet werden. In der Benutzeroberfläche kann über die Auswahl der Version innerhalb der Breadcrumbs vorherige und nachfolgende Versionen ausgewählt werden.

F-6 Diese Anforderung wurde erfüllt. Durch die Einführung der Ref Entität und dem mitsenden des Kontexts durch den die Integration ausgelöst wurde, können Versionen unter Pull-Requests, Tags oder Branches gruppiert werden. Dafür ist die Auswahl einer Ref in den Breadcrumbs hinzugefügt worden. Dadurch kann schnell zu der neuesten Version einer Referenz gewechselt werden.

F-7 Diese Anforderung wurde erfüllt. Durch die VerUnitParent- und Ref-Entität kann der Nutzer in der Benutzeroberfläche zwischen hochgeladenen Versionen eines Projektes wechseln. Dafür kann er innerhalb der Breadcrumbs der Nutzeroberfläche sowohl zwischen Refs als auch den Versionen wechseln. Wird eine Ref ausgewählt, so wird zunächst die neueste Version dieser Ref angezeigt. Durch das letzte Dropwdown innerhalb der Breadcrumbs, kann die Version gewechselt werden. Diese sind nach ihrem hochlade Zeitpunkt sortiert.

F-8 Diese Anforderung wurde erfüllt. Das System validiert die Eingaben des Nutzers und gibt im Falle einer fehlerhaften Eingabe oder eines Systemsfehlers während der Erstellung einer neuen Version, einen Fehler als Antwort auf den Hochladeversuch zurück. Dabei wird zwischen dem Produktivsystem und Entwicklungssystemen unterschieden und entweder statische Fehlermeldungen die keinerlei Nutzerdaten enthalten, oder detaillierte Fehlermeldungen ausgegeben, die das debuggen von Fehlern vereinfachen sollen.

F-9 Diese Anforderung wurde erfüllt. Nach Auswahl des Projekttyps als Integration und der Eingabe der Repository URL bekommt der Nutzer eine Anleitung angezeigt, in der beschrieben ist, was konfiguriert werden muss, um einen Scan erfolgreich zu starten. Dabei wird eine Anleitung sowohl für GitHub als auch Gitlab zur Verfügung gestellt. Die entsprechende Benutzeroberfläche ist in Anhang A Abbildung A.3 (GitHub) beziehungsweise Abbildung A.4 (GitLab) zu sehen.

F-10 Diese Anforderung wurde erfüllt. Die Benutzeroberfläche stellt eine Integrations-Konfiguration bereit, die durch den Nutzer durch zusätzliche Eingaben weiter spezifiziert werden kann. Diese kann anschließend entweder in die Zwischenablage kopiert oder als Datei heruntergeladen werden, die im Anschluss lediglich in das entpsrechende System eingefügt werden muss. Die zu kopierende oder herunterzuladende Konfiguration kann in Anhang A Abbildung A.3 (GitHub) und Abbildung A.4 (GitLab) in Form eines Dunklen Kasten gesehen werden.

F-11 Diese Anforderung wurde erfüllt. Dem Nutzer werden aus der Repository-URL generierte Links für die sichere Speicherung des API-Keys an-

gezeigt. Dadurch wird indirekt auch die korrekte Konfiguration der URL geprüft. Außerdem wurden einige Links zu externen Plattformen ergänzt. Diese sollen dem Nutzer bei Problemen weiterhelfen, oder weitere Informationen zu der Nutzung der CI/CD-Komponente der jeweiligen Plattformen anbieten.

Integration

F-12 Diese Anforderung wurde erfüllt. Die Integration nutzt dafür die erstellte Public API und den enthaltenen Endpunkt `integration/ci-triggered-upload`.

F-13 Diese Anforderung wurde erfüllt. Da das Integrationsscript sowohl als GitHub-Action als auch als GitLab-Component nutzbar ist, ist es mit den beiden Systemen kompatibel.

F-14 Diese Anforderung wurde erfüllt. Mittels eines optionalen Parameters `serverUrl` kann eine alternative URL eingegeben werden. Dadurch kann entweder eine Test- beziehungsweise lokale Systeminstanzen genutzt werden. Auch der zurückgesendete Link in der Antwort des Backends entspricht dabei der URL der Instanz und kann somit ohne Probleme genutzt werden. Diese Funktionalität wird auch bereits durch das SCA Tool Team genutzt. Dabei wird als Teil der CI die Abhängigkeiten des Projektes in das Staging System hochgeladen. Dadurch kann SCA Tool nicht nur selbst analysiert werden, sondern auch die Funktionalität der Integration geprüft werden.

F-15 Diese Anforderung wurde erfüllt. Der Nutzer kann über den `excludedPaths` Parameter Dateien, Ordner und Dateitypen im glob-Format ausschließen. Bei einer Eingabe von `scanner/**` würde etwa der Ordner `scanner` und die darin enthalten Dateien ausgeschlossen und nicht an den Server geschickt werden.

F-16 Diese Anforderung wurde teilweise erfüllt. Die Integration ist in der Lage verschiedene Dateitypen zu finden. Jedoch werden die gesuchten Typen nicht vom Nutzer, sondern vom System bestimmt. Dies wurde entschieden, da dadurch das System entscheiden kann, welche Abhängigkeiten und Dateien es verarbeiten kann. Zusätzlich kann der Nutzer dennoch bestimmte Dateien und somit auch Dateitypen und ihre Technologien durch Spezifizierung in den `excludedPaths` ausschließen.

F-17 Diese Anforderung wurde erfüllt. Die Integration validiert die Eingaben bevor es eine Kommunikation mit dem Server aufbaut und gibt die Validierungsfehler zurück. Sollte ein Fehler auf der Seite des Ser-

vers auftreten, wird die Fehlermeldung des Backends zurückgegeben. Fehler die bereits von der Integration selber gefunden werden können sind fehlende notwendige Eingaben, abgelaufene API-Keys oder fehlerhafte Konfiguration, die zu einem leeren beziehungsweise ungültigen Upload führen würden. Wird ein Fehler festgestellt, endet das Script in einem Fehlerzustand, wodurch die CI/CD-Plattformen die Ausführung als Fehlgeschlagen einstufen.

