Webdienst zur Überwachung von Schwachstellen in Software-Stücklisten (SBOM)

BACHELOR THESIS

Lukas Nehrke

Eingereicht am 4. September 2023

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

Betreuer:
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, 4. September 2023

Lizenz

Erlangen, 4. September 2023

Abstract

The use of open source software (OSS) plays an important role in modern software development. Despite its various benefits, the transparency and popularity of OSS also introduce new security challenges. This has been highlighted recently by the critical Log4Shell vulnerability that jeopardized a large number of systems. To address these issues, it is essential to keep track of used components and their vulnerabilities. However, due to the complexity of software supply chains, automated tools are indispensable. This bachelor's thesis presents 'VulnAware', a web service that detects vulnerabilities in software dependencies. It allows users to upload their Software Bill of Materials (SBOM) to the platform, which then cross-references the included components with selected vulnerability databases. Additionally, it offers continuous monitoring and alerts users when new vulnerabilities have been found. By integrating 'VulnAware' into the development process, security issues can be identified and resolved faster, ensuring safer applications.

Zusammenfassung

Die Verwendung von Open Source Software (OSS) spielt eine wichtige Rolle in der modernen Softwareentwicklung. Trotz der zahlreichen Vorteile führen die Transparenz und Popularität von OSS auch zu neuen Sicherheitsherausforderungen. Dies wurde kürzlich durch die kritische Log4Shell-Schwachstelle hervorgehoben, die eine große Anzahl von Systemen gefährdet hat. Um diesen Problemen entgegenzuwirken, ist es notwendig, verwendete Komponenten und deren Schwachstellen im Auge zu behalten. Aufgrund der Komplexität von Software-Lieferketten sind jedoch automatisierte Werkzeuge unverzichtbar. In dieser Bachelorarbeit wird „VulnAware“ vorgestellt, ein Webdienst, der Schwachstellen in Softwareabhängigkeiten erkennt. Entwickler können ihre Software-Stücklisten (SBOM) auf die Plattform hochladen, welche dann die darin enthaltenen Komponenten mit ausgewählten Schwachstellendatenbanken abgleicht. Darüber hinaus werden Stücklisten kontinuierlich überwacht und Benutzer informiert, wenn neue Schwachstellen gefunden wurden. Durch die Integration von „VulnAware“ in den Entwicklungsprozess können Sicherheitsprobleme schneller erkannt und behoben werden, was zu sichereren Anwendungen führt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

AWS Amazon Web Services

BSI Bundesamt für Sicherheit in der Informationstechnik

CD Continuous Delivery

CI Continuous Integration

CLI Command-line Interface

CPE Common Platform Enumeration

DBMS Database Management System

DDD Domain-Driven Design

GCS Google Cloud Storage

gRPC gRPC Remote Procedure Calls

JPQL Java Persistence Query Language

JSON JavaScript Object Notation

JVM Java Virtual Machine

JWT JSON Web Token

NTIA National Telecommunications and Information Administration

NIST National Institute of Standards and Technology

NVD National Vulnerability Database

ORM Object-Relational Mapping

OSV Open Source Vulnerabilities

OWASP Open Worldwide Application Security Project

PURL Package URL

REST Representational State Transfer

SaaS Software as a Service

SBOM Software Bill of Materials

SPA Single Page Application

SPDX Software Package Data Exchange

SWID Software Identification

UI User Interface

UML Unified Modeling Language

URL Uniform Resource Locator

XML Extensible Markup Language

1 Einleitung

Die Integration von Open Source Software (OSS) ist ein wichtiger Bestandteil moderner Softwareentwicklung. Durch die Offenheit und der starken Verbreitung von OSS können allerdings auch Sicherheitsrisiken entstehen, wie die kürzlich die kritische *Log4Shell*-Sicherheitslücke (CVE-2021-44228) gezeigt hat. Die Schwachstelle in der beliebten Java Logging Bibliothek *log4j* ermöglichte es Angreifern, über eine präparierte Nachricht beliebigen Code auszuführen. Sie wurde vom NIST (2021) und dem BSI (2021) mit der höchsten Risikobewertung eingestuft und betraf weite Teile des Java Ökosystems (Wetter & Ringland, 2021).

Problematisch ist vor allem, wie mit bekannten Schwachstellen umgegangen wird. Um der Ausnutzung von Open Source Schwachstellen entgegenzuwirken, müssen Abhängigkeiten regelmäßig aktualisiert werden. Nach einer empirischen Studie von Kula et al. (2018) aktualisierten allerdings 81.5% der untersuchten Projekte ihre Abhängigkeiten nicht auf die neueste Version. Ein ähnlich schlechtes Bild zeigt auch ein aktueller Bericht von Synopsis, nach welchem in 11% der untersuchten Java-Projekten trotz der großen medialen Aufmerksamkeit immer noch eine verwundbare Version von *log4j* gefunden wurde (Synopsys, 2023).

Die Ursachen dafür sind vielfältig. Bereits die hohe Anzahl an Abhängigkeiten macht Software-Lieferketten komplex und schwierig zu überblicken. Hinzu kommt, dass sich schätzungsweise 78% der Schwachstellen auf die indirekten Abhängigkeiten zurückführen lassen (Tal, 2019). Oftmals wissen Entwickler überhaupt nicht, dass ihre Software verwundbare Komponenten enthält oder sie schätzen die Risiken falsch ein (Kula et al., 2018; Sonatype, 2023). Automatisierte Werkzeuge können jedoch helfen, Schwachstellen gezielter zu erkennen und zu beheben.

Das Ziel dieser Bachelorarbeit ist die Entwicklung von „VulnAware“, ein Webdienst, der Schwachstellen in den Abhängigkeiten von Softwareprodukten erkennt. Dafür lädt der Benutzer die Stückliste seiner Software (SBOM) entweder automatisiert oder manuell über die Weboberfläche hoch. Der Dienst extrahiert anschließend die darin enthaltenen Komponenten und gleicht sie mit Datenbanken über bekannte Schwachstellen ab. Zusätzlich werden hochgeladene Stücklisten konti-

nuierlich überwacht und Benutzer benachrichtigt, wenn sie von neuen Schwachstellen betroffen sind.

Die Arbeit ist in acht Kapitel unterteilt. Nach dieser Einleitung werden im zweiten Kapitel die nötigen Grundlagen erläutert und bereits existierende Lösungen vorgestellt. Im dritten Kapitel werden die detaillierten Anforderungen an die Software festgehalten. Das vierte Kapitel dient als grobe Einführung in die Komponenten des Systems. Im fünften Kapitel werden die wichtigsten Designentscheidungen erläutert, deren Implementierung in Kapitel sechs folgt. Anschließend werden im siebten Kapitel die zuvor festgelegten Anforderungen evaluiert, bevor im achten Kapitel die Ergebnisse der Arbeit zusammengefasst werden.

2 Verwandte Arbeit

Dieses Kapitel legt die Basis für die in Kapitel 3 definierten Anforderungen an die Software. Im ersten Teil des Kapitels werden die nötigen Grundlagen zur Erkennung von Schwachstellen geklärt, bevor im zweiten Teil bereits bestehende Softwarelösungen mit verwandter Funktionalität vorgestellt werden.

2.1 Software-Stücklisten

Eine Software-Stückliste oder -Teileliste (engl. Software Bill of Materials; SBOM) ist eine Datei, in der alle Bestandteile einer Software aufgeführt werden. Dazu gehören nicht nur Informationen über den eigenen Quellcode, sondern auch über alle direkten und indirekten Abhängigkeiten. Insbesondere bei moderner Software machen Open-Source-Komponenten einen großen Teil der Stückliste aus.

Die Nutzung von Software-Stücklisten hat einige wichtige Vorteile. Für diese Arbeit besonders relevant ist das einfache Identifizieren von verwundbaren Abhängigkeiten. Da alle Komponenten in maschinenlesbarer Form aufgelistet werden, können sie von speziellen Werkzeugen effizient mit bekannten Schwachstellen abgeglichen werden. So lassen sich potentielle Sicherheitsrisiken schneller erkennen und beheben.

Sowohl die Software Package Data Exchange® (SPDX®) Spezifikation als auch die CycloneDX-Spezifikation sind weit verbreitete SBOM-Standards. Während es beide erlauben, die Komponenten eines Softwareprodukts zu beschreiben, fokussiert sich SPDX eher auf die Lizenzkonformität, wohingegen sich CycloneDX auf Anwendungsfälle im Sicherheitsbereich konzentriert (OWASP Foundation, n. d.; SPDX Workgroup, 2021). Allerdings bringt auch die kommende dritte Version[1] der SPDX-Spezifikation wichtige Verbesserungen für den Umgang mit Schwachstellen mit sich.

[1] https://github.com/spdx/spdx-3-model

2.2 Identifikation von Software

Um Stücklisten nach bekannten Schwachstellen durchsuchen zu können, ist es eine notwendige Voraussetzung, Software einheitlich über mehrere Plattformen hinweg beschreiben zu können. Im Kontext dieser Arbeit sind zwei Spezifikationen von besonderer Bedeutung.

PURL

Die Package URL (PURL) Spezifikation hat das Ziel, Softwarepakete sicher identifizieren und lokalisieren zu können (Ombredanne, 2023). Eine PURL ist eine kompakte Zeichenkette, die aus sieben Komponenten besteht:

`pkg:type/namespace/name@version?qualifiers#subpath`

Die Bedeutung der Komponenten ist hierbei klar definiert:

1. `scheme`: Konstante Zeichenkette 'pkg'
2. `type`: Typ oder Ökosystem des Pakets[2]
3. `namespace` (optional): Ökosystem-spezifische Gruppierung von Paketen
4. `name`: Name des Pakets
5. `version` (optional): Version des Pakets
6. `qualifiers` (optional): Zusätzliche Informationen zum Paket
7. `subpath` (optional): Pfad innerhalb des Pakets

Für das Erkennen von Schwachstellen ist in der Regel das Tupel aus `type`, `namespace`, `name` und `version` bereits ausreichend. Diese Werte lassen sich zum Beispiel aus den Lock- und Manifestdateien eines Projektes ableiten.
Beispiele für PURLs sind:

- `pkg:maven/log4j/log4j@1.2.17?type=jar`
- `pkg:npm/react@18.2.0`
- `pkg:docker/node@20.5.1-bookworm`

PURLs eignen sich besonders zur Identifikation von (veröffentlichten) Paketen. Möchte man sonstige Komponenten wie Anwendungen, Betriebssysteme oder Hardware beschreiben, ist die Spezifikation aufgrund uneindeutiger Schemata eher ungeeignet.

[2] https://github.com/package-url/purl-spec/blob/master/PURL-TYPES.rst

CPE

Common Platform Enumeration (CPE) ist ein verbreiteter Standard zum Identifizieren von Hardware- und Softwarekomponenten. Im Gegensatz zum dezentralen Ansatz von PURLs werden CPE-Namen in einem zentralen Verzeichnis erfasst, welches vom National Institute of Standards and Technology (NIST) verwaltet wird. CPE-Namen lassen sich ebenfalls als Zeichenketten aus mehreren Komponenten darstellen. Der Aufbau ist in der 'CPE Naming Specification' (Cheikes et al., 2011) festgehalten. Beispiele für CPE-Namen sind:

- `cpe:2.3:a:apache:log4j:*:*:*:*:*:*:*:*`
- `cpe:2.3:h:intel:core_i9-11900k:-:*:*:*:*:*:*:*`
- `cpe:2.3:o:linux:linux_kernel:2.6.0:-:*:*:*:*:*:*`

Im Kontext der Schwachstellenerkennung haben CPE-Namen jedoch einige Nachteile. So sind beispielsweise nicht alle Soft- und Hardwareprodukte im zentralen Verzeichnis gelistet. Auch ist es schwierig bis unmöglich, den korrekten CPE-Namen für ein gegebenes Paket zu finden, was oftmals dazu führt, dass SBOM-Generatoren versuchen, den Namen zu raten. Ein weiteres Problem ist die grobe Granularität. So lässt sich beispielsweise nur das Produkt `log4j` angeben, anstatt eine Unterscheidung zwischen den Modulen `log4j-api` und `log4j-core` zu treffen (Springett, 2022).

2.3 Bestehende Software

Aufgrund der Wichtigkeit des Themas existiert eine Vielzahl an Software mit verwandter Funktionalität. In diesem Abschnitt sollen ausgewählte Projekte mit ihren Stärken und Schwächen vorgestellt werden. Diese dienen als Grundlage für die Anforderungen der zu entwickelnden Software.

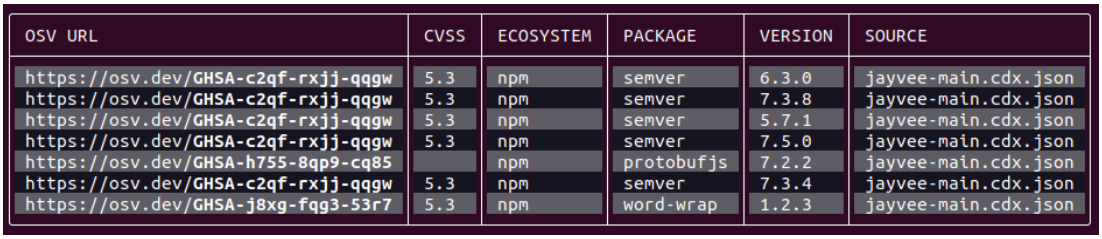
2.3.1 OSV-Scanner

OSV-Scanner ist ein kostenloses Werkzeug von Google, um bekannte Schwachstellen in den Abhängigkeiten von Projekten zu finden (Google, 2023e). Es ist unter der Apache 2.0 Lizenz als Open-Source-Projekt verfügbar.

Beim OSV-Scanner handelt es sich um ein Command-line Interface (CLI) Werkzeug. Die Interaktion mit dem Programm erfolgt also über die Kommandozeile direkt im Quellcode des zu analysierenden Projekts. Eine mögliche Ausgabe ist in Abbildung 2.1 darstellt.

Als Quelle für Schwachstellen wird die Open Source Vulnerabilities (OSV) Datenbank verwendet. Diese aggregiert Sicherheitsempfehlungen von verschiedenen Datenbanken wie der GitHub Advisory Database und stellt sie über eine öffentliche Schnittstelle zur Verfügung.

Das Monitoring von Schwachstellen kann zum Beispiel dadurch realisiert werden, dass der Scanner als Cronjob in einer Continuous Integration (CI) Umgebung ausgeführt wird. Es werden auch GitHub Actions bereitgestellt, um die Integration in GitHub Projekte zu vereinfachen.



OSV URL	CVSS	ECOSYSTEM	PACKAGE	VERSION	SOURCE
https://osv.dev/GHSA-c2qf-rxjj-qqgw	5.3	npm	semver	6.3.0	jayvee-main.cdx.json
https://osv.dev/GHSA-c2qf-rxjj-qqgw	5.3	npm	semver	7.3.8	jayvee-main.cdx.json
https://osv.dev/GHSA-c2qf-rxjj-qqgw	5.3	npm	semver	5.7.1	jayvee-main.cdx.json
https://osv.dev/GHSA-c2qf-rxjj-qqgw	5.3	npm	semver	7.5.0	jayvee-main.cdx.json
https://osv.dev/GHSA-h755-8qp9-cq85		npm	protobufjs	7.2.2	jayvee-main.cdx.json
https://osv.dev/GHSA-c2qf-rxjj-qqgw	5.3	npm	semver	7.3.4	jayvee-main.cdx.json
https://osv.dev/GHSA-j8xg-fqg3-53r7	5.3	npm	word-wrap	1.2.3	jayvee-main.cdx.json

Abbildung 2.1: Standardausgabe des OSV-Scanners auf der Konsole.