F-18 Diese Anforderung wurde erfüllt. Die Integration holt sich die benötigten Daten von den jeweiligen Systemvariablen und übergibt sie als zusätzliche Parameter des Uploads.

F-19 Diese Anforderung wurde erfüllt. Die Integration gibt die Meldungen und Antwort des Integrationsscript über den Log in den Details des CI/CD-Durchlaufes auf der jeweiligen Plattformen weiter.

F-20 Diese Anforderung wurde teilweise erfüllt. Vor dem Upload wird überprüft ob mindestens eine der vom Backend definierten Dateikombinationen vorhanden ist. Das stellt allerdings nicht sicher, dass alle benötigten Dateien mitgeschickt wurden. Dieser Fehler kann erst im Analyzer auf dem Server identifiziert werden. Dieser Fehler ist daher erst auf der Übersichtsseite der hochgeladenen Version zu sehen.

F-21 Diese Anforderung wurde erfüllt. Die Integration überprüft vor dem Upload die Größe der hochzuladenden Dateien. Falls eine Datei die maximale Größe überschreitet, bricht das Skript mit einer entsprechenden Meldung ab.

F-22 Diese Anforderung wurde erfüllt. Durch die Veröffentlichung der Integration als GitHub-Action im GitHub-Marketplace[1] und als GitLab-Component im GitLab CI/CD-Catalog[2]. Dadurch können Nutzer diese einfach finden und mit wenigen Zeilen in ihre eigenen Abläufe integrieren.

F-23 Diese Anforderung wurde teilweise erfüllt. Diese Funktionalität ist für GitHub implementiert. Durch das setzen des Parameters `github_token` mit einem standardmäßig bereitgestellten Token im GitHub Workflow, wird die erfolgreiche Antwort des Servers mit dem Link direkt in den Pull Request gepostet. Für GitLab wurde dies nicht umgesetzt, da für das Posten von Kommentaren in GitLab ein extra Token benötigt wird, der nicht automatisch bereitgestellt wird. Der Nutzer müsste deswegen einen Token bereitstellen, der alle Berechtigung für ein Projekt benötigen würde. Um die Akzeptanz zu erhöhen und da die Inte-

[1] https://github.com/marketplace/actions/sca-tool-github-action
[2] https://gitlab.com/explore/catalog/scatool/gitlab-job

gration auch ohne diesen Token funktioniert und der Nutzer den Link zu den Ergebnissen effizient über die Pipeline Logs einsehen kann, wurde sich gegen eine Kommentarfunktion für GitLab entschieden.

API-Key

F-24 Diese Anforderung wurde erfüllt. Ein API-Key kann auf SCA Tool Seite erzeugt werden und der Integration durch den Nutzer über die jeweilige Plattform zur Verfügung gestellt werden. Diesen sendet die Integration bei einem Upload mit, um den Request zu authentifizieren.

F-25 Diese Anforderung wurde erfüllt. Durch die in Kapitel 7.2 beschriebenen Generierung eines 32-Byte SecureRandom Strings als Key und einer Speicherung als Hash, wird der Key in einer nicht reproduzierbaren Art generiert.

F-26 Diese Anforderung wurde erfüllt. Durch das Hinzufügen eines Ablaufdatums in den gesendeten API-Key und der entsprechenden Speicherung in der Datenbank, zusammen mit den in Kapitel 7.3.1 beschriebenen wiederkehrenden Jobs, können API-Keys nur für eine vorkonfigurierte Zeit verwendet werden. Dadurch müssen sie regelmäßig ausgetauscht werden.

F-27 Diese Anforderung wurde erfüllt. Wie in Anhang B dargestellt, ist ein API-Key einer Organisation zugeordnet. Zusätzlich ist der Tabelle eine Einschränkung hinzugefügt, wodurch nicht zwei Mal der selbe Hash in der Datenbank gespeichert werden kann. Dies ist auch trotz Hash-Funktion sichergestellt, da die gleichen Keys zu gleichen Hash Werten führen würden. Dadurch ist sichergestellt, dass ein API-Key einmalig ist und eindeutig einer Organisation zugewiesen werden kann.

F-28 Diese Anforderung wurde erfüllt. Das Format und Ablaufdatum werden sowohl als Teil des Integrationsscript als auch auf dem Server überprüft. Auf dem Server wird neben der Existenz des API-Keys auch geprüft, ob das Ablaufdatum manipuliert wurde.

F-29 Diese Anforderung wurde erfüllt. Der Nutzer kann über die Organisationseinstellung alle aktiven API-Keys einsehen und wenn gewollt deaktivieren. Dadurch wird im Backend die `isActive`-Flag des API-Keys auf false gesetzt, wodurch dieser API Key nicht mehr genutzt werden kann und nach 30 Tagen gelöscht wird.

F-30 Diese Anforderung wurde erfüllt. In Anhang A Abbildung A.5 ist die API-Key Management Benutzeroberfläche innerhalb der Organisationseinstellungen zu sehen. Hier kann sowohl ein neuer API-Key erstellt, bestehende eingesehen und falls notwendig deaktiviert werden.

F-31 Diese Anforderung wurde erfüllt. Auf der Konfigurationsseite der Integration auf der Benutzeroberfläche wird der Nutzer instruiert, wie er den API-Key sicher auf den jeweiligen Plattformen speichert. Dabei wird ein Link zu der entsprechenden Einstellung des Repositories auf der jeweiligen Plattform bereitgestellt. Dort kann der Nutzer den Key eintragen, so dass er später durch die Integration sicher genutzt werden kann.

8.2 Nicht-funktionale Anforderungen

Allgemein

NF-1 Diese Anforderung wurde erfüllt. Mit Hilfe des Swagger Editors wurde die Konfiguration auf Kompatibilität mit der spezifizierten OpenAPI Version geprüft.

NF-2 Diese Anforderung wurde erfüllt. Da die Github-Action eine JavaScript-Action ist, ist diese mit allen Betriebssystemen kompatibel. Die GitLab-Component funktioniert auf Grundlage von Docker und ist im GitLab-CI/CD-System ebenfalls mit allen Betriebssystem kompatibel. Dabei können sowohl von den Plattform bereitgestellte als auch selbst aufgesetzte Runner genutzt werden.

NF-3 Diese Anforderung wurde erfüllt. Eine README-Datei liegt in den jeweiligen Repositories der GitLab-Component[3] beziehungsweise der GitHub-Action[4]. Darin wird neben einer beispielhaften Konfiguration auch Kontext und Link zu SCA Tool gegeben. Dadurch können Nutzer die durch die CI/CD-Plattformen auf SCA Tool aufmerksam geworden sind auf die Projektwebseite weitergeleitet werden. Zusätzlich wird die Funktionalität der Integration und die möglichen Inputs genau dokumentiert. Auch wird darin erklärt, wie neue Versionen veröffentlicht werden können, um Nutzern die Änderungen zur Verfügung zu stellen.

NF-4 Diese Anforderung wurde erfüllt. Sowohl die Public-API als auch die veröffentlichten CI/CD-Komponenten sind versioniert. Die Versionierung der Komponenten ist dabei durch ein Release auf dem GitHub-Marketplace beziehungsweise im CI/CD-Catalog ermöglicht. In den Endpunkten der Public-API ist eine Versionsnummer im Pfad eingebaut. So kann bei Änderungen, die zu einer Inkompatibilität alter Versionen führen, die Versionsnummer erhöht werden. Auf dem Ser-

[3]https://gitlab.com/scatool/gitlab-job
[4]https://github.com/scatool/github-action

ver kann für eine begrenzte Zeit auch eine alte Version des Backends laufen, die für die alten Schnittstellen eine Warnung ausgibt, dass die Konfiguration auf die neue Version geändert werden soll. Unter den Endpunkten mit den neuen Versionen kann die neue Funktionalität wie geplant genutzt werden. Alternativ kann auch eine Fehlermeldung auf den alten Endpunkten ausgegeben werden, die angibt, dass die alte Version nicht mehr nutzbar ist. Durch diese Versionierung besteht daher die Möglichkeit, je nach Dringlichkeit den Nutzer eine temporäre Weiternutzung alter Integrationsversionen zu ermöglichen oder eine Fehlermeldung auszugeben, die genau beschreibt, was geändert werden muss, um die Integration wieder nutzen zu können.

Benutzerfreundlichkeit

NF-5 Diese Anforderung wurde erfüllt. Die Konfigurationsseiten sind in Anhang A Abbildung A.2 bis Abbildung A.4 zu sehen. Gängige Probleme während der Konfiguration sind fehlerhaft eingegebene Repository-URLs oder fehlende Kenntnisse über die CI/CD-Funktionalität der jeweiligen Plattform. Einerseits werden Eingaben des Nutzers mit Hilfe von Regulären-Ausdrücken auf syntaktische Korrektheit geprüft. Andererseits werden durch externe Links sowohl Hilfestellungen zum Verständnis als auch eine indirekt Überprüfung der URL des angegebenen Repositories zur Verfügung gestellt. Für die Speicherung des API-Keys auf der Plattform wird ein konfigurierter Direktlink zu der Einstellung des Repositories angegeben. Funktioniert der angegebene Link nicht, erhält der Nutzer einen Erklärungstext und die Möglichkeit, die Repository-URL zu ändern. Durch diese Maßnahmen sollen Fehlkonfigurationen verhindert werden und der Nutzer mit den notwendigen Informationen versorgt werden um Probleme während der Konfiguration selbstständig zu lösen.

NF-6 Diese Anforderung wurde erfüllt. Die bereits erwähnten Breadcrumbs stellen genau diese Hierarchie dar. Dabei wird von links nach rechts das aktuelle Projekt, die dazugehörige Ref und die ausgewählte Version angezeigt. Dadurch ist die Hierarchie für den Nutzer verständlich in der Benutzeroberfläche abgebildet.

NF-7 Diese Anforderung wurde erfüllt. In Anhnag A Abbildung A.1 sind im oberen Rand die sogennanten Breadcrumbs zu sehen. In diesen ist das letzte Element die aktuell angezeigte Version. Durch einen Klick auf die aktuelle Version öffnet sich eine Liste an auswählbaren Versionen. Dabei werden die nächsten 5 verfügbaren Versionen für neuere und ältere Versionen angezeigt. Sollte die gewünschte Version außerhalb dieses Bereichs liegen, können zusätzliche geladen werden. Mittels des

zweiten Klicks kann dann die gewünschte Version ausgewählt werden, die darauf hin geladen und angezeigt wird.

NF-8 Diese Anforderung wurde erfüllt. Für erfahrene Nutzer ist nach Eingabe der Repository-URL die Projekt-Id separat angezeigt, die der einzige Parameter ist, der von SCA Tool kopiert werden muss, wenn bereits eine andere funktionierende Integration eingerichtet wurde. Für Nutzer die noch keine Integration erstellt haben, sich aber mit der Nutzung von GitHub-Workflows beziehungsweise GitLab-Pipelines auskennen, kann effizient die Konfiguration erstellt und kopiert werden. Dazu kann der API-Key direkt von der Konfigurationsseite aus erstellt werden und der Nutzer wird angeleitet, wie er diesen sicher speichern kann. Mit der kopierten Konfiguration kann der Nutzer schnell die SCA Tool Integration zu seinem Setup hinzufügen. Für unerfahrene Nutzer gibt es zusätzliche Erklärungen zu den Einstellungen die auf der externen Plattform getroffen werden müssen. Auch kann die Konfiguration als Datei heruntergeladen werden, wodurch diese Datei lediglich in das Repository hochgeladen werden muss. Außerdem wurde ein Link hinzugefügt, der die CI/CD-Funktionalität der entsprechenden Plattform erklärt.