OSV-Scanner unterstützt sowohl einzelne Lock- und Manifestdateien als auch SBOM-Dateien im SPDX- und CycloneDX-Format. CPE-Namen werden allerdings aktuell nicht unterstützt.

Eine wichtige Besonderheit des Programms ist die Analyse des Aufrufsgraphen, die mit einem experimentellen Kommandozeilenparameter aktiviert werden kann und zurzeit nur für Go-Projekte verfügbar ist. Wurde eine vermeintliche Schwachstelle gefunden, wird zusätzlich anhand des Quellcodes validiert, ob die betroffenen Symbole überhaupt verwendet werden. So kann die Anzahl falsch-positiver Ergebnisse reduziert werden.

Zusätzliche Schwächen liegen vor allem bei der grafischen Oberfläche und den fehlenden Features zur Verwaltung von Stücklisten und Schwachstellen. Des Weiteren hat OSV-Scanner keine Unterstützung für das Annotieren der Stücklisten. Auch das Monitoring von Schwachstellen über Cronjobs kann bei mehreren Projekten und Versionen zu aufwendig und unübersichtlich werden. Aus diesen Gründen ist möglicherweise eine Kombination aus weiteren Werkzeugen notwendig, um den eigenen Anforderungen gerecht zu werden.

2.3.2 Dependency-Track

Dependency-Track ist eine quelloffene Plattform zur Identifikation und Behebung von Risiken in Software-Lieferketten (Springett, 2023c). Das Projekt ist Teil der OWASP Foundation und steht unter der Apache 2.0 Lizenz.

Ein wichtiger Vorteil von Dependency-Track sind die zahlreichen Integrationen mit anderen Diensten. So lassen sich zum Beispiel mehrere Datenquellen konfigurieren, darunter die GitHub Advisory Database, OSS-Index, die OSV-Datenbank oder die NVD. Die Kombination mehrerer Quellen ermöglicht eine breitere Unterstützung von Komponenten und kann die Anzahl falsch-negativer Ergebnisse verringern.

Component	Version	Vulnerability	CWE	Severity	Analyzer	Attributed On
word-wrap	1.2.3	GITHUB GHSA-j8xg-fqg3-53r7	-	Medium	GITHUB	31 Aug 2023
protobufjs	7.2.2	GITHUB GHSA-h755-8qp9-cq85	CWE-1321	High	GITHUB	31 Aug 2023
semver	6.3.0	GITHUB GHSA-c2qf-rxjj-qqgw	CWE-1333	Medium	GITHUB	31 Aug 2023
semver	7.3.8	GITHUB GHSA-c2qf-rxjj-qqgw	CWE-1333	Medium	GITHUB	31 Aug 2023
semver	5.7.1	GITHUB GHSA-c2qf-rxjj-qqgw	CWE-1333	Medium	GITHUB	31 Aug 2023
semver	7.5.0	GITHUB GHSA-c2qf-rxjj-qqgw	CWE-1333	Medium	GITHUB	31 Aug 2023
semver	7.3.4	GITHUB GHSA-c2qf-rxjj-qqgw	CWE-1333	Medium	GITHUB	31 Aug 2023
protobufjs	7.2.2	NVD CVE-2023-36665	CWE-1321	Critical	OSS Index	31 Aug 2023
ejs	3.1.8	NVD CVE-2023-29827	CWE-74	Critical	OSS Index	31 Aug 2023
word-wrap	1.2.3	NVD CVE-2023-26115	CWE-1333	High	OSS Index	31 Aug 2023

Abbildung 2.2: Schwachstelleninformationen in Dependency-Track.

Das Frontend von Dependency-Track ist besonders umfangreich. Über Diagramme und Tabellen werden schnell die wichtigsten Informationen vermittelt. Stücklisten lassen sich sowohl automatisiert über eine REST-API, als auch manuell importieren. Das Hochladen und Exportieren von SBOM-Dateien ist allerdings ausschließlich im CycloneDX-Format möglich. Das SPDX-Format wird aktuell explizit nicht unterstützt (Springett, 2021).

Um Dependency-Track zu installieren, werden Docker Images und eine vorgefertigte Docker-Compose-Datei bereitgestellt. Eine Software as a Service (SaaS) Lösung wird nicht angeboten.

2.3.3 GitHub Security

GitHub ist ein kommerzieller Dienst zur Versionsverwaltung von Software. Die Plattform integriert dabei auch einige Sicherheitswerkzeuge.

GitHub Dependency Graph

GitHub kann die Lock- und Manifestdateien im Quellcode eines Repositories nutzen, um einen vollständigen Abhängigkeitsgraphen aufzubauen (GitHub, 2023a).

Für Ökosysteme wie Maven/Gradle, die keine Lockdateien verwenden, können (transitive) Abhängigkeiten nachträglich über die *Dependency Submission API* hinzugefügt werden. GitHub stellt dafür unter anderem eine *GitHub Action* für das Hochladen von SPDX-Dateien bereit[3]. Über das UI oder eine REST-Schnittstelle lässt sich der Abhängigkeitsgraph zusätzlich als SPDX-Datei exportieren. Eine starke Einschränkung ist allerdings aktuell, dass der Abhängigkeitsgraph nur für Haupt-Branch aufgestellt und exportiert werden kann.

GitHub Advisory Database

Die *GitHub Advisory Database* ist eine Sammlung von Sicherheitsempfehlungen, die im OSV-Format gespeichert werden und über ein GitHub Repository[4] verfügbar sind (GitHub, 2023b). Neben bekannten Softwareschwachstellen enthält die Datenbank auch Informationen über Pakete mit Schadsoftware. Als Datenquellen dienen einerseits Sicherheitsempfehlungen, die über GitHub eingereicht werden, die National Vulnerability Database (NVD), Datenbanken von Paketmanagern, sowie von der Community über Pull Requests eingereichte Sicherheitsempfehlungen. Um bei der großen Menge an Daten falsch-positive Ergebnisse zu vermeiden, werden die Daten zusätzlich von GitHub überprüft. Aktuell sind von insgesamt 202.384 Einträgen etwa 6% als „von GitHub überprüft“ gekennzeichnet (GitHub, 2023c). Da die Daten frei unter CC-BY-4.0 verfügbar sind und über eine GraphQL-API abgefragt werden können, wird die Datenbank auch von weiteren Trackern als Quelle verwendet.

GitHub Dependabot

Dependabot ist ein Open Source Werkzeug[5], das die Daten aus dem Abhängigkeitsgraphen mit der GitHub Advisory Database abgleicht, um Schwachstellen zu finden (GitHub, 2023d). Sind *Dependabot Alerts* in einem Repository aktiviert, sendet GitHub zusätzlich eine Warnung an die Besitzer, sobald neue Schwachstellen erkannt werden. Über die Schaltfläche „Security“ lassen sich die gefundene Schwachstellen mit weiteren Metadaten einsehen. Sind zusätzlich *Dependabot Security Updates* aktiviert, wird automatisch eine Pull-Request geöffnet, die das betroffene Paket aktualisiert.

[3] https://github.com/marketplace/actions/spdx-dependency-submission-action
[4] https://github.com/github/advisory-database
[5] https://github.com/dependabot

2.4 Zielsetzung

Dieser Abschnitt dient als Brücke zwischen den existierenden Softwarelösungen und den Anforderungen an „VulnAware“. Im Folgenden sollen die Hauptziele vorgestellt werden, die dann in Kapitel 3 zu konkreten Anforderungen präzisiert werden.

2.4.1 Hybride Architektur

Das Erkennen von Schwachstellen ist nicht nur für Organisationen, sondern auch für einzelne Entwickler oder kleinere Teams essenziell. Diese könnten sich aber mit einer komplexen Software wie Dependency-Track zu überfordert fühlen. Aus diesem Grund soll für „VulnAware“ eine hybride Architektur entwickelt werden, die sich sowohl für den SaaS-Betrieb eignet, als auch in einer On-Premises-Umgebung genutzt werden kann. Dies stellt auch besondere Ansprüche an die Portabilität und Skalierbarkeit der Software.

2.4.2 Interoperabilität

Um Interoperabilität mit anderen Systemen und Werkzeugen zu gewährleisten, soll „VulnAware“ eine möglichst breite Unterstützung an Standards bieten. Dazu sollen insbesondere die SBOM-Formate SPDX und CycloneDX gehören, sowie mehrere Identifikationsmechanismen für Komponenten. Darüber hinaus sollte die Architektur so konzipiert sein, dass sie gut um neue Standards oder Versionen erweitert werden kann.

2.4.3 Integration in den Entwicklungsprozess

„VulnAware“ soll besonders einfach in den Softwareentwicklungsprozess integriert werden können. Im Gegensatz zu GitHub soll es möglich sein, Komponenten in sämtlichen Branches, Tags und Pull-Requests eines Repositories zu betrachten, zu überwachen und zu exportieren. Dadurch lassen sich potentielle Sicherheitsprobleme erkennen, noch bevor die betroffenen Komponenten ausgeliefert werden.

2.4.4 Intuitive Benutzeroberfläche

Die UIs von verwandter Software wie Dependency-Track sind oftmals mit Informationen überladen und für Unerfahrene schwer handhabbar. Aus diesem Grund soll das letzte Hauptziel eine besonders simple und übersichtliche Benutzeroberfläche sein.

3 Anforderungen

Vor der Implementierung des Webdienstes werden die Anforderungen an die Software festgehalten. Diese werden zunächst in funktionale und nichtfunktionale Anforderungen unterteilt. Am Ende der Arbeit soll die Umsetzung der Anforderungen evaluiert werden.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben, welches Verhalten der Webdienst erbringen soll. Alle Anforderungen sind gemäß den von SOPHIST GmbH (2016) beschrieben Satzschablonen formuliert und besitzen eine eindeutige Kennung, mit der innerhalb dieser Arbeit auf die Anforderung verwiesen werden kann.

Allgemeine Anforderungen

F-01: Das System muss fähig sein, Benutzer mittels E-Mail-Adresse und Passwort zu authentifizieren.

F-02: Das System muss fähig sein, externe Software wie CLI-Werkzeuge mittels eines API-Keys zu authentifizieren.

F-03: Das System muss Benutzern die Möglichkeit bieten, ein Benutzerkonto mittels E-Mail-Adresse und Passwort zu erstellen.

F-04: Das System muss authentifizierten Benutzern die Möglichkeit bieten, einen API-Key zu (re-)generieren.

F-05: Das System muss authentifizierten Benutzern die Möglichkeit bieten, mehrere Projekte zu verwalten.

F-06: Das System muss Projekterstellern die Möglichkeit bieten, die Mitglieder eines Projekts zu verwalten.

Verwaltung von Stücklisten

F-07: Das System muss Projektmitgliedern die Möglichkeit bieten, eine Stückliste im SPDX-Format hochzuladen.

F-08: Das System sollte Projektmitgliedern die Möglichkeit bieten, eine Stückliste im CycloneDX-Format hochzuladen.

F-09: Das System muss Projektmitgliedern die Möglichkeit bieten, mehrere Stücklisten pro Projekt zu verwalten.

F-10: Falls ein Benutzer eine SBOM-Datei mit einer Version hochlädt, die bereits existiert, muss das System die alte Stückliste durch die neue ersetzen.

F-11: Das System muss fähig sein, die Komponenten in Stücklisten mittels dem PackageURL-Schema, CPE-Namen und SWID-Tags zu identifizieren.

F-12: Das System sollte fähig sein, die Komponenten der Stückliste mit zusätzlichen Informationen aus Package Registries zu annotieren.

F-13: Das System muss Projektmitgliedern die Möglichkeit bieten, die Komponenten der Stückliste zu modifizieren.

F-14: Die Benutzeroberfläche muss fähig sein, die Komponenten der Stückliste tabellarisch anzuzeigen.

Schwachstellenanalyse und -monitoring

F-15: Sobald eine Stückliste hochgeladen wird, muss das System die darin enthaltenen Komponenten auf Basis von PURLs und CPE-Namen mit ausgewählten Schwachstellendatenbanken abgleichen.

F-16: Sobald neue Schwachstellen bekannt werden, muss das System bereits hochgeladene Stücklisten neu abgleichen.

F-17: Das System muss Projektmitgliedern die Möglichkeit bieten, gefundene Schwachstellen mit einem Status und Kommentaren zu versehen.

F-18: Das System muss Projektmitgliedern die Möglichkeit bieten, gefundene Schwachstellen zu ignorieren.

F-19: Das Benutzeroberfläche muss fähig sein, Beschreibungen, Risiken und Handlungsempfehlungen für gefundene Schwachstellen anzuzeigen.

F-20: Das Benutzeroberfläche muss Projektmitgliedern ermöglichen, gefundene Schwachstellen nach Status und Risiko zu sortieren.

Export der Komponenten

F-21: Das System muss Projektmitgliedern die Möglichkeit bieten, die Stückliste als SBOM-Datei im SPDX-Format herunterzuladen.

F-22: Das System sollte Projektmitgliedern die Möglichkeit bieten, die Stückliste als SBOM-Datei im CycloneDX-Format herunterzuladen.

F-23: Das System sollte Projektmitgliedern die Möglichkeit bieten, die Stückliste als CSV-Datei herunterzuladen.

F-24: Das System muss Projektmitgliedern die Möglichkeit bieten, die mit Schwachstelleninformationen annotierte SBOM-Datei im SPDX-Format herunterzuladen.

F-25: Das System muss Projektmitgliedern die Möglichkeit bieten, die mit Schwachstelleninformationen annotierte SBOM-Datei im CycloneDX-Format herunterzuladen.

Benachrichtigungen

F-26: Sobald neue Schwachstellen in Komponenten der Stückliste erkannt werden, muss das System dem Benutzer eine Benachrichtigung mit Beschreibung, Risiko und Handlungsempfehlung senden.

F-27: Die Benutzeroberfläche muss Benutzern die Möglichkeit bieten, Benachrichtigungen zu verwalten.

F-28: Das System sollte Benutzern die Möglichkeit bieten, sich Benachrichtigungen per E-Mail zusenden zu lassen.

Automatisierung

F-29: Das System sollte einen API-Endpunkt anbieten, über den eine SBOM-Datei hochgeladen werden kann.

F-30: Das System sollte einen API-Endpunkt anbieten, über den eine SBOM-Datei hochgeladen werden kann und eine Liste von Schwachstellen zurückgegeben wird.

3.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen stellen Randbedingungen an die zuvor beschriebene Funktionalität der Software. Diese sind im Folgenden nach den Hauptmerkmalen der ISO/IEC 25010 Spezifikation untergliedert.

Effizienz

NF-01: Die Schwachstellenanalyse von Komponenten einer Stückliste muss asynchron erfolgen.

NF-02: Generierte bzw. annotierte SBOM-Dateien müssen mindestens 7 Tage zwischengespeichert werden.

NF-03: Die Zeit zwischen den Bekanntwerden einer Schwachstelle und der Benachrichtigung des Benutzers sollte höchstens 120 Minuten sein.

NF-04: Beim Monitoring sollten nur die Komponenten betrachtet werden, die von neuen Schwachstellen betroffen sind.

Kompatibilität

NF-05: Zu den unterstützten SBOM-Formaten muss das SPDX Format v2.3 in den Datenformaten Tag-Value, JSON und XML gehören.

NF-06: Zu den unterstützten SBOM-Formaten sollte das CycloneDX Format v1.4 in dem Datenformat JSON gehören.