Sicherheit

NF-9 Diese Anforderung wurde erfüllt. Es wurde ein eigener API-Filter erstellt, der Anfragen anhand eines mitgesendeten API-Key filtert. Dieser wurde konfiguriert, dass er lediglich Requests der Public-API überprüft.

NF-10 Diese Anforderung wurde erfüllt. Das Integrationsskript sendet Anfragen mittels des `https`-Protokolls an das System. Zwar sind die Testsysteme nur mit dem `http`-Protokoll kompatibel, das Produktivsystem erfüllt jedoch die Anforderung. Deswegen wird sie als erfüllt betrachtet.

NF-11 Diese Anforderung wurde erfüllt. Wie in Kapitel 7.2 beschrieben, wird der API-Key mit Hilfe des HMAC-SHA-256 gehasht, bevor er in der Datenbank gespeichert wird.

NF-12 Diese Anforderung wurde erfüllt. Es werden lediglich die Dateien hochgeladen, die der Server verarbeiten kann. Die hochgeladenen Metadaten sind auf die zur Sortierung und Gruppierung erstellter Versionen notwendigen Metadaten beschränkt. Der Nutzer kann zusätzlich Dateien ausschließen, die nicht hochgeladen werden sollen. Auch wurde darauf geachtet, dass der Nutzer keine Zugriffstokens mit umfangreichen Berechtigungen erstellen muss. Die Kommentarfunktion der

GitHub Action ist optional und kann den eingeschränkten automatisch von GitHub erstellten Zugriffstoken nutzen. Durch die Konfiguration des Ausführungszeitpunkts der Integration kann auch der Zeitpunkt eines Uploads durch den Nutzer bestimmt werden. Durch diese Maßnahmen hat der Nutzer die volle Kontrolle darüber, welche Daten zu welchem Zeitpunkt mit dem SCA Tool Server geteilt werden und muss unter keinen Umständen einen Zugriffs-Token freigeben.

Codequalität

NF-13 Diese Anforderung wurde erfüllt. Für den Java Code innerhalb des SCA Tool Codes wird Checkstyle genutzt. Für die Webanwendung und den Integrationscode wird Prettier genutzt. Dadurch wird sichergestellt, dass der Code sich an den festgelegten Codestyle hält.

NF-14 Diese Anforderung wurde erfüllt. Von einem zentralen `main`-TypeScript aus, werden die einzelnen Funktionen aufgerufen. Die Funktionen sind in eigene Dateien ausgelagert. Durch diese Modularität kann die genaue Implementierung später einfach ausgetauscht werden, ohne andere Teile des Codes anpassen zu müssen.

NF-15 Diese Anforderung wurde erfüllt. Es wurden zwei seperate API Spezifikationen erstellt. Die Endpunkte die durch die Webanwendung genutzt werden, sind in der `web.yaml` Spezifikation definiert. Die Endpunkte der Integration in `public.yaml`. Jede API folgt somit dem Single Responsibility Prinzip, bei dem eine Unterteilung anhand des Aufgabengebiets erstellt wird. Zusätzlich ist der Controller der Integration in einem separatem Package, so dass auch die Controller entsprechend ihrer Aufgabe getrennt sind.

9 Fazit

Das Ziel dieser Arbeit war es, eine OpenAPI-konforme API für automatisierte CI-Workflows zu entwickeln, um den Prozess der Nutzung des SCA Tool zu automatisieren. Dadurch wurde eine tiefere Integration in die Entwicklungsabläufe von Softwareprojekten ermöglicht.

Die im Rahmen dieser Arbeit entwickelte GitHub-Action sowie die GitLab - Component, erlauben es Entwicklern, automatisiert Abhängigkeiten an SCA Tool zur Analyse zu übermitteln. Dabei behalten sie jederzeit die Kontrolle darüber, welche Daten zu welchem Zeitpunkt hochgeladen werden. Ergänzt durch die Bereitstellung eines direkten Links zu den Ergebnissen eines Uploads wird ein schneller Zugriff auf die Resultate ermöglicht. Dies unterstützt Entwickler dabei, die Qualität ihrer Software kontinuierlich zu überwachen und mit früheren Versionen zu vergleichen.

Durch die Bereitstellung der erstellten Komponenten über den CI/CD-Catalog von GitLab beziehungsweise den GitHub-Marketplace, zusammen mit der im Frontend hinzugefügten Konfigurationsseite zur Erstellung der Integration, wird die Nutzung der vorgestellten Automatisierung erleichtert. Dadurch können selbst unerfahrene Nutzer die Integration in wenigen Schritten einrichten und Versionen hochladen.

Zukünftige Erweiterungen könnten die Unterstützung weiterer CI/CD-Plattformen wie Jenkins oder Azure umfassen. Darüber hinaus wäre es denkbar, nicht nur einen Link zu den Ergebnissen bereitzustellen, sondern auch die Scan-Ergebnisse direkt in das CI/CD-System zurückzuführen. Dies würde eine umfassendere Integration erfordern, die deutlich höhere Berechtigungen und Zugriff benötigt. Die in dieser Arbeit vorgestellte Lösung wurde jedoch in erster Linie für Nutzer konzipiert, die mit minimalem Berechtigungsbedarf und höchster Kontrolle über geteilte Daten vom Einsatz des SCA Tool profitieren möchten.

Anhänge

A Benutzeroberfläche

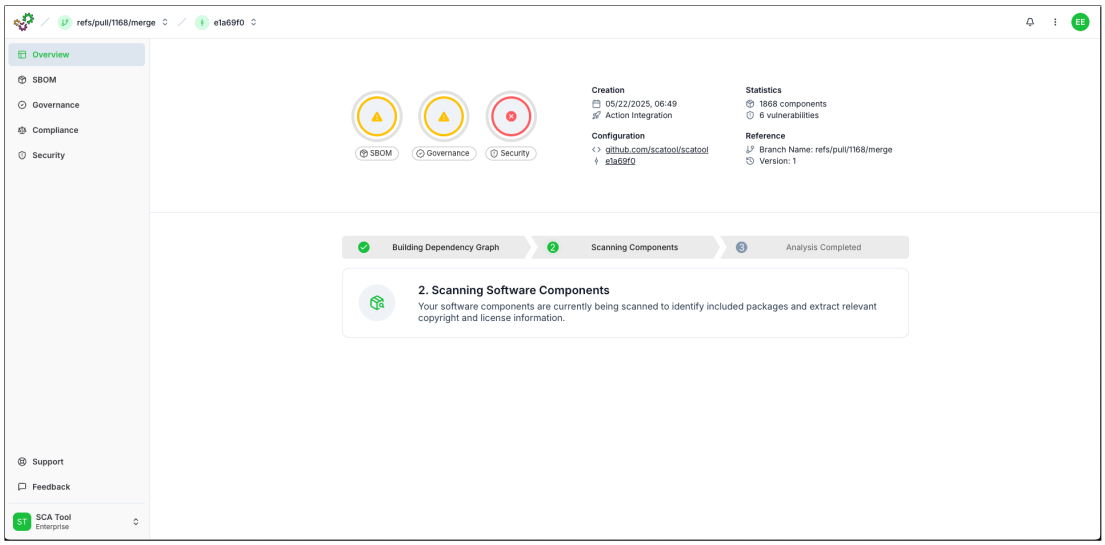


Abbildung A.1: Ergebnisseite eines Scans mit Breadcrumbs zur Navigation von Refs und VerUnits

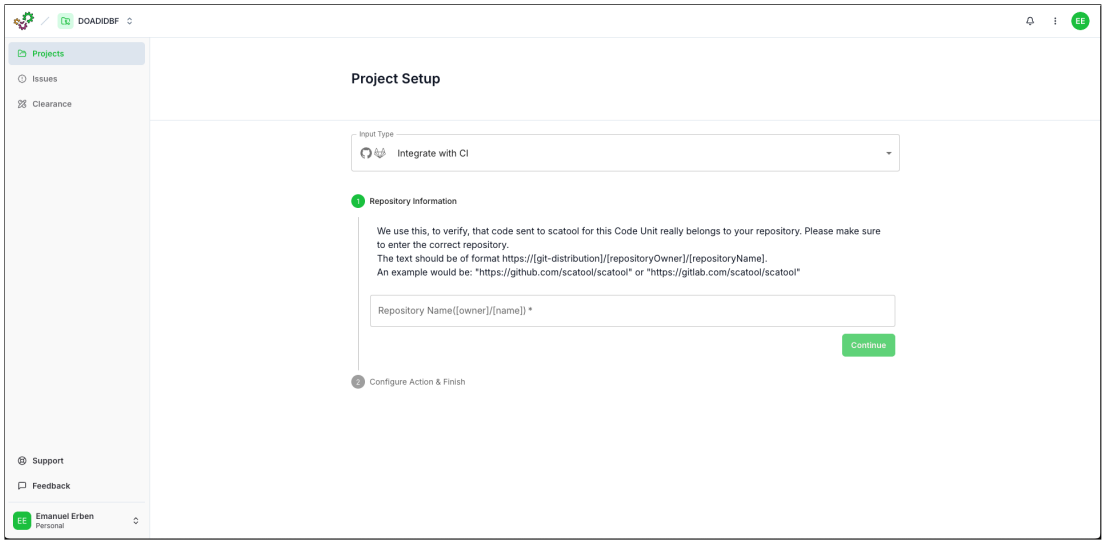


Abbildung A.2: Konfiguration des Projektes als ActionOrigin und Festlegen der Repository-URL