Portabilität

NF-07: Die Software sollte unabhängig von einem spezifischen DBMS (z.B. PostgreSQL, MySQL) sein.

NF-08: Die Architektur sollte unabhängig von einem spezifischen Cloud-Anbieter (z.B. Google Cloud, AWS) sein.

Sicherheit

NF-09: Das System muss sicherstellen, dass nur authentifizierte Benutzer auf den Dienst zugreifen können.

NF-10: Das System muss sicherstellen, dass nur Projektmitglieder auf ein Projekt zugreifen können.

Benutzbarkeit

NF-11: Die Benutzeroberfläche muss den Material Design Richtlinien folgen.

NF-12: Die Benutzeroberfläche muss für alle Bildschirmgrößen optimiert sein.

4 Architektur

Dieses Kapitel dient als grobe Einführung in den Aufbau der Software, bevor in den nächsten Kapiteln detailliert auf die Funktionalitäten eingegangen wird. Es werden die einzelnen Dienste des Systems vorgestellt, auf welchen Technologien diese basieren und wie sie miteinander agieren.

Abschnitt 4.1 gibt zuerst einen Überblick über die Komponenten, bevor in Abschnitt 4.2 die Komponenten im Detail betrachtet werden. Abschnitt 4.3 stellt daraufhin die externen Dienste vor, aus denen die Anwendung ihre Daten bezieht.

4.1 Überblick

Die Komponenten der Architektur sind grafisch in Abbildung 4.1 veranschaulicht. Es handelt sich um eine klassische Client-Server-Architektur, in der die Webanwendung und die GitHub-Action unterschiedliche Arten von Clients bilden. Beide interagieren mit den REST-Schnittstellen des Servers, der den wichtigsten Teil des Backends bildet.

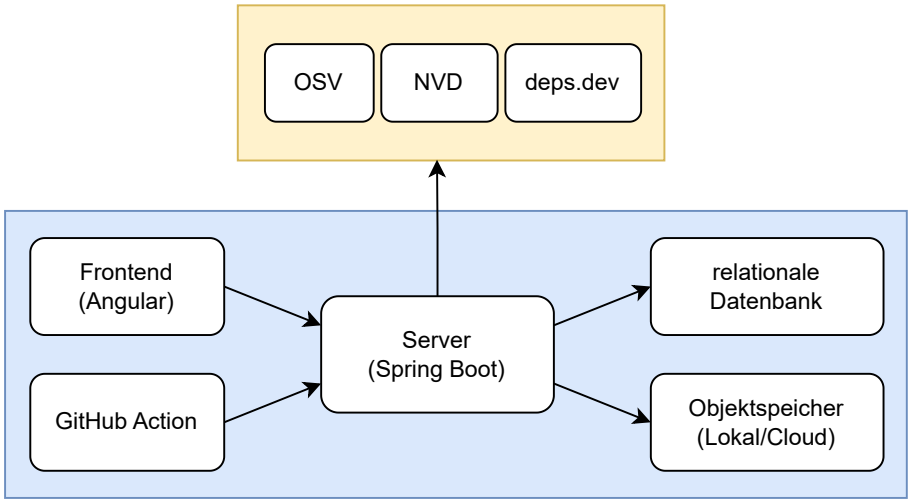


Abbildung 4.1: Illustration der verwendeten Softwarearchitektur.

Zum Backend gehören auch eine relationale Datenbank sowie ein optionaler Objektspeicher zum Verwalten großer, seltener benötigter Daten. Zusätzlich wird auf eine Reihe externen Datenquellen zugegriffen, die in der Abbildung gelb umrandet dargestellt werden.

4.2 Komponenten und Technologien

In diesem Abschnitt werden die Komponenten im Einzelnen betrachtet. Außerdem werden die verwendete Technologien vorgestellt, die aufgrund ihrer Eigenschaften nicht nur für die Implementierung relevant sind, sondern auch auf das Design der Software Einfluss nehmen.

4.2.1 Backend

Das Backend ist für den Großteil der Datenverarbeitung verantwortlich. Wichtige Aufgaben sind unter anderem das Analysieren von SBOM-Dateien, der Zugriff auf interne und externe Datenquellen, sowie das Durchführen des Schwachstellenabgleichs. Zur Bewältigung dieser Aufgaben wird das *Spring Framework* verwendet, ein Open-Source-Framework von VMware Tanzu für die Entwicklung von Anwendungen auf Basis der JVM-Plattform (VMware, 2023a). Spring stellt eine Reihe an vorgefertigten Lösungen zu Verfügung, um die Entwicklung zu vereinfachen und zu beschleunigen. Besonders wichtig ist das Modul *Spring Data JPA*, welches zusammen mit *Hibernate* die Interaktion mit der relationalen Datenbank übernimmt.

Um gesamte Dateien zu Speichern steht zusätzlich zur Datenbank noch einen Objektspeicher (*object storage*) zur Verfügung. Um Portabilität zu gewährleisten, wurden unterschiedliche Implementierungen zur Speicherung von Dateien umgesetzt, einschließlich der Speicherung im lokalen Dateisystem.

4.2.2 Frontend

Die Weboberfläche ist die zentrale Schnittstelle zwischen dem Benutzer und dem System. Sie ermöglicht es dem Benutzer, die Stückliste seiner Produkte hochzuladen, Komponenten und Schwachstellen einzusehen, diese zu verwalten und zu exportieren. Für die Entwicklung der Anwendung wird Angular (Google, 2023b) verwendet, ein Framework für die Entwicklung von Single Page Applications (SPAs). Eine SPA ist eine moderne Webanwendung, bei der beim Navigieren zwischen Seiten keine neue Anfrage an den Server gesendet, sondern die aktuelle Seite mit neuen Inhalten überschrieben wird (Mozilla Developer Network, 2023). So wird eine bessere Performance und Entkopplung zum Server ermöglicht. Als Programmiersprache wird TypeScript verwendet, eine Erweiterung von JavaScript mit statischer Typisierung.

4.2.3 GitHub Action

GitHub Actions ist die Continuous Integration (CI) und Continuous Delivery (CD) Lösung, die in die Softwareplattform GitHub integriert ist. Eine „action“ ist eine portable Anwendung, oftmals nur kleinere Skripte, die in der CI-Umgebung ausgeführt werden und mit einem GitHub Repository interagieren können (GitHub, 2023e).

Die GitHub Action in dieser konkreten Architektur hat die Aufgabe, eine Stückliste über die Entwickler-Schnittstellen (F-29, F-30) automatisch hochzuladen. Damit entfällt für den Benutzer das manuelle Hochladen der Stückliste über die Webanwendung und ermöglicht eine bessere Integration in den Entwicklungsprozess seiner Software.

4.3 Externe Datenquellen

Die Anwendung ist von mehreren externen Diensten abhängig, die zwei verschiedene Arten von Daten bereitstellen:

1. **Sicherheitsempfehlungen**: Informationen über bekannte Schwachstellen, welche Produkte und Versionen betroffen sind, sowie weitere Metadaten wie Beschreibung, Risiko oder Handlungsanweisungen,
2. **Paket-Informationen**: Informationen über Softwarepakete, wie zum Beispiel die aktuellste Version, dazugehörige Websites oder Lizenzen.

Bei der Wahl der Datenquellen wurde darauf geachtet, dass die Daten frei verfügbar sind. Insbesondere ist auch keine Registrierung bei den Diensten nötig, um auf die Daten zugreifen zu können. Somit entstehen bei der Verwendung keine Kosten und die Software ist nach der Installation sofort einsatzbereit.

4.3.1 OSV-Datenbank

Open Source Vulnerabilities (OSV) ist ein Ökosystem von Google, um einerseits Nutzern von Open Source Software zu helfen, Schwachstellen zu finden und zu beheben, und es andererseits Entwicklern von Open Source Software zu erleichtern, Schwachstellen zu melden (Chang & Lewandowski, 2021). Es besteht zurzeit aus drei Projekten:

- dem **OpenSSF/OSV-Schema**[1], ein einheitliches Format, um Schwachstellen über verschiedene Ökosysteme und Datenbanken hinweg beschreiben zu können,

[1] https://ossf.github.io/osv-schema/

- der OSV-Datenbank bzw. **OSV.dev**, eine Schwachstellendatenbank, die Informationen über Schwachstellen aus mehreren Quellen aggregiert und sie über eine API im OSV-Format bereitstellt,
- und **OSV-scanner**[2], einer Anwendung, die auf Basis der OSV-Datenbank Schwachstellen in Softwareprodukten findet kann.

In dieser Arbeit dient die OSV-Datenbank als primäre Datenquelle für Sicherheitsempfehlungen. Anhand dieser Datenbank werden Komponenten abgeglichen, die eine Package URL definiert haben.

4.3.2 NVD

Die National Vulnerability Database (NVD) ist eine Datenbank des NIST, einer Bundesbehörde der Vereinigten Staaten, die Informationen über Schwachstellen in Software- und Hardwareprodukten sammelt und bereitstellt (Byers et al., 2022). Mit über über 200.000 bekannten Schwachstellen, die älteste aus dem Jahr 1999, zählt die NVD zu einer der umfangreichsten Quellen.

Aufgrund des speziellen Schemas der NVD nutzt der Webdienst diese Daten für den Abgleich von Komponenten, die einen CPE-Namen definiert haben. Dies führt zu einer breiteren Unterstützung von Komponenten, insbesondere ermöglicht es den Abgleich von Anwendungen, Betriebssystemen und Hardwaregeräten.

4.3.3 Open Source Insights

Open Source Insights ist ein weiteres Projekt von Google, um Entwicklern zu helfen, die Struktur und Sicherheit von Open Source Softwareprodukten besser zu verstehen (Google, 2023a). Dazu indexiert der Dienst Software aus verschiedenen Paketregistern, stellt einen vollständigen Abhängigkeitsgraphen auf und annotiert diese mit Metadaten, wie zum Beispiel Popularität, Lizenzen oder Schwachstellen. Die Daten sind sowohl über die eigene Website `deps.dev` als auch über eine offene REST- und gRPC-Schnittstelle abrufbar. Zusätzlich wird ein öffentliches BigQuery-Dataset[3] zur Verfügung gestellt.

Der Webdienst nutzt diese REST-Schnittstelle, um die Komponenten der Stückliste mit Lizenzinformationen und der aktuellsten Version zu ergänzen. Somit kann der Benutzer informiert werden, wenn er eine veraltete Komponente verwendet.

[2] https://github.com/google/osv-scanner
[3] https://cloud.google.com/bigquery/public-data

5 Design

Dieses Kapitel stellt die wichtigsten Designentscheidungen im Kontext der vorher festgelegten Anforderungen vor. Die konkrete Implementierung der vorgestellten Algorithmen und Konzepte wird im darauf folgenden Kapitel 6 dargestellt.

5.1 Entitäten und Beziehungen

Abbildung 5.1 zeigt die Beziehungen zwischen den wichtigsten Entitätstypen der Software. Ein vollständiges Klassendiagramm findet sich in Anhang A.

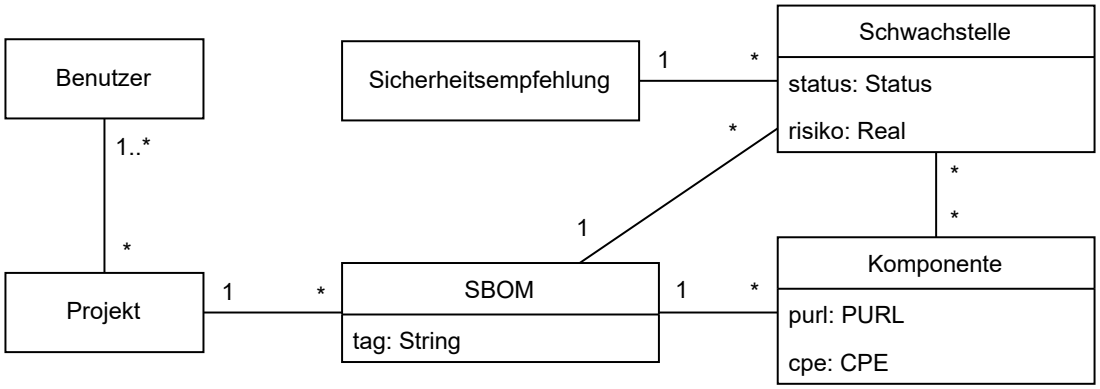


Abbildung 5.1: Übersicht der wichtigsten Entitätstypen.

Ein *Projekt* repräsentiert ein Softwareprodukt in der echten Welt. Der Benutzer kann dabei entschieden, wie breit er sein Produkt definiert. In einer Microservice-Architektur beispielsweise könnte ein Projekt sowohl ein einzelner Dienst als auch das gesamte System darstellen.

Produkte können in vielen Variationen auftreten, weswegen auch ein Projekt mehrere *SBOM* Entitäten verwalten kann. Unterschieden werden diese durch eine projektweit eindeutige Kennung, die im Folgenden *Tag* genannt wird. In der Regel handelt es sich dabei um die Version des Produkts, denkbar sind aber auch zum Beispiel Git-Branches oder einzelne Commits.

Als *Komponenten* sollen die direkten und indirekten Abhängigkeiten eines Produkts bezeichnet werden, die in der SBOM enthalten sind. Dabei handelt es sich meistens, aber nicht ausschließlich, um Bibliotheken oder Frameworks. Das CycloneDX Schema v1.5 führt beispielsweise auch Betriebssysteme, Hardwaregeräte oder Machine-Learning-Modelle als Typen von Komponenten auf (OWASP Foundation, 2023b).

Eine *Schwachstelle* ist die Beziehung zwischen einer Sicherheitsempfehlung bzw. einem Eintrag in einer Schwachstellendatenbank und einer SBOM. Typischerweise betrifft eine Schwachstelle mindestens eine Komponente. Schwachstellen die keine Komponenten betreffen, werden als „erledigt“ gekennzeichnet (siehe auch Abschnitt 5.2.3).

5.2 Analysieren von Stücklisten

Nachdem eine SBOM-Datei zum Server hochgeladen wurde, ist der erste Schritt diese zu analysieren (*parsing*). Dabei wird das externe Format (SPDX, CycloneDX) in die internen Datenstrukturen überführt und anschließend in der Datenbank gespeichert.

5.2.1 Extrahieren der Komponenten

Das Speichern der Stückliste ist nicht einfach umsetzbar. Zum einen handelt es sich um große, verschachtelte Strukturen, deren Konvertierung in ein relationales Datenbankschema besonders aufwendig ist. Ein weiteres Problem ist, dass das Schema gut an zukünftige Versionen und Standards anpassbar sein muss.

Aus diesen Gründen soll in dieser Arbeit ein Ansatz verfolgt werden, bei der nur die nötigsten Informationen aus der Datei extrahiert und in der Datenbank gespeichert werden. Dies schließt insbesondere die Identifikationsmerkmale von Komponenten mit ein (PURL und CPE), sowie die Informationen über die Abhängigkeiten zwischen den Komponenten. Das SPDX- und CycloneDX-Format definieren diese in unterschiedlicher Weise.

SPDX

Die SPDX Spezifikation (v2.3) definiert in Anhang F, wie Pakete identifiziert werden können (The Linux Foundation, 2022). Unter anderem erlaubt sie das Annotieren von Paketen mit PURLs (F.3.5) und CPE-Namen (F.2.1 und F.2.2). Hat eine Komponente mehrere PURLs oder CPE-Namen, soll hier aus Einfachheit nur der erste Wert betrachtet werden.

Abhängigkeiten zwischen zwei Paketen werden über *relationships* (Clause 11) definiert. Je nachdem ob es sich beispielsweise um eine Build- oder Laufzeitabhän-

```
"packages": [{
  "SPDXID" : "SPDXRef-Package1",
  "name": "Jena",
  "externalRefs" : [{
    "referenceCategory" : "PACKAGE-MANAGER",
    "referenceLocator" : "pkg:maven/org.apache.jena/apache-jena@3.12.0",
    "referenceType" : "purl"
  }],
}, {
  "SPDXID" : "SPDXRef-Package2",
  "name" : "Spring",
  "externalRefs" : [{
    "referenceCategory" : "SECURITY",
    "referenceLocator" : "cpe:2.3:a:pivotal_software:spring_framework:4.1.0:*:*:*:*:*:*:*",
    "referenceType" : "cpe23Type"
  }],
}],
"relationships" : [{
  "spdxElementId" : "SPDXRef-Package2",
  "relationshipType" : "DEPENDENCY_OF",
  "relatedSpdxElement" : "SPDXRef-Package1"
}, {
  "spdxElementId" : "SPDXRef-Package2",
  "relationshipType" : "TEST_DEPENDENCY_OF",
  "relatedSpdxElement" : "SPDXRef-Package1"
}]
```

Abbildung 5.2: Ausschnitt einer SPDX-BOM (JSON).

gigkeit handelt, lässt sich ein anderer Typ spezifizieren. Nach den NTIA Mindestanforderungen für SPDX-Dokumente sollen Abhängigkeiten zwischen Paketen allerdings mittels dem Beziehungstyp `CONTAINS` ausgedrückt werden (Winslow, 2023). Ein Beispiel für eine BOM im SPDX-Format ist in Abbildung 5.2 dargestellt.

CycloneDX

Das CycloneDX Schema (v1.5) ist in diesem Kontext wesentlich simpler und spezifiziert für PURL und CPE eigene Felder (OWASP Foundation, 2023c). Auch gibt es kein allgemeines Konzept von Beziehungen, sondern Abhängigkeiten werden unter dem Feld `dependencies` aufgelistet (siehe Abbildung 5.3).

5.2.2 Verlustfreie Speicherung

Damit die Daten, die nicht in der Datenbank gespeichert werden, nicht verloren gehen, wird die gesamte SBOM-Datei zusätzlich auf dem Objektspeicher abgelegt. Das ist insbesondere für die Annotation der Dateien relevant, die in den Anforderungen F-24 und F-25 festgehalten worden sind (vgl. auch Abschnitt 5.4).

```
"components": [{
  "bom-ref": "CDXRef-Package1",
  "name": "Jena",
  "type": "framework",
  "purl": "pkg:maven/org.apache.jena/apache-jena@3.12.0"
}, {
  "bom-ref": "CDXRef-Package2",
  "name": "Spring",
  "type": "framework",
  "cpe": "cpe:2.3:a:pivotal_software:spring_framework:4.1.0:*:*:*:*:*:*:*"
}],
"dependencies": [{
  "ref": "CDXRef-Package2",
  "dependsOn": [
    "CDXRef-Package1"
  ]
}]
```

Abbildung 5.3: Ausschnitt einer CycloneDX-BOM (JSON).

5.2.3 Versionierung

Versionierung soll den Prozess beschreiben, der ausgelöst wird, wenn eine SBOM-Datei mit einem Tag hochgeladen wird, im System aber bereits eine Datenstruktur mit demselben Tag existiert. Der Prozess resultiert aus der Anforderung F-10 und ist in Abbildung 5.4 dargestellt.

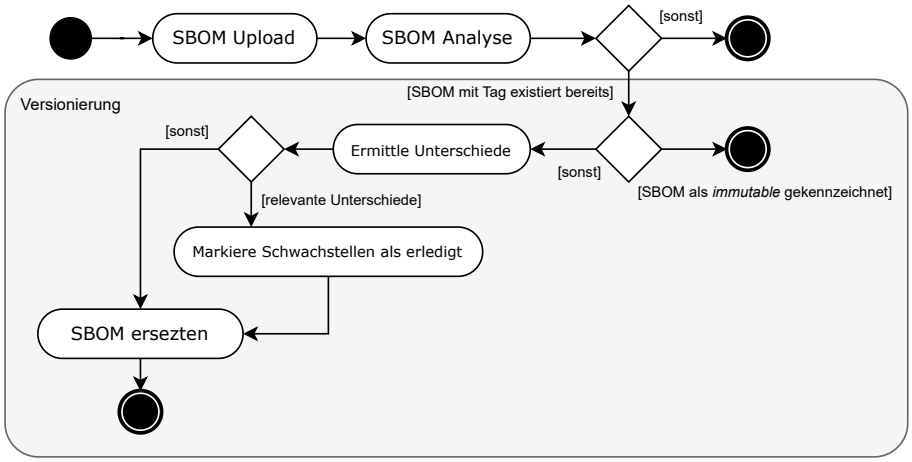


Abbildung 5.4: UML-Aktivitätsdiagramm für den Versionierungsprozess.

Nachdem die Datei hochgeladen und analysiert wurde, werden die Komponenten beider Stücklisten miteinander verglichen. Es wird ermittelt, welche Komponenten hinzugefügt wurden, welche entfernt wurden und welche Versionen sich

geändert haben. Die neue Stückliste ersetzt dabei vollständig die alte Stückliste. Anschließend werden die neuen Komponenten auf Schwachstellen abgeglichen. Schwachstellen, die nicht mehr auf die neue Stückliste zutreffen, weil das betroffene Paket entfernt oder aktualisiert wurde, bekommen automatisch den Status *erledigt*.

Versionierung ist nur sinnvoll, wenn es sich bei dem Tag keine Version, sondern zum Beispiel um einen Git-Branch handelt. Ein SBOM kann als *immutable* gekennzeichnet werden, damit das erneute Hochladen stattdessen zu einem Fehler führt.

5.3 Schwachstellenabgleich

Das Erkennen von Schwachstellen soll funktionieren, indem für die identifizierenden Merkmale der Komponenten einer Stückliste geprüft wird, ob sie in einem Eintrag von ausgewählten Schwachstellendatenbanken vorkommen.

Wie bereits in Abschnitt 4.3 erwähnt, sollen sowohl die OSV-Datenbank, als auch die NVD als Quelle für Sicherheitsempfehlungen genutzt werden. Dabei werden die Schwachstellendaten der OSV-Datenbank für den Abgleich mit Komponenten verwendet, die eine PURL definiert haben, während die Daten der NVD für den Abgleich mit Komponenten verwendet werden, die einen CPE-Namen definiert haben. Hat eine Komponente weder PURL noch CPE-Name definiert, kann sie nicht nach Schwachstellen untersucht werden.

Das Konvertieren einer CPE in eine PURL oder umgekehrt ist im Allgemeinen nicht möglich und erhöht das Risiko von falschen Ergebnissen. Es gibt dennoch Projekte, die versuchen, die Formate aufeinander abzubilden (SCANOSS, 2023).

5.3.1 Importieren der Daten

Es gibt grundsätzlich zwei Verfahren, wie der Zugriff auf diese Datenbanken erfolgen kann. In einem einfachen Ansatz werden die Komponenten der SBOM direkt an die Schnittstelle der Datenbank übergeben. Hier übernimmt der externe Dienst den Abgleich von Komponenten und Schwachstellen. Diese Methode wird von den meisten Schwachstellen Scannern verwendet, die lokal, zum Beispiel in der Befehlszeile oder in einer CI-Umgebung, ausgeführt werden. Für einen Webdienst kann dieses Verfahren allerdings zu Problemen führen:

- **Abhängigkeit**: Der Webdienst macht sich stark von der Datenbank abhängig. Ein Ausfall der Datenbank führt auch zu einem Ausfall des Webdienstes.
- **Performance**: Besonders bei großen Stücklisten können sich die Netzwerklatenzen negativ auf die Performance des Systems auswirken.

- **Sicherheit und Datenschutz**: Ist der externe Dienst nicht vertrauenswürdig, könnte das Weiterleiten der Stücklisten ein potenzielles Sicherheitsrisiko darstellen.
- **Ineffizientes Tracking**: Es ist kein inkrementeller Abgleich möglich, da keine Informationen über neue Schwachstellen verfügbar sind.

Aus diesen Gründen wurde in dieser Arbeit eine Methode gewählt, bei welcher der Webdienst eine lokale Kopie der Datenbanken verwaltet und den Abgleich selber durchführt. Dies mildert zwar die oben genannten Probleme ab, erhöht allerdings auch deutlich den Implementierungsaufwand.

5.3.2 Unterschiede der Schemata

Wie der konkrete Abgleich funktioniert, ist abhängig von dem Schema, in welchem die Daten von der jeweiligen Schwachstellendatenbank bereitgestellt werden. In diesem Abschnitt werden die wesentlichen Unterschiede der Schemata im Bezug auf die Schwachstellenerkennung dargestellt.

OSV-Schema

Das OSV-Schema listet die betroffenen Komponenten als Array unter dem Schlüssel `affected` auf. Zur eindeutigen Identifikation der Komponente sind die Werte `ecosystem` und `name` verpflichtend und deren Aufbau klar definiert. Eine PURL lässt sich somit problemlos in diese Struktur umformen. Optional lässt sich im Schema auch direkt eine PURL angeben.

Für die betroffenen Versionen stellt das Schema einerseits eine Liste, um bei einer vorliegenden Version eine einfache Überprüfung zu ermöglichen, und andererseits eine Gruppe von Versionsreichweiten (*version ranges*) bereit. Mit Hilfe dieser Reichweiten lässt sich eine Handlungsanweisung ableiten, auf welche Version das betroffene Paket aktualisiert werden sollte.

Abbildung 5.5 zeigt einen Ausschnitt aus einem OSV-Eintrag. Durch die angegebene PURL wird das Paket eindeutig identifiziert. Über des Feld *ranges* wird angegeben, dass alle Versionen bis zu (exklusive) 2.3.1 betroffen sind.

NVD-Schema

Das NVD-Schema ist wesentlich ausdrucksstärker. Statt einer Auflistung an betroffener Software sind komplexe Ausdrücke möglich. Über die Operatoren UND, ODER und Negation lässt sich spezifizieren, dass nur eine bestimmte Kombination an Software betroffen ist. Auch hier ist es möglich, Komponenten mittels Versionsreichweiten zu beschreiben (vgl. Abbildung 5.6).

```
{
  "id": "GHSA-jfh8-c2jp-5v3q",
  "aliases": ["CVE-2021-44228"],
  "affected": [{
    "package": {
      "name": "org.apache.logging.log4j:log4j-core",
      "ecosystem": "Maven",
    },
    "ranges": [{
      "type": "ECOSYSTEM",
      "events": [{
        "introduced": "0"
      }, {
        "fixed": "2.3.1"
      }]
    }],
    "versions": ["2.0","2.0-alpha1","2.0-alpha2","2.0-beta1", ...]
  }],
  ...
}
```

Abbildung 5.5: Beispiel für ein betroffenes Paket im OSV-Schema.

```
{
  "cve": {
    "id": "CVE-2021-44228",
    "configurations": [{
      "nodes": [{
        "operator": "OR",
        "negate": false,
        "cpeMatch": [{
          "vulnerable": true,
          "criteria": "cpe:2.3:a:apache:log4j:*:*:*:*:*:*:*:*",
          "versionStartIncluding": "2.0.1",
          "versionEndExcluding": "2.3.1"
        }, {
          "vulnerable": true,
          "criteria": "cpe:2.3:a:apache:log4j:2.0:-:*:*:*:*:*:*",
        }]
      }]
    }
  ]}
}
```

Abbildung 5.6: Beispiel für ein betroffenes Paket im NVD-Schema.

5.3.3 Matching-Algorithmus

Da eine relationale Datenbank verwendet wird, ist es naheliegend, einen `JOIN` zu verwenden, um Komponenten und Sicherheitsempfehlungen miteinander abzugleichen. Die Logik zum Abgleichen von Versionen ist aber schwierig und nicht portabel in SQL umsetzbar. Aus diesem Grund soll der Abgleich in zwei Schritten erfolgen.

Im ersten Schritt wird eine Obermenge der tatsächlichen Schwachstellen gebildet, indem eine Verbundoperation über die Komponenten der Stückliste und der Tabelle der betroffenen Komponenten durchgeführt wird. Dabei werden nur die Attribute betrachtet, die sich portabel und effizient miteinander vergleichen lassen. Insbesondere sollen vorerst die Versionsreichweiten und die komplexen Ausdrücke des NVD-Schemas ignoriert werden.

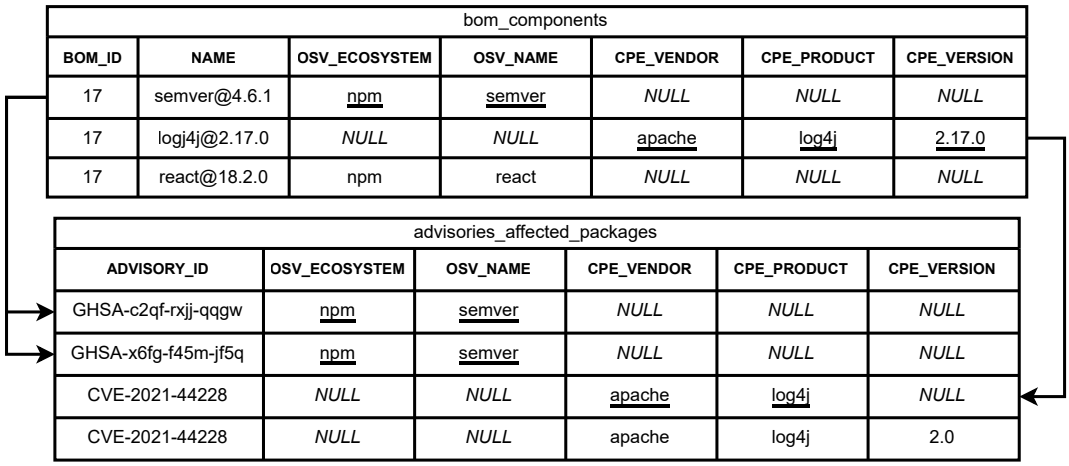


Abbildung 5.7: Verbundoperation zwischen Komponenten der Stückliste und betroffenen Komponenten.

Im zweiten Schritt wird das Ergebnis der Datenbank-Anfrage lokal weiter verfeinert. Dabei wird für jeden Eintrag der Obermenge einzeln geprüft, ob er auf die Stückliste zutrifft. Für Schwachstellen nach dem OSV-Schema stellt die Spezifikation einen Pseudocode[1] zum Abgleichen von Versionen bereit. Der Abgleich der CPE-Namen ist in der 'CPE Name Matching Specification' von Parmelee et al. (2011) beschrieben.

Um Schwachstellen der OSV-Datenbank und NVD nicht doppelt aufzuzählen, müssen zusätzlich Duplikate eliminiert werden. Diese werden über das `aliases`-Feld des OSV-Schemas erkannt (vgl. Abbildung 5.5). Aufgrund der besserer Metadaten verdrängen die Einträge der OSV-Datenbank dabei stets die Einträge der NVD.

[1] https://ossf.github.io/osv-schema/#evaluation

5.3.4 Monitoring

Das kontinuierliche Analysieren von Stücklisten (F-16) erfolgt durch einen Hintergrundablauf, der in regelmäßigen Abständen ausgeführt wird. Um das Monitoring zu optimieren, werden nur die Schwachstellen berücksichtigt, die seit der letzten Analyse hinzugekommen sind oder verändert wurden.