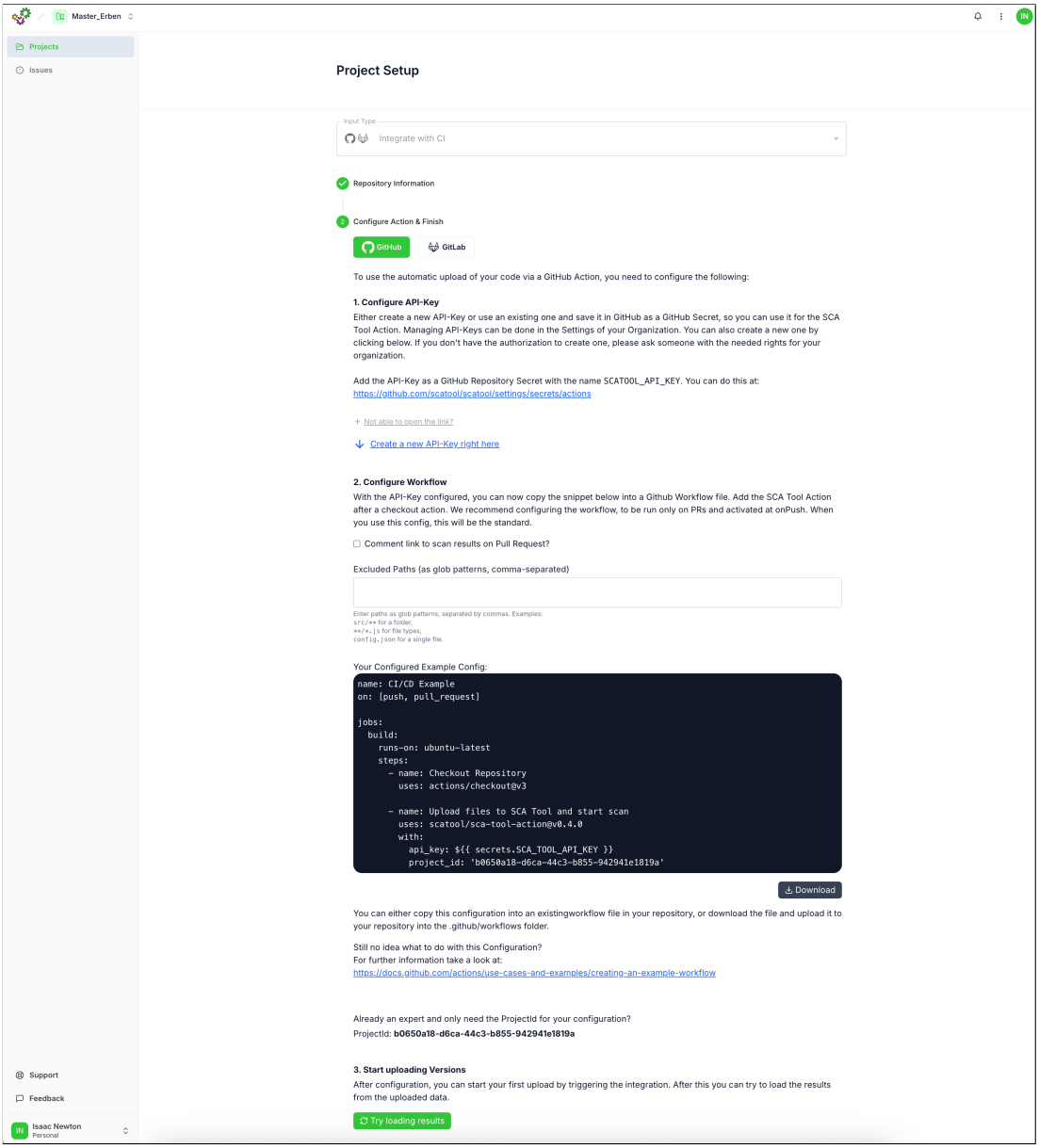


Abbildung A.3: Konfigurationsanleitung GitHub-Action

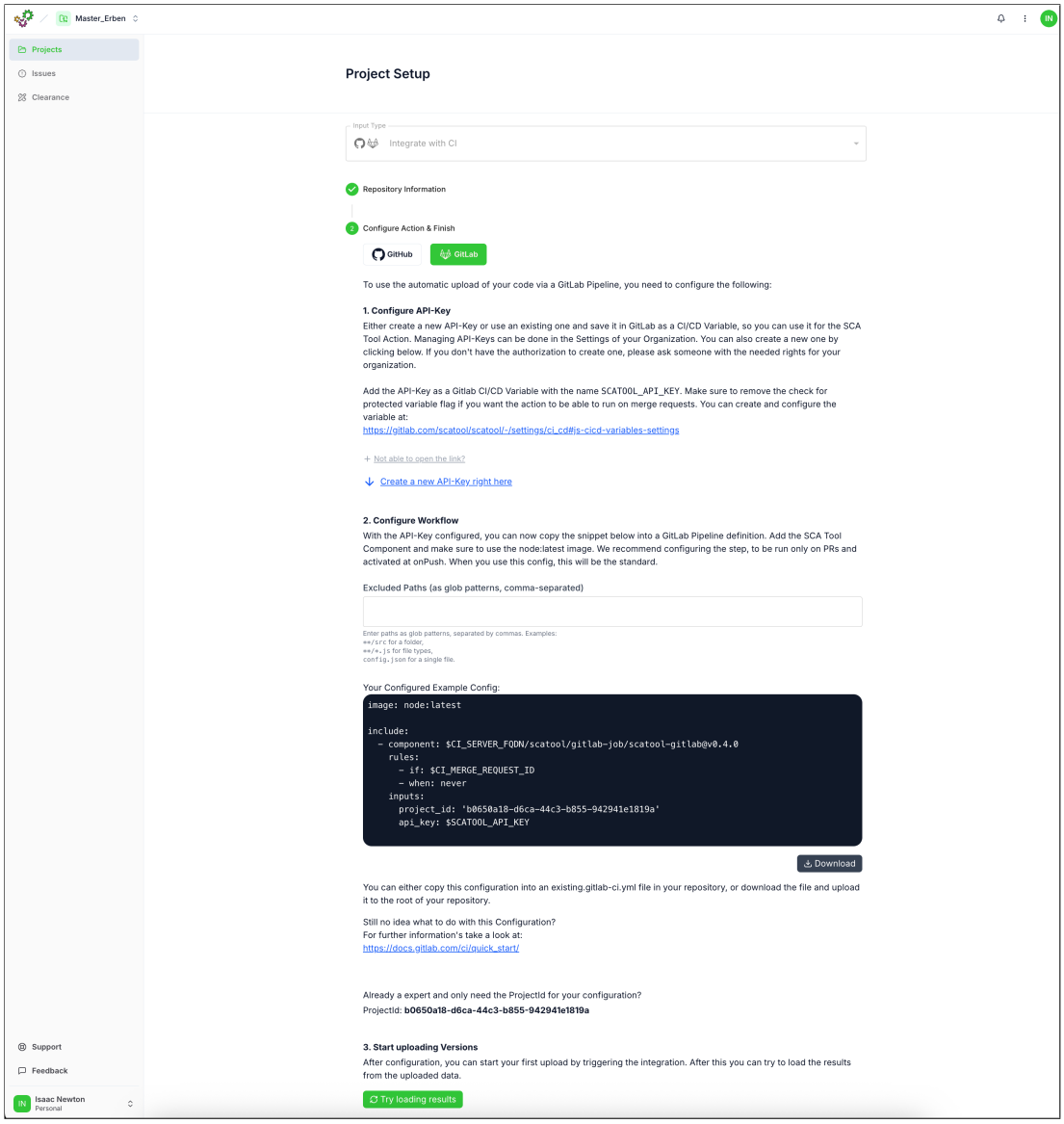


Abbildung A.4: Konfigurationsanleitung GitLab-Component

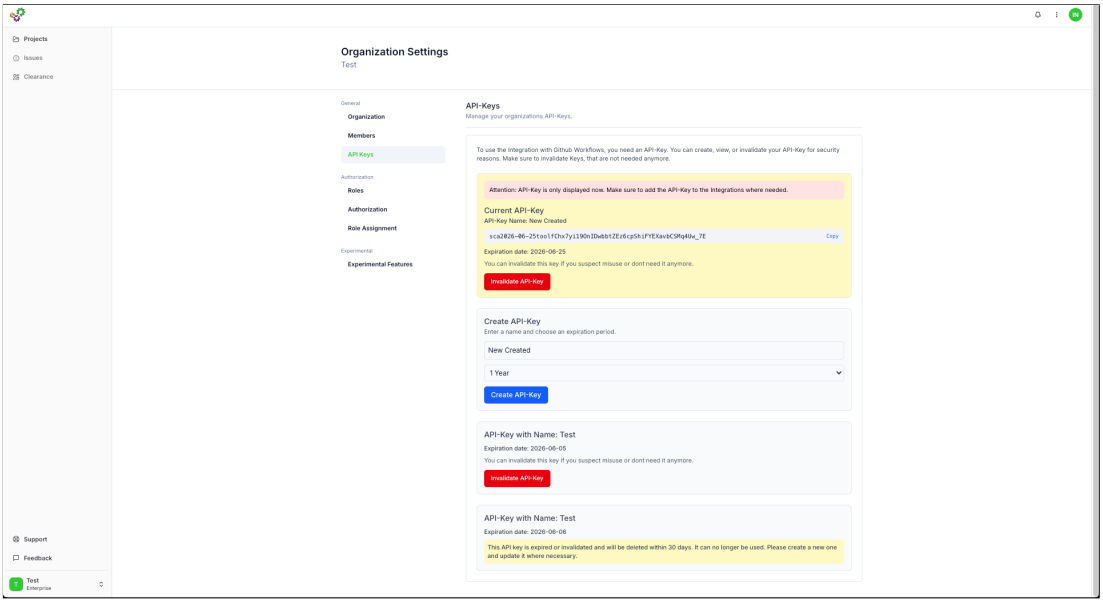
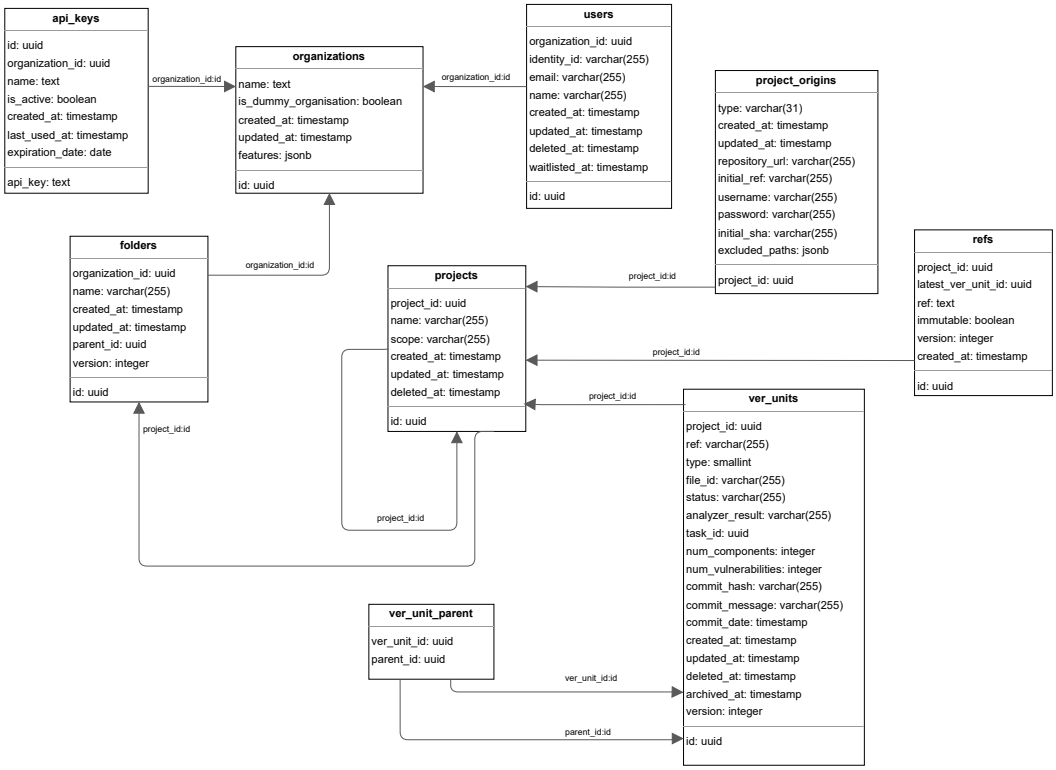


Abbildung A.5: Übersichts- und Managementseite der API-Keys

B Ausschnitt Datenbank



Literaturverzeichnis

Bhorkar, T. (2017). A survey of password attacks and safe hashing algorithms. *04*(12). https://www.irjet.net/archives/V4/i12/IRJET-V4I12284.pdf

Chi, L., & Zhu, X. (2017). Hashing Techniques: A Survey and Taxonomy. *ACM Comput. Surv.*, *50*(1), 11:1–11:36. https://doi.org/10.1145/3047307

Decan, A., Mens, T., Mazrae, P. R., & Golzadeh, M. (2022). On the Use of GitHub Actions in Software Development Repositories [ISSN: 2576-3148]. *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 235–245. https://doi.org/10.1109/ICSME55016.2022.00029

Farrell, S. (2009). API Keys to the Kingdom. *IEEE Internet Computing*, *13*(5), 91–93. https://doi.org/10.1109/MIC.2009.100

GitHub. (2025a). *About custom actions* [GitHub docs]. Verfügbar 31. Mai 2025 unter https://docs.github.com/actions/sharing-automations/creating-actions/about-custom-actions