5.4 Export und Annotation

Das Exportieren und Annotieren von Stücklisten wurde in den Anforderungen F-21 bis F-25 festgehalten. Über die Weboberfläche lässt sich auswählen, ob die Stückliste im CSV-, CycloneDX- oder SPDX-Format exportiert werden soll. Wird eine Stückliste im selben Format exportiert, in der sie hochgeladen wurde, wird die hochgeladene Stückliste mit den neuen Informationen ergänzt (vgl. Abbildung 5.8). Andernfalls wird eine neue Datei erzeugt. Da der Prozess sowohl sehr zeit- als auch speicherintensiv sein kann, wird in jedem Fall die exportierte Stückliste für weitere Zugriffe auf dem Objektspeicher abgelegt.

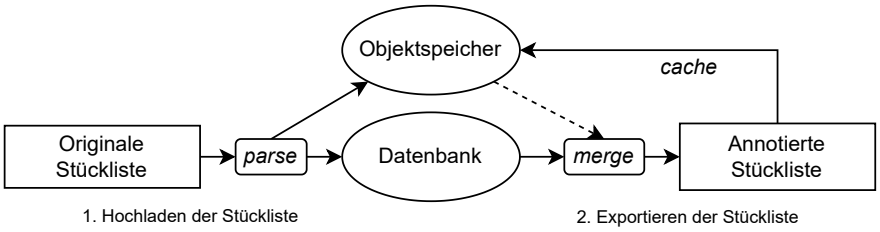


Abbildung 5.8: Übersicht des Annotationsprozesses.

Die Annotation einer SBOM-Datei mit Schwachstelleninformationen kann kritisch gesehen werden. So empfiehlt die NTIA die Schwachstellendaten außerhalb des SBOM zu speichern, da es sich im Gegensatz zu den Komponenten um dynamische Daten handelt, deren Vollständigkeit und Aktualität nicht gewährleistet werden kann (NTIA, 2021). Auch das BSI bezeichnet das Aufnehmen von Schwachstelleninformation in SBOM-Dateien als „nicht sinnvoll“ (BSI, 2023). Aus diesem Grund wird dem Benutzer die Entscheidungsfreiheit gegeben, ob beim Exportieren die Schwachstellen hinzugefügt werden sollen.

In der aktuellen Version der SPDX-Spezifikation sind die Möglichkeiten zur Annotation ohnehin stark begrenzt. Gemäß Anhang K der Spezifikation kann die Annotation über eine *external reference* (F.2.3) erfolgen, die auf eine URL mit der Sicherheitsempfehlung verweist. Abbildung 5.9 zeigt ein Beispiel für ein Softwarepaket mit einer annotierten Sicherheitsempfehlung. Die CycloneDX-Spezifikation erlaubt es hingegen, eine Schwachstelle detailliert zu beschreiben (vgl. Abbildung 5.10).

```
{
  "packages": [{
    "SPDXID" : "SPDXRef-Package1",
    "name": "log4j",
    "filesAnalyzed": false,
    "externalRefs" : [{
      "referenceCategory" : "PACKAGE-MANAGER",
      "referenceLocator" : "pkg:maven/org.apache.logging.log4j/log4j-core@2.1",
      "referenceType" : "purl"
    }, {
      "referenceCategory" : "SECURITY",
      "referenceLocator" : "https://osv.dev/vulnerability/GHSA-jfh8-c2jp-5v3q",
      "referenceType" : "advisory"
    }]
  }],
  ...
}
```

Abbildung 5.9: Beispiel für eine Schwachstelle im SPDX-Format.

```
{
  "components": [{
    "bom-ref": "org.apache.logging.log4j:log4j-core:2.1",
    "name": "log4j",
    "type": "library",
    "purl": "pkg:maven/org.apache.logging.log4j/log4j-core@2.1"
  }],
  "vulnerabilities" : [{
    "bom-ref" : "GHSA-269g-pwp5-87pp-102",
    "id" : "GHSA-269g-pwp5-87pp",
    "source" : {
      "name" : "OSV",
      "url" : "https://osv.dev/vulnerability/GHSA-jfh8-c2jp-5v3q"
    },
    "description" : "Remote code injection in Log4j",
    "detail": "Log4j versions prior to 2.16.0 are subject to a remote...",
    "published": "2021-12-10T00:40:56Z",
    "updated": "2023-04-11T01:49:42.832848Z",
    "affects" : [{
      "ref" : "org.apache.logging.log4j:log4j-core:2.1"
    }]
  }],
  ...
}
```

Abbildung 5.10: Beispiel für eine Schwachstelle im CycloneDX-Format.

6 Implementierung

Dieses Kapitel behandelt die Umsetzung der in den Kapiteln 4 und 5 vorgestellten Konzepte und Algorithmen. Besonderer Fokus liegt dabei auf den Problemen und Herausforderungen, die während der Implementierung aufgetreten sind.

In Abschnitt 6.1 wird zunächst ausführlich auf die Implementierung des Backends eingegangen, bevor im Anschluss in den Abschnitten 6.2 und 6.3 die Realisierung der Webanwendung und der GitHub Action beschrieben wird.

6.1 Backend

Der Spring Monolith besteht aus mehreren Modulen, die jeweils eine bestimmte Funktionalität abdecken. Eine grobe Übersicht der Module mit deren Beziehungen ist in Abbildung 6.1 dargestellt.

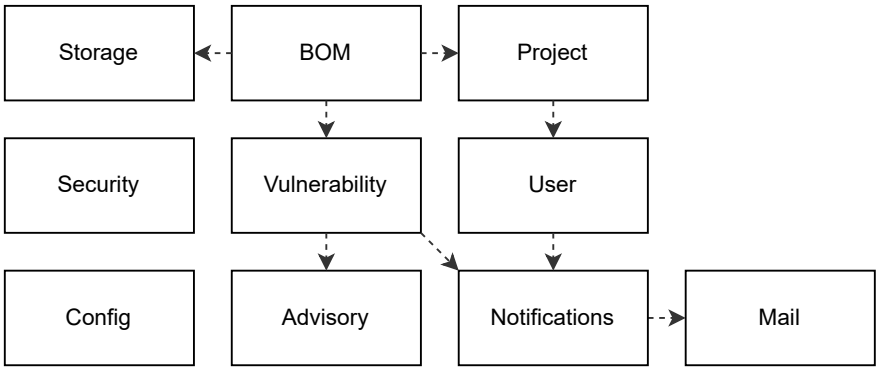


Abbildung 6.1: Modulübersicht des Servers.

Die Struktur der Anwendung orientiert sich an den Konzepten des Domain-Driven Design (DDD) nach Evans (2004) und der konkreten Anwendung nach Sharma (2019) im Kontext des Spring Frameworks. Ein Ausschnitt der Paket-Struktur ist in Abbildung 6.2 dargestellt. In den folgenden Abschnitten soll der Aufbau und die Funktionsweise der einzelnen Module im Detail betrachtet werden.

```
[d.l.vulnaware]
        ├── advisory
        │     ├──model
        │     ├──repository
        │     ├──service
        │     └──task
        ├── bom
        │     ├──controller
        │     ├──dto
        │     ├──model
        │     ├──repository
        │     ├──service
        │     └──task
        ├── ...
        └── VulnAwareApplication.java
```

Abbildung 6.2: Übersicht über die Paket-Struktur des Backends.

6.1.1 SBOM und Komponenten

Das BOM-Modul ist für das Verwalten von Stücklisten und deren Komponenten zuständig. Ein Ausschnitt des Moduls ist in Abbildung 6.3 dargestellt.

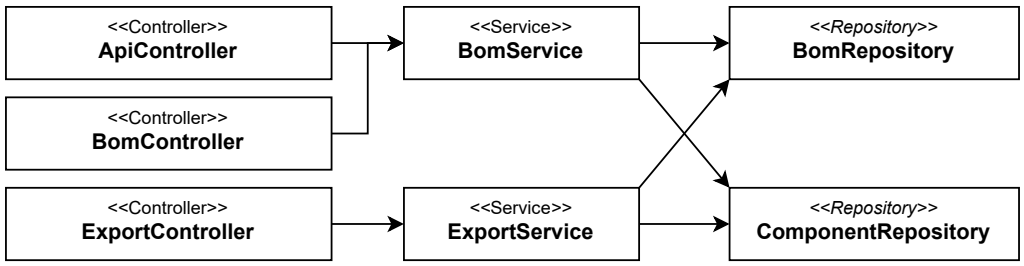


Abbildung 6.3: Ausgewählte Klassen des BOM-Moduls.

Klassen in Spring haben bestimmte Rollen, die über Annotationen wie `@Entity` oder `@Repository` festgelegt werden. Im Folgenden sollen diese zur besseren Verständlichkeit in UML-Diagrammen als Stereotyp angegeben werden.

Extrahieren von Komponenten

Das Hochladen von Dateien wird von Spring über die `MultipartFile`-Klasse ermöglicht. Da mehrere BOM-Standards und Datenformate unterstützt werden sollen, bietet sich das Strategie-Entwurfsmuster für die Analyse der Datei an (vgl. Abbildung 6.4). Über die Methode `parse(MultipartFile)` wird die hochgeladene Datei von der entsprechenden `Parser`-Klasse in eine `BOM`-Entität konvertiert. Diese wird anschließend vom `BomService` persistiert.

Für die Analyse der Dateien wurden die offiziellen Bibliotheken der Standards benutzt (O'Neall, 2023; OWASP Foundation, 2023a). Auch für das Analysieren von PURLs und CPE-Namen wurden entsprechende Bibliotheken verwendet (Springett, 2023a, 2023b).

Klasse	Content-Type	Dateiendung
`SpdxTagParser`	-	.spdx
`SpdxJsonParser`	spdx+json	.spdx.json
`SpdxXmlParser`	-	.spdx.xml
`CycloneDxParser`	vnd.cyclonedx+json	.cdx.json

Tabelle 6.1: Kriterien zur Erkennung des BOM- und Datenformats.

Die Methode `canHandle(MultipartFile)` gibt an, ob die `Parser`-Klasse die Analyse der Datei unterstützt. Somit kann das Format der Datei automatisch erkannt werden. In der Praxis führt dieses Verfahren allerdings zu Problemen, da es sich beim *Content-Type* und der Dateiendung lediglich um Konventionen handelt und sie nicht von den Spezifikationen erzwungen werden. Eine Verbesserung wäre, dem Nutzer zusätzlich die Möglichkeit zu geben, das Format selbst auszuwählen.

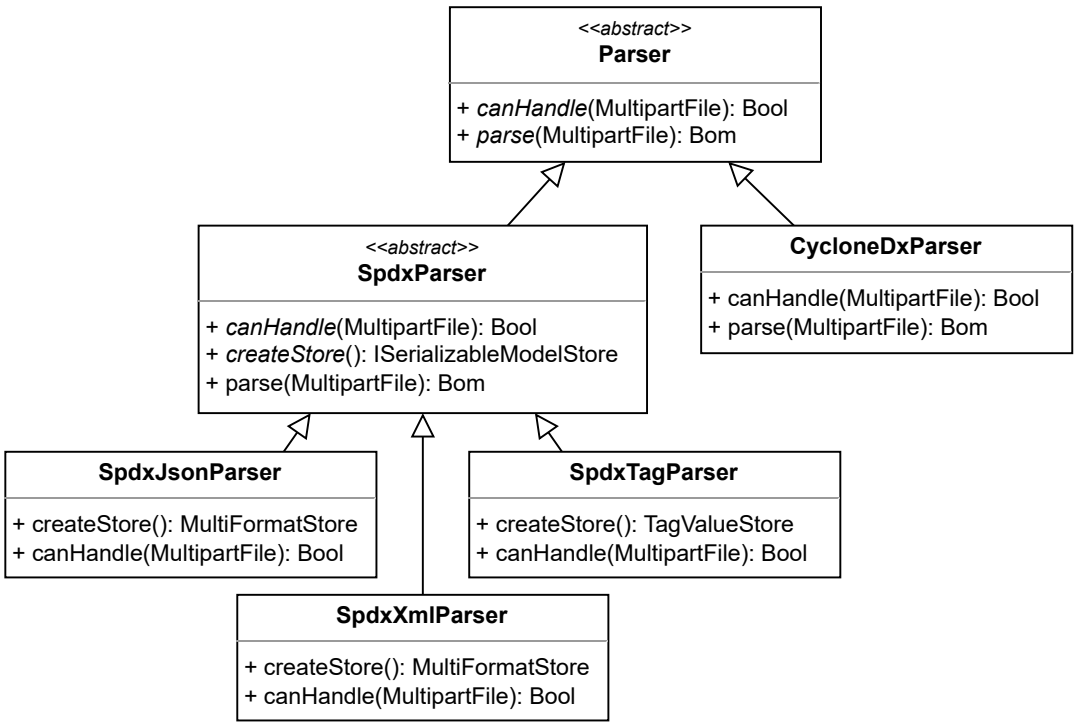


Abbildung 6.4: Implementierungen des Parser-Interfaces.

6.1.2 Sicherheitsempfehlungen

Wie in Abschnitt 5.3.1 beschrieben, sollen die Sicherheitsempfehlungen der OSV-Datenbank und der NVD in die lokale Datenbank importiert werden.

Die `DataSource` Entität speichert den Zustand, in der sich eine Datenquelle befindet. Besonders wichtig ist das Attribut `lastSynchronized`, der den Startpunkt der nächsten Synchronization festlegt und damit inkrementelles Aktualisieren ermöglicht. Während die NVD von einem einzelnen `DataSource`-Objekt beschrieben wird, besitzt jedes Ökosystem der OSV-Datenbank ein eigenes Objekt. So können Ökosysteme ignoriert werden, die sich seit dem letzten Import nicht verändert haben.

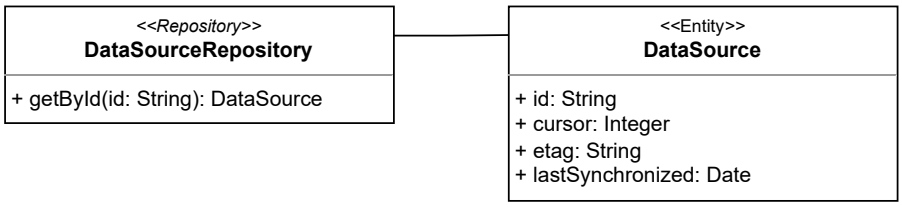


Abbildung 6.5: Klassendiagramm der DataSource-Entität.

Die Entitäten `NvdAdvisory` und `OsvAdvisory` dienen zur Speicherung der importieren Schwachstelleninformationen. Zum einfachen und effizienten Importieren, Aktualisieren und Abrufen werden die Daten denormalisiert als JSON-Zeichenkette in den Attributen `osvRawData` und `nvdRawData` gespeichert. Lediglich die Beschreibung (*summary*) und die Angabe, ob die Sicherheitsempfehlung zurückgezogen wurde (*withdrawn*), werden als eigene Attribute gespeichert, um das Suchen und Filtern nach diesen Werten zu ermöglichen.

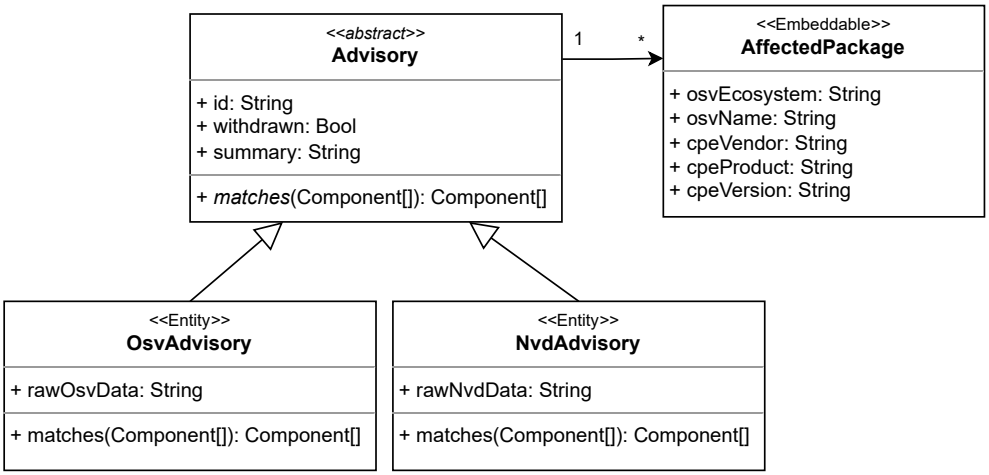


Abbildung 6.6: Klassendiagramm der Advisory-Entität.

Importieren der OSV-Datenbank

Das Importieren der OSV-Datenbank war besonders herausfordernd, da keine offizielle Schnittstelle angeboten wird, die das Abrufen der Daten ermöglicht. Für das Herunterladen der Datenbank wird zwar ein GCS Bucket bereitgestellt, dieser enthält allerdings keine Zeitstempel und erschwert somit das inkrementelle Aktualisieren der Daten.

Der GCS-Bucket enthält die Sicherheitsempfehlungen als einzelne JSON-Dateien, die nach Ökosystemen gruppiert sind. Pro Ökosystem wird eine `all.zip`-Datei bereitgestellt, die alle Sicherheitsempfehlungen komprimiert speichert.

Um die Daten zu importieren, wurde ein `OsvImporterTask` implementiert, der zusammen mit dem `OsvImporterService` für jedes in der Konfiguration angegebene Ökosystem folgenden Prozess ausgeführt:

1. Die von GCS bereitgestellte Prüfsumme (bzw. ETag[1]) der `all.zip` Datei wird mit gespeicherten Prüfsumme des `DataSource`-Entities verglichen. Sind die Werte identisch, ist keine Aktualisierung notwendig.
2. Andernfalls wird die ZIP-Datei heruntergeladen und anschließend über die Inhalte des Archivs iteriert:
   2.1 Ist der Zeitstempel des im OSV-Schema enthaltenem `modified` Feld nicht neuer als die letzte Synchronisation, wird die Schwachstelle ignoriert.
   2.2 Ansonsten wird eine neue `OsvAdvisory`-Entität angelegt und in der Datenbank persistiert. Ein bestehender Eintrag mit der gleichen `id` wird von Hibernate automatisch ersetzt.
3. Die Prüfsumme des ZIP-Archivs wird im `DataSource`-Entity aktualisiert.

Importieren der NVD

Zum Importieren der NVD wurde die offizielle CVE API 2.0[2] verwendet, da sie das vollständige Herunterladen und inkrementelle Aktualisieren bereits unterstützt.

Die Schwierigkeit liegt hier allerdings bei der Größe der Datenbank. Selbst mit der maximalen Seitengröße sind noch über 200 Anfragen nötig, um alle Einträge zu speichern. Durch die Durchsatzbegrenzung der Anfragen und sporadische Fehler wird der Prozess noch weiter verlangsamt.

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag
[2] https://nvd.nist.gov/developers/vulnerabilities

6.1.3 Schwachstellen

Das Schwachstellen-Modul (*vulnerability*) ist für die Verwaltung von Schwachstellen zuständig und führt gemeinsam mit dem Advisory-Modul die Schwachstellenanalyse durch.

Matching-Algorithmus

Der Algorithmus für den Abgleich wurde zuvor in Abschnitt 5.3.3 beschrieben. Die Verbundoperation im ersten Schritt wurde über eine JPQL-Abfrage implementiert, die in Abbildung 6.7 dargestellt ist. Über den Parameter `date` wird ein effizientes Monitoring ermöglicht, indem unveränderte Sicherheitsempfehlungen ignoriert werden.

```
@Query(
  value = "SELECT DISTINCT a " +
  "FROM Advisory a INNER JOIN a.affectedPackages ap " +
  "INNER JOIN Component c ON " +
    "c.osvEcosystem = ap.osvEcosystem AND c.osvName = ap.osvName OR " +
    "(c.cpeVendor = ap.cpeVendor AND c.cpeProduct = ap.cpeProduct
      AND (ap.cpeVersion IS NULL OR c.cpeVersion = ap.cpeVersion))" +
  "WHERE c.bom.id = :bomId AND a.withdrawn = false AND a.updatedAt > :date"
)
List<Advisory> selectByBomAndDate(@Param("bomId") Long bomId,
                                  @Param("date") Date date);
```

Abbildung 6.7: JPQL-Abfrage zur (groben) Erkennung betroffener Pakete.

Im zweiten Schritt wird die `matches(Component[])`-Methode für jede `Advisory`-Klasse aufgerufen, um Sicherheitsempfehlungen zu filtern, die nicht auf die Stückliste zutreffen, sowie die betroffenen Komponenten zu ermitteln.

War die Stückliste schon vor dem Prozess von der Schwachstelle betroffen, wird die jeweilige `Vulnerability`-Entität aktualisiert. Andernfalls wird eine neue Entität angelegt.

Vergleichen von Versionen

Eine weitere Schwierigkeit stellt das Vergleichen von Versionen dar. Um festzustellen, ob eine Komponente betroffen ist, muss überprüft werden, ob die Version der Komponente in den entsprechenden Reichweiten liegt. Hersteller können allerdings ihre Versionen in der Regel frei wählen und eine lineare Ordnung muss nicht immer direkt erkennbar sein. Besonders schwierig ist dies beim NVD-Schema, da hier auch kein Ökosystem (NPM, Maven, etc.) angegeben ist, welches in der Regel Konventionen für die Struktur vorschreibt.

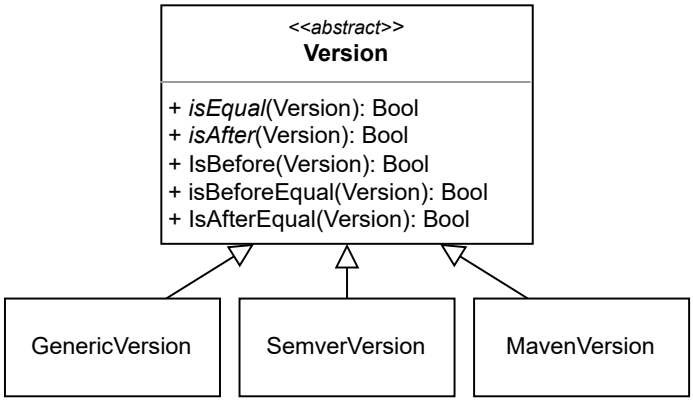


Abbildung 6.8: Klassen zum Vergleichen von Versionen.

Abhilfe schafft hier die „Semantic-Versioning“ Spezifikation von Preston-Werner (n. d.), die den Aufbau und die Bedeutung von Versionen eindeutig definiert. Der dazugehörige Algorithmus wurde in der Klasse `SemverVersion` implementiert. Dieser kann allerdings nur angewendet werden, wenn im OSV-Schema der Typ `SEMVER` auch explizit angegeben ist.

Um einzelne Ökosysteme zu unterstützen, können weitere Implementierungen der Klasse `Version` hinzugefügt werden (siehe Abbildung 6.8). Für Pakete im Maven-Ökosystem dient die Klasse `MavenVersion`, die auf die offizielle Maven-Bibliothek (The Apache Software Foundation, 2023) zurückgreift, um die Versionen zu vergleichen.

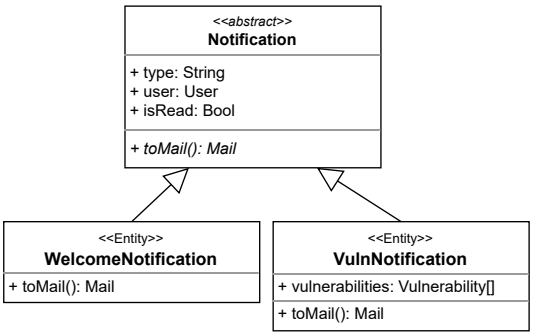
Sind keine über Informationen über das Ökosystem vorhanden, wird die Klasse `GenericVersion` benutzt, welche allgemein versucht, möglichst viele Fälle abzudecken.

6.1.4 Benachrichtigungen

Das Benachrichtigungs-Modul (*notifications*) ist für das Erstellen und Abrufen von Benachrichtigungen zuständig. Es sind zwei Typen von Benachrichtigungen implementiert:

- `WelcomeNotification`: Benachrichtigung, die dem Nutzer gesendet wird, wenn er sich registriert.
- `VulnNotification`: Benachrichtigung, die an alle Projektmitglieder gesendet wird, wenn neue Schwachstellen in einem Projekt gefunden werden.

Für jeden dieser Typen wird eine eigene Entität in der Datenbank angelegt, die alle nötigen Informationen speichert bzw. referenziert, um die Nachricht anzuzeigen (siehe Abbildung 6.9).




```java
@Transient
@Override
public Mail toMail() {
  final String subject = "Welcome to VulnAware!";
  final String body = String.format("""
  Dear %s,

  thank you for registering!
  Create a new project to get started.
  """, getUser().getName());

  return new Mail(subject, body);
}
```

Abbildung 6.9: Implementierung der Benachrichtigungen.

Senden von E-Mails

Zum Versenden von E-Mails werden die von Spring Boot bereitgestellten Klassen `JavaMailSender` und `SimpleMailMessage` verwendet. Jeder Benachrichtigungstyp implementiert die Methode `toMail()`, welche die Benachrichtigung in ein Mail-Objekt umwandelt. Dieses kann dann von der Klasse `MailService` an die E-Mail-Adresse des Nutzers gesendet werden.

6.1.5 Speicher

Das Speicher-Modul (*storage*) interagiert mit dem Objektspeicher und ist für das Speichern und Abrufen von Dateien zuständig.
Das Hochladen einer Datei läuft über den `AssetService`, der die Elemente des Objektspeichers im Überblick behält, indem er ein entsprechendes `Asset`-Entity in der Datenbank anlegt. Der `StorageService` ist für die eigentliche Verwaltung der Dateien zuständig.

Es sind zwei Implementierungen für den `StorageService` umgesetzt:

- `LocalStorageService`: Speichert die Dateien lokal in dem Dateisystem. Das ist die Standardoption und wird vor allem für die Entwicklung verwendet.
- `CloudStorageService`: Speichert die Dateien in einem GCS-Bucket. Diese Option wird im SaaS Betrieb verwendet und ermöglicht horizontale Skalierung. Für die Interaktion wurde die offizielle Google Cloud Storage Bibliothek für Java benutzt (Google, 2023c).

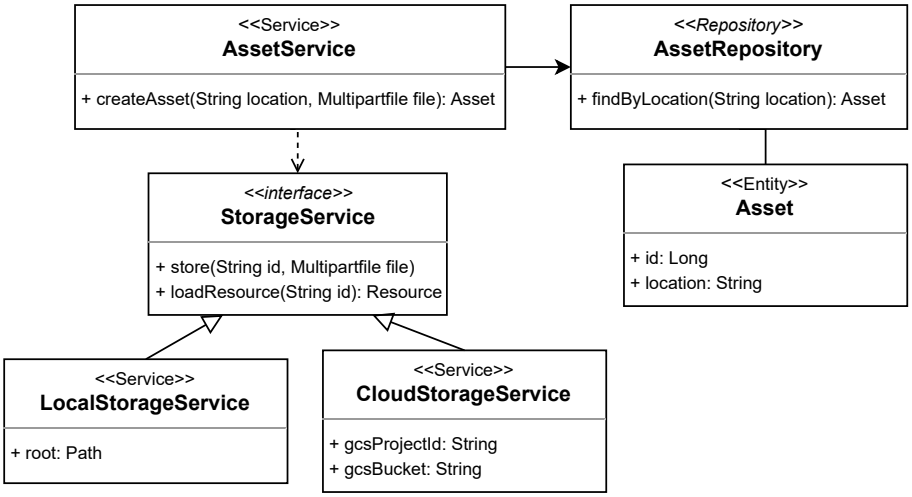


Abbildung 6.10: Übersicht der Klassen des Speicher-Moduls.

6.1.6 Sicherheit

Um die nötigen Sicherheitsmaßnahmen der Applikation zu implementieren, wurde das Modul Spring Security (VMware, 2023b) des Spring Frameworks verwendet.

Authentifikation

Unter *Authentifikation* versteht man den Identitätsnachweis des Benutzers gegenüber dem System. Es wurden zwei verschiedene Authentifikationsmechanismen implementiert, die abhängig vom Typ der Schnittstelle zum Einsatz kommen:

- Authentifikation mittels JSON Web Token (JWT)
- Authentifikation mittels API-Schlüssel

Mechanismus	**JWT**	**API-Schlüssel**
Format	signiertes JSON-Objekt (base64)	randomisierte Zeichenkette
Erstellung	bei erfolgreicher Anmeldung	über Benutzeroberfläche
Schnittstelle	*privat* (Angular Frontend)	*extern* (GitHub Action)
Gültigkeit	12 Stunden	unbeschränkt
Implementierung	`JwtFilter`	`ApiKeyFilter`
assoziiert mit	Benutzer	Benutzer

Tabelle 6.2: Gegenüberstellung der Authentifikationsmechanismen.

Umgesetzt wurden die Mechanismen über die Erweiterung der von der Spring Security bereitgestellten Klassen, wie in Abbildung 6.11 dargestellt. Die Klasse `SecurityConfig` ist dabei für die Konfiguration des Frameworks zuständig und bindet die restlichen Klassen ein.

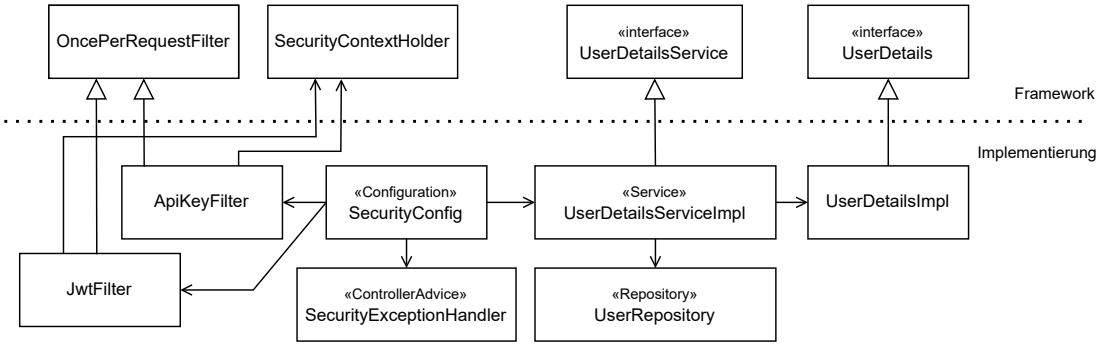


Abbildung 6.11: Übersicht der Klassen des Sicherheit-Moduls.

Abhängig von der Syntax des Anfragepfads ist entweder der `JwtFilter` oder der `ApiKeyFilter` für die Authentifikation einer Anfrage zuständig. Der `JwtFilter` überprüft dabei die Gültigkeit des JWT und findet den zugehörigen Benutzer über die im Token codierte E-Mail-Adresse. Beim `ApiKeyFilter` hingegen wird mittels des im HTTP-Header angegebenen eindeutigen Schlüssels direkt der dazugehörige Benutzer ermittelt.