GitHub. (2025b). *About workflows* [GitHub docs]. Verfügbar 1. Juni 2025 unter https://docs.github.com/actions/writing-workflows/about-workflows

GitHub. (2025c). *Creating a composite action* [GitHub docs]. Verfügbar 1. Juni 2025 unter https://docs.github.com/actions/sharing-automations/creating-actions/creating-a-composite-action

GitHub. (2025d). *Creating a Docker container action* [GitHub Docs]. Verfügbar 1. Juni 2025 unter https://docs.github.com/actions/sharing-automations/creating-actions/creating-a-docker-container-action

GitHub. (2025e). *Creating a JavaScript action* [GitHub docs]. Verfügbar 1. Juni 2025 unter https://docs.github.com/actions/sharing-automations/creating-actions/creating-a-javascript-action

GitHub. (2025f). *Publishing actions in GitHub marketplace* [GitHub docs]. Verfügbar 2. Juni 2025 unter https://docs.github.com/actions/sharing-automations/creating-actions/publishing-actions-in-github-marketplace

GitLab. (2025a). *CI/CD components GitLab Docs*. Verfügbar 17. Mai 2025 unter https://docs.gitlab.com/ci/components/#publish-a-new-release

GitLab. (2025b). *CI/CD Jobs GitLab Docs*. Verfügbar 17. Mai 2025 unter https://docs.gitlab.com/ci/jobs/

GitLab. (2025c). *CI/CD pipelines GitLab Docs*. Verfügbar 17. Mai 2025 unter https://docs.gitlab.com/ci/pipelines/

Guadrsquare. (2025a). *Analysis capabilities | guardsquare*. Verfügbar 5. Mai 2025 unter https://help.guardsquare.com/en/articles/246646-analysis-capabilities

Guadrsquare. (2025b). Integrating AppSweep with Github Projects | Guardsquare Help Center. Verfügbar 5. Mai 2025 unter https://help.guardsquare.com/en/articles/118545-integrating-appsweep-with-github-projects

Gupta, A., Panda, M., & Gupta, A. (2024). Advancing API security: A comprehensive evaluation of authentication mechanisms and their implications for cybersecurity. *International Journal of Global Innovations and Solutions (IJGIS)*. https://doi.org/10.21428/e90189c8.406d2328

Harry Peter. (2025, März). *API-first development: A guide to reusable components for data-centric web apps* [ResearchGate]. Verfügbar 22. Juni 2025 unter https://www.researchgate.net/publication/391010474_API-First_Development_A_Guide_to_Reusable_Components_for_Data-Centric_Web_Apps

Kin Lane & Steef-Jan Wiggers. (2022). A standardized, specification-driven API lifecycle. *InfoQ*. Verfügbar 25. Mai 2025 unter https://www.infoq.com/articles/Standardized-Specification-Driven-API-Lifecycle/

Kornienko, D. V., Mishina, S. V., Shcherbatykh, S. V., & Melnikov, M. O. (2021). Principles of securing RESTful API web services developed with python frameworks [Publisher: IOP Publishing]. *Journal of Physics: Conference Series*, *2094*(3), 032016. https://doi.org/10.1088/1742-6596/2094/3/032016

Manisha Rana, Ayush Panday, Ankit Mishra & Vishal Kandu. (2024, September). *A comprehensive study on the efficacy of JSON web token (JWT) and HMAC SHA-256 algorithm for web application security* [ResearchGate]. Verfügbar 8. Juni 2025 unter https://www.researchgate.net/publication/384037326_International_Journal_on_Recent_and_Innovation_Trends_in_Computing_and_Communication_Enhancing_Data_Security_A_Comprehensive_Study_on_the_Efficacy_of_JSON_Web_Token_JWT_and_HMAC_SHA-256_Algorithm_for_

Maurice Dawson, Darell Norman Burrell, Emad Rahim & Stephen Brewster. (2010, Januar). *Integrating software assurance into the software development life cycle (SDLC)* [ResearchGate]. Verfügbar 9. Juni 2025 unter https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC

Metz, C. (1999). AAA protocols: authentication, authorization, and accounting for the Internet. *IEEE Internet Computing*, *3*(6), 75–79. https://doi.org/10.1109/4236.807015

OpenAPI Initiative. (2024, 24. Oktober). *OpenAPI Specification v3.1.1*. Verfügbar 25. Mai 2025 unter https://spec.openapis.org/oas/v3.1.1.html

Oracle. (2025). *SecureRandom (Java Platform SE 8)*. Verfügbar 27. Juni 2025 unter https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

OSS Review Toolkit. (2025). *OSS review toolkit | OSS review toolkit*. Verfügbar 29. Mai 2025 unter https://oss-review-toolkit.org/ort/docs/intro

OWASP. (2025). *REST Security - OWASP Cheat Sheet Series*. Verfügbar 30. Mai 2025 unter https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#api-keys

Ramachandrappa, N. C. (2024). SOLID Design Principles in Software Engineering. *International Journal of Computer Trends and Technology*, *72*(9), 18–23. https://www.ijcttjournal.org/archives/ijctt-v72i9p104

Singh, A., & Raj, S. (2022). Securing password using dynamic password policy generator algorithm. *Journal of King Saud University - Computer and Information Sciences*, *34*(4), 1357–1361. https://doi.org/10.1016/j.jksuci.2019.06.006

SmartBear. (2020). *State of API 2020 Report*. Verfügbar 20. Juni 2025 unter https://smartbear.com/resources/ebooks/the-state-of-api-2020-report/

Snyk. (2022). *State of Open Source security 2022* [Snyk]. Verfügbar 12. Juli 2025 unter https://snyk.io/de/reports/open-source-security-2022/

Sriramya, P., & Karthika, R. A. (2015). Providing password security by salted password hashing using bcrypt algorithm. *ARPN Journal of Engineering and Applied Sciences*, *10*. https://www.researchgate.net/publication/282690050_Providing_password_security_by_salted_password_hashing_using_Bcrypt_algorithm