Die Klasse `AuthController` bietet Endpunkte für die Anmeldung und Registrierung eines Benutzers mit E-Mail-Adresse und Passwort an. Die Verifikation von E-Mail-Adressen wurde allerdings nicht implementiert.

Für die Erstellung und Validierung der JWTs wurde die `java-jwt` Bibliothek (Auth0 Inc., 2023) verwendet.

Autorisierung

Autorisierung bezeichnet das Absichern von Ressourcen vor unerlaubten Zugriffen. Insgesamt gibt drei Fälle die beachtet werden müssen:

1. Benutzer dürfen nur auf ihre eigenen Benachrichtigungen zugreifen
2. Benutzer dürfen nur auf ihre eigenen Kontodaten zugreifen
3. Nur Projektmitglieder dürfen auf ein Projekt zugreifen

Die notwendigen Maßnahmen wurden ebenfalls mittels Spring Security implementiert. Dazu bietet Spring Annotationen für Methoden an, die vor oder nach Ausführung eine bestimmte Aussage überprüfen. In Abbildung 6.12 wird die Annotation `@PostAuthorize` benutzt, um sicherzustellen, dass der aktuelle Benutzer Mitglied des angeforderten Projektes ist.

```java
@Transactional(readOnly = true)
@PostAuthorize("@projectService.isMember(authentication, returnObject)")
public Project findByIdentifier(final String identifier) {
  ...
}

@Transactional(readOnly = true)
public boolean isMember(final Authentication auth, final Project project) {
  return project
    .getMembers()
    .stream()
    .map(member -> member.getUser().getId())
    .anyMatch(id -> id.equals(((UserDetailsImpl) auth.getPrincipal()).getId()));
}
```

Abbildung 6.12: Codeausschnitt der Klasse ProjectService zur Absicherung von Projekten.

6.2 Webanwendung

Die Webanwendung hat die Hauptaufgabe, dem Benutzer die Daten aus dem Backend zu präsentieren. Dafür nutzt sie das `HttpClient`-Modul von Angular, um REST-Anfragen an die (private) Schnittstelle des Servers zu senden. Es wird dabei ein ähnlich modularer Aufbau wie im Backend verfolgt. Jede Domäne besitzt einen *Service*, der mit der dazugehörigen Domäne des Backends kommuniziert (siehe Abbildung 6.13).

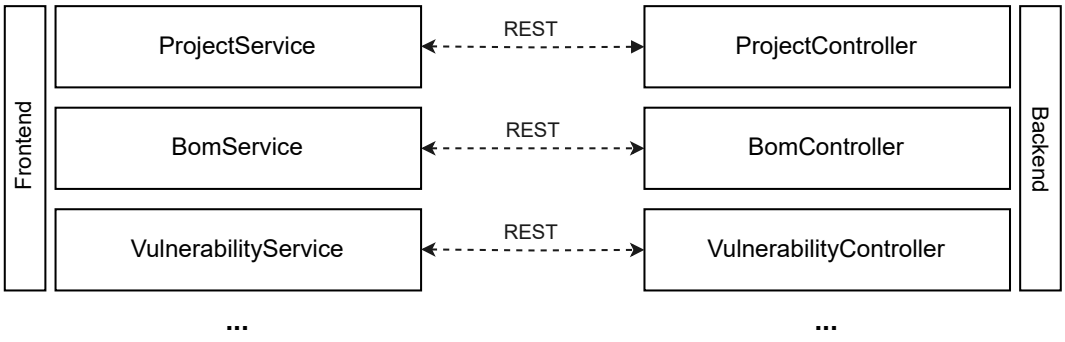


Abbildung 6.13: Kommunikation zwischen Frontend und Backend.

6.2.1 Angular Material

Als wichtige UI-Bibliothek wird Angular Material (Google, 2023d) verwendet. Direkt vom Angular Team entwickelt, stellt Angular Material eine Reihe von vorgefertigten Komponenten bereit, die eine einheitliche Gestaltung der Anwendung nach dem Material Design[3] ermöglichen.

6.2.2 Seitenübersicht

Abbildung 6.14 gibt eine Übersicht über der Seiten der Webanwendung. Unterschiedliche Farben repräsentieren unterschiedliche Angular Module, die erst beim Aufruf einer der entsprechenden Seiten geladen werden (*lazy loading*). Dadurch werden die Module besser entkoppelt und eine höhere Seitenladegeschwindigkeit erzielt. Ausschnitte der Benutzeroberfläche sind in Anhang B dargestellt.

Das Komponenten-Modul ist trotz der Zugehörigkeit zum Projekt-Modul in einem eigenen Modul, um die Bibliotheken zur Darstellung von Diagrammen besser zu isolieren.

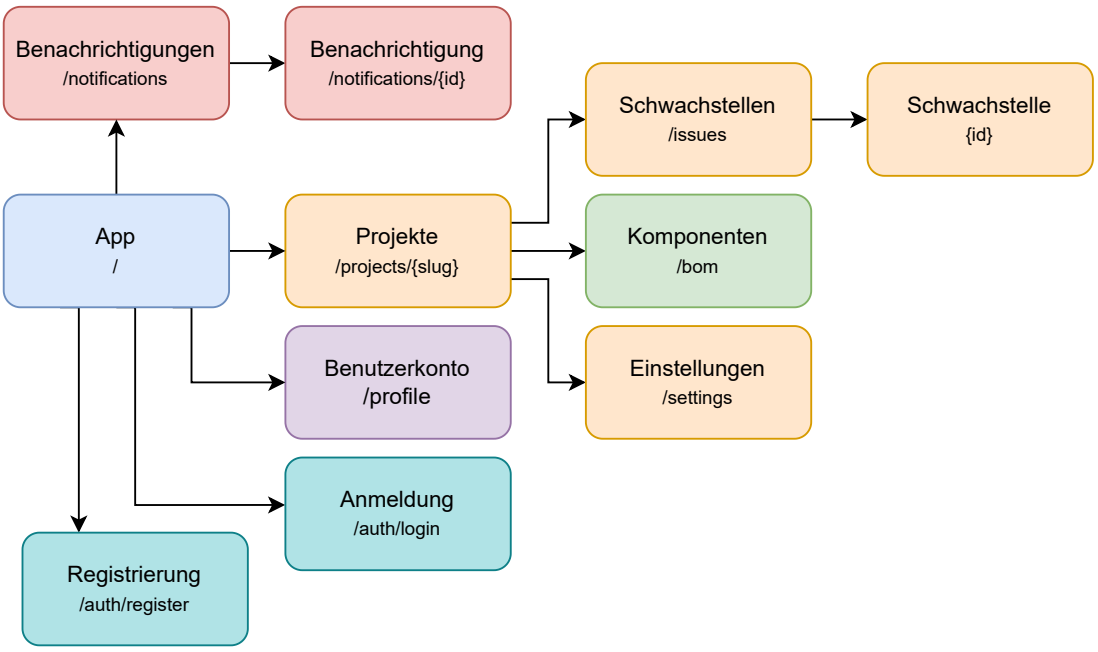


Abbildung 6.14: Übersicht der Module und Seiten des Frontends.

[3] https://m2.material.io/

6.2.3 Authentifizierung

Damit die Anwendung auf die Schnittstellen des Backends zugreifen kann, ist ein gültiger JSON Web Token (JWT) erforderlich. Dieser wird nach Anmeldung oder Registrierung erworben und im *localStorage*[4] des Browsers gespeichert. Über einen `HttpInterceptor` (vgl. Abbildung 6.15) wird der JWT bei jeder Anfrage an das Backend im HTTP-Header mitgeschickt.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
    constructor(private auth: AuthService) {}

    intercept(req: HttpRequest<any>, next: HttpHandler) {
        const authToken = this.auth.getToken();
        if (authToken) {
            return next.handle(
                req.clone({ setHeaders: { Authorization: `Bearer ${authToken}` } })
            );
        }
        return next.handle(req);
    }
}
```

Abbildung 6.15: Implementierung der AuthInterceptor-Klasse.

6.3 GitHub Action

Es gibt es mehrere Wege, eine *GitHub Action* zu erstellen. Es wurde sich für eine JavaScript-Action entschieden, da diese im Gegensatz zu Docker-Containern weniger Overhead bietet und direkt auf der Maschine ausgeführt werden kann.

Um im ersten Schritt die Stückliste zu generieren, wird die REST-Schnittstelle[5] von GitHub verwendet, die es ermöglicht, den Abhängigkeitsgraphen eines Repositories im SPDX-Format zu exportieren. Für die einfache Interaktion mit der GitHub-API wurde die *octokit*[6] Bibliothek verwendet. Der *Tag* des SBOM wird auf den aktuellen Branch bzw. die aktuelle Pull-Request gesetzt. Aufgrund der Limitationen von GitHub kann jedoch derzeit nur der Haupt-Branch exportiert werden.

Nach der Erstellung wird die Datei im zweiten Schritt über den entsprechenden API-Endpunkt zum angegebenen VulnAware-Server hochgeladen.

[4] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
[5] https://docs.github.com/en/rest/dependency-graph/sboms
[6] https://github.com/octokit/octokit.js

Schlüssel	Benötigt?	Beschreibung
`token`	Ja	GitHub-Token, um auf die GitHub API zugreifen zu können.
`apiKey`	Ja	API-Key, der über die Webanwendung generiert wurde.
`server`	Ja	URL zum VulnAware Server, wo die Datei hochgeladen werden soll.
`project`	Ja	ID des VulnAware Projektes.
`maxVulns`	Nein	Anzahl der erlaubten Schwachstellen, bevor der Prozess fehlschlägt.

Tabelle 6.3: Konfigurationsmöglichkeiten der GitHub Action.

Die Implementierung war jedoch unerwartet kompliziert, da für den Upload mindestens NodeJS 18 benötigt wird, GitHub Actions aber aktuell höchstens NodeJS 16 unterstützen. Aus diesem Grund musste für die HTTP-Anfrage die *axios*-Bibliothek[7] hinzugezogen werden. Abbildung 6.16 zeigt ein Beispiel, wie die GitHub Action konfiguriert werden kann.

```yaml
name: Upload BOM
on:
  push:
    branches: [main, dev]
jobs:
  upload-bom:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v3
        with:
          node-version: "latest"
      - name: Upload BOM
        uses: lukasnehrke/vulnaware/action
        with:
          token: ${{ secrets.GITHUB_TOKEN }}
          apiKey: ${{ secrets.VA_API_KEY }}
          server: "https://va-web-am2gwvqqva-ew.a.run.app"
          project: "vulnaware-875774"
```

Abbildung 6.16: Beispielkonfiguration der GitHub Action.

[7] Version 1.4.0; https://github.com/axios/axios

7 Evaluation

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

7.1 Evaluation funktionaler Anforderungen

Die funktionalen Anforderungen wurden in Abschnitt 3.1 festgehalten.

Allgemeine Anforderungen

F-01: Die Anforderung wurde erfüllt. Das System erlaubt es Benutzern, sich mittels E-Mail-Adresse und Passwort anzumelden.

F-02: Die Anforderung wurde erfüllt. Das System erlaubt es, externe Dienste über einen API-Schlüssel zu authentifizieren, wenn diese auf die Schnittstellen zur Automatisierung zugreifen wollen. (vgl. Abschnitt 6.1.6)

F-03: Die Anforderung wurde teilweise erfüllt. Das System erlaubt es Benutzern, sich über die Weboberfläche mittels E-Mail-Adresse und Passwort zu registrieren. Es wird allerdings zusätzlich ein Name benötigt, der zum Beispiel als Anrede in E-Mails verwendet wird.

F-04: Die Anforderung wurde erfüllt. Das System erlaubt es Benutzern, einen API-Schlüssel zu generieren (vgl. Abschnitt 6.1.6).

F-05: Die Anforderung wurde erfüllt. Das System erlaubt es authentifizierten Benutzern, mehrere Projekte zu verwalten.

F-06: Die Anforderung wurde erfüllt. Das System erlaubt es Projekterstellern, Benutzer als Mitglied eines Projektes hinzuzufügen und zu entfernen.

Verwaltung von Stücklisten

F-07: Die Anforderung wurde erfüllt. Das System unterstützt das Hochladen von SBOM-Dateien im SPDX-Format.

F-08: Die Anforderung wurde erfüllt. Das System unterstützt das Hochladen von SBOM-Dateien im CycloneDX-Format.

F-09: Die Anforderung wurde erfüllt. Das System erlaubt es Projektmitgliedern, mehrere Stücklisten zu verwalten.

F-10: Die Anforderung wurde erfüllt. Das System implementiert den Versionierungsprozess, um die alte Stückliste durch die neue zu ersetzen (vgl. Abschnitt 5.4).

F-11: Die Anforderung wurde teilweise erfüllt. Das System erlaubt die Identifikation von Komponenten mittels PURLs und CPE-Namen. Die Identifikation basierend auf SWID-Tags wurde nicht umgesetzt, da sie kaum von SBOM-Generatoren und Schwachstellendatenbanken unterstützt werden.

F-12: Die Anforderung wurde teilweise erfüllt. Das System nutzt die `deps.dev` API, um die Lizenz und die aktuellste Version einer Komponente herauszufinden. Allerdings wird für jede Komponente mindestens eine API-Anfrage benötigt. Da Stücklisten hunderte bis tausende Komponenten enthalten können, ist die Implementierung nicht skalierbar.

F-13: Die Anforderung wurde nicht erfüllt. Das manuelle Modifizieren von Komponenten ist sowohl umständlich für den Benutzer als auch umständlich zu implementieren. Aus diesem Grund wurde die Anforderung nicht umgesetzt.

F-14: Die Anforderung wurde erfüllt. Das Weboberfläche ist fähig, die Komponenten der Stückliste tabellarisch anzuzeigen.

Schwachstellenanalyse und -monitoring

F-15: Die Anforderung wurde erfüllt. Das System nutzt die OSV-Datenbank und die NVD, um Komponenten auf Basis von PURL und CPE abzuglechen.

F-16: Die Anforderung wurde nicht erfüllt. Das System gleicht Komponenten in regelmäßigen Abständen neu ab. Dies ist einfacher zu implementieren und hat keine signifikanten Nachteile.

F-17: Die Anforderung wurde erfüllt. Das System erlaubt es Projektmitgliedern, gefundene Schwachstellen mit einem Status und Kommentaren zu versehen.

F-18: Die Anforderung wurde erfüllt. Das System erlaubt es Projektmitgliedern, gefundene Schwachstellen zu ignorieren.

F-19: Die Anforderung wurde teilweise erfüllt. Das Benutzeroberfläche zeigt Beschreibungen und Risiken für gefundene Schwachstellen an. Handlungsempfehlungen werden hingegen nur für Ökosysteme angezeigt, deren Versionen eindeutig verglichen werden können.

F-20: Die Anforderung wurde erfüllt. Das Benutzeroberfläche erlaubt es, gefundene Schwachstellen nach Status und Risiko zu sortieren.

Export der Komponenten

F-21: Die Anforderung wurde erfüllt. Das System bietet Projektmitgliedern die Möglichkeit, Stücklisten als SBOM-Datei im SPDX-Format herunterzuladen. Wurde die Stückliste jedoch zuvor im CycloneDX-Format hochgeladen, gehen bei der Konvertierung einige Informationen verloren (vgl. Abschnitt 5.4).

F-22: Die Anforderung wurde erfüllt. Das System bietet Projektmitgliedern die Möglichkeit, Stücklisten als SBOM-Datei im CycloneDX-Format herunterzuladen. Wurde die Stückliste jedoch im SPDX-Format hochgeladen, gehen bei der Konvertierung einige Informationen verloren (vgl. Abschnitt 5.4).

F-23: Die Anforderung wurde erfüllt. Das System bietet Projektmitgliedern die Möglichkeit, die Stückliste als CSV-Datei herunterzuladen. Enhaltene Spalten sind Name, PackageURL, CPE und IDs der Schwachstellen, die die Komponente betreffen.

F-24: Die Anforderung wurde erfüllt. Das System bietet Projektmitgliedern die Möglichkeit, die mit Schwachstelleninformationen annotierte SBOM-Datei im SPDX-Format herunterzuladen.

F-25: Die Anforderung wurde erfüllt. Das System bietet Projektmitgliedern die Möglichkeit, die mit Schwachstelleninformationen annotierte SBOM-Datei im CycloneDX-Format herunterzuladen.

Benachrichtigungen

F-26: Die Anforderung wurde teilweise erfüllt. Sobald neue Schwachstellen in einer Stückliste gefunden werden, wird eine Benachrichtigung an die Projektmitglieder gesendet. Wird die Benachrichtigung als E-Mail gesendet, enthält sie lediglich die Information, dass neue Schwachstellen verfügbar sind.

F-27: Die Anforderung wurde erfüllt. Der Benutzer kann Benachrichtigungen über die Seiten `/notifications` und `/notifications/{id}` ansehen. Das Besuchen der Seite einer Benachrichtigung markiert die Benachrichtigung als gelesen.

F-28: Die Anforderung wurde teilweise erfüllt. Das System ist fähig, Benachrichtigungen als E-Mail zu versenden (vgl. Abschnitt 6.1.4). Es ist Benutzern allerdings nicht möglich, zu konfigurieren, ob und welche Benachrichtigungen sie erhalten möchten.

Automatisierung

F-29: Die Anforderung wurde erfüllt. Das System bietet über den `ApiController` den Endpunkt `/api/v1/{project}/upload/` an, über den eine SBOM-Datei hochgeladen werden kann.

F-30: Die Anforderung wurde erfüllt. Das System bietet über den `ApiController` den Endpunkt `/api/v1/{project}/analyze/` an, über den eine SBOM-Datei hochgeladen werden kann und eine Liste von IDs der gefundenen Schwachstellen zurückgegeben wird.

7.2 Evaluation nichtfunktionaler Anforderungen

Die nichtfunktionalen Anforderungen wurden in Abschnitt 3.2 festgehalten.

Effizienz

NF-01: Die Anforderung wurde erfüllt. Nachdem eine Stückliste analysiert wurde, wird der Abgleich nach Schwachstellen asynchron ausgeführt.

NF-02: Die Anforderung wurde erfüllt. Nachdem eine Stückliste generiert wurde, wird sie auf dem Objektspeicher für weitere Zugriffe abgelegt. Haben sich die Komponenten oder Schwachstellen geändert, wird die Datei stattdessen neu generiert.

NF-03: Die Anforderung wurde nicht erfüllt. Zwischen dem Hinzufügen einer Schwachstelleninformation in der OSV-Datenbank und der Benachrichtigung des Nutzern liegen bis zu 24 Stunden, da der GCS-Bucket, den die Software importiert, nur einmal täglich aktualisiert wird.

NF-04: Die Anforderung wurde nicht erfüllt. Für den Matching-Algorithmus werden immer alle Komponenten einer Stückliste betrachtet. Zur Optimierung werden aber Sicherheitsempfehlungen gefiltert, die sich seit der letzten Analyse nicht geändert haben.

Kompatibilität

NF-05: Die Anforderung wurde erfüllt. Das System unterstützt die SPDX-Spezifikation in der Version 2.3 in den Formaten Tag-Value, JSON, und XML (siehe Abschnitt 6.1.1).

NF-06: Die Anforderung wurde erfüllt. Das System unterstützt die CycloneDX-Spezifikation in der Version 1.4 im JSON-Format (siehe Abschnitt 6.1.1).

Portabilität

NF-07: Die Anforderung wurde erfüllt. Durch die Nutzung des ORM-Frameworks Hibernate ist die Architektur nicht abhängig von einem bestimmten DBMS.

NF-08: Die Anforderung wurde erfüllt. Die Architektur kann in verschiedenen Umgebungen bereitgestellt werden und ist nicht abhängig von einem bestimmten Cloud-Anbieter.

Sicherheit

NF-09: Die Anforderung wurde erfüllt. Das System nutzt das Spring Security Framework, um sicherzustellen, dass nur authentifizierte Benutzer auf den Dienst zugreifen können (siehe Abschnitt 6.1.6).

NF-10: Die Anforderung wurde erfüllt. Das System nutzt das Spring Security Framework, um sicherzustellen, dass nur Projektmitglieder auf ein Projekt zugreifen können (siehe Abschnitt 6.1.6).

Benutzbarkeit

NF-11: Die Anforderung wurde erfüllt. Die Weboberfläche nutzt die Angular Material Bibliothek, um den Material Design Guidelines zu folgen (vgl. Abschnitt 6.2.1).

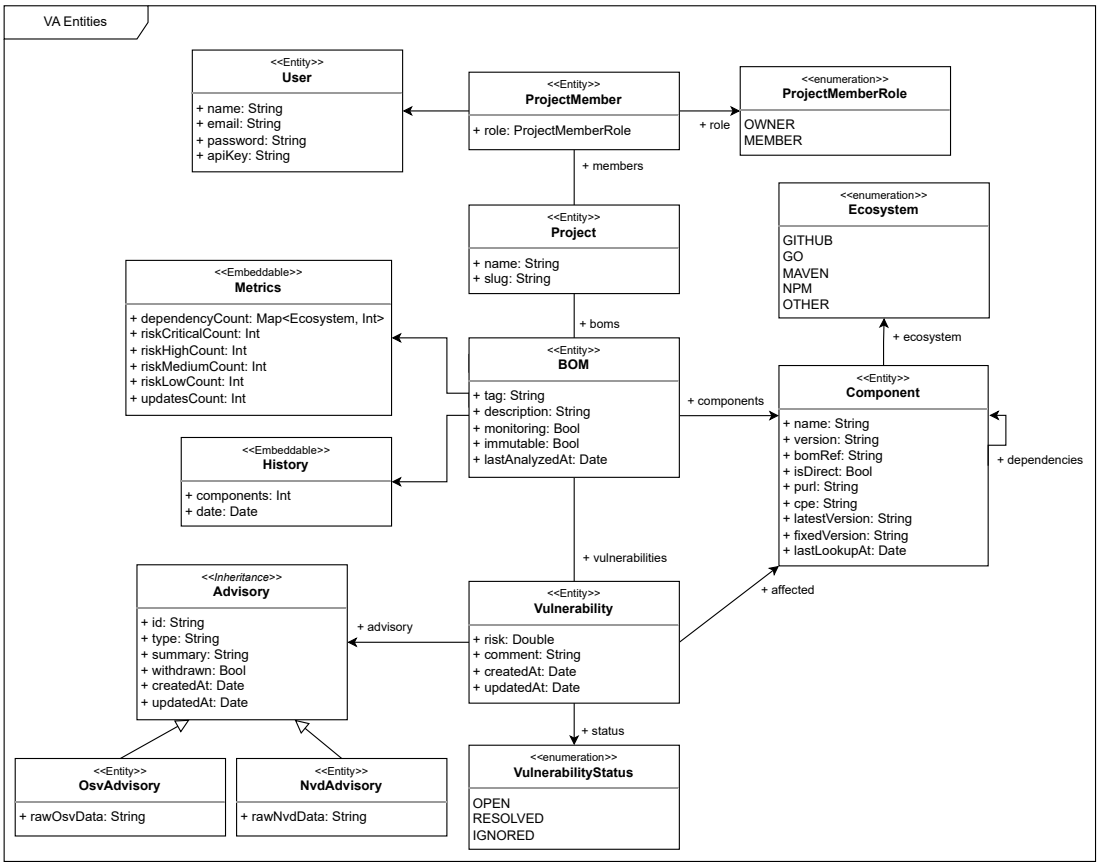
NF-12: Die Anforderung wurde nicht erfüllt. Die Weboberfläche ist nur teilweise responsiv und auf kleineren Geräten wie Smartphones kaum nutzbar. Die Schwierigkeit lag hier vor allem bei der Darstellung der Diagramme und Tabellen auf geringeren Bildschirmbreiten.

8 Fazit

Das Ziel dieser Arbeit bestand in der Entwicklung einer Software, die Entwickler dabei unterstützt, bekannte Schwachstellen in ihren Softwareabhängigkeiten zu identifizieren und zu beheben. Dazu wurde der Webdienst „VulnAware“ vorgestellt, der die Stückliste eines Softwareprodukts entgegennimmt und die darin enthaltenen Komponenten kontinuierlich nach Schwachstellen abgleicht. Über die Weboberfläche wird Entwicklern zudem intuitiv präsentiert, welche Komponenten sie verwenden, welches Risiko von diesen Komponenten ausgeht und wie sich mögliche Risiken beheben lassen. Durch die Unterstützung mehrerer SBOM-Standards sowie der PURL- und CPE-Spezifikation wird eine besonders hohe Kompatibilität zu anderen Systemen und Werkzeugen ermöglicht. Außerdem lässt sich der Webdienst mithilfe der Schnittstellen zur Automatisierung und der GitHub Action besonders leicht in den Entwicklungsprozess integrieren. So wird Entwicklern und Sicherheitsteams nicht nur mehr Arbeit abgenommen, sondern es ermöglicht auch das Erkennen von Sicherheitsproblemen, noch bevor eine verwundbare Komponente ausgeliefert wird.

Anhang

A Entitäten und Beziehungen



B Benutzeroberfläche

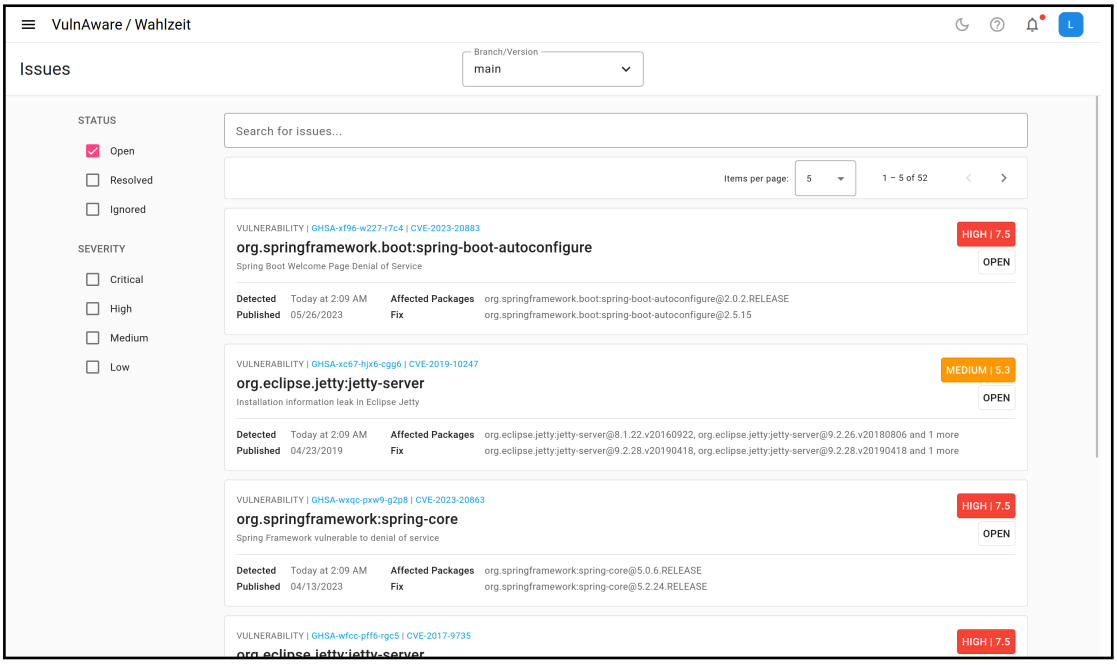


Abbildung 1: Seite zur Verwaltung von Schwachstellen.

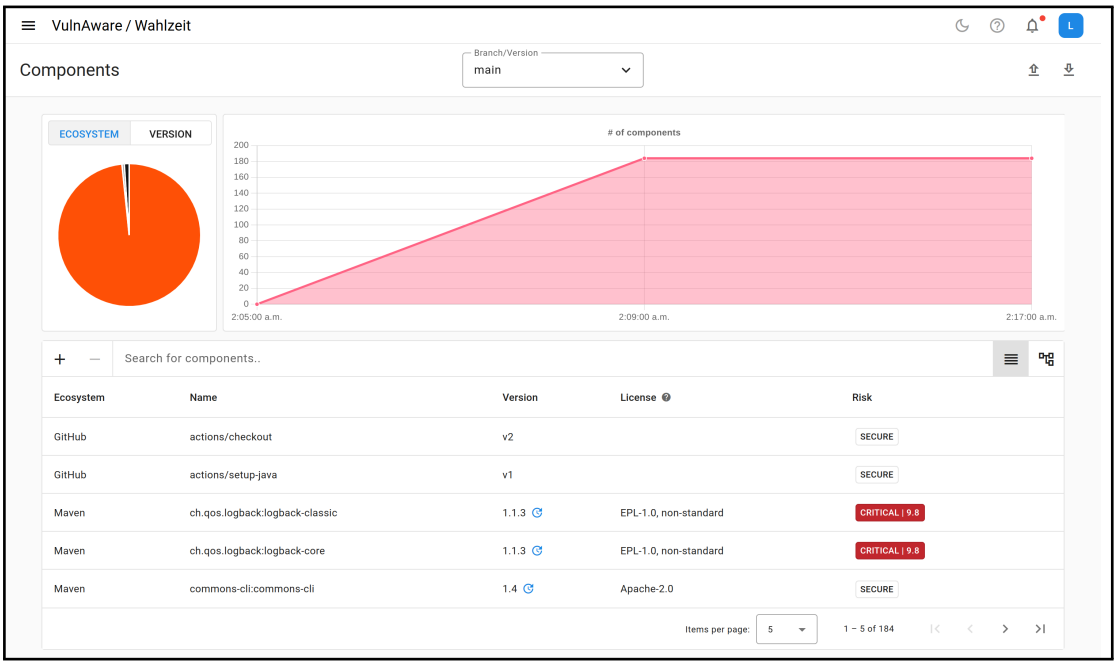


Abbildung 2: Seite zur Verwaltung von Komponenten.

Literaturverzeichnis

Auth0 Inc. (2023). *java-jwt* (Version 4.4.0). https://github.com/auth0/java-jwt

BSI. (2021). *Kritische Schwachstelle in log4j veröffentlicht (CVE-2021-44228)*. Verfügbar 20. August 2023 unter https://www.bsi.bund.de/SharedDocs/Cybersicherheitswarnungen/DE/2021/2021-549032-10F2.pdf?__blob=publicationFile&v=10

BSI. (2023). *Technische Richtlinie TR-03183: Cyber-Resilienz-Anforderungen an Hersteller und Produkte*. Verfügbar 12. August 2023 unter https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03183/BSI-TR-03183-2.pdf?__blob=publicationFile&v=3

Byers, R., Turner, C., & Brewer, T. (2022). *National Vulnerability Database*. National Institute of Standards and Technology. https://doi.org/10.18434/M3436

Chang, O., & Lewandowski, K. (2021). *Launching OSV - Better vulnerability triage for open source*. Verfügbar 12. August 2023 unter https://security.googleblog.com/2021/02/launching-osv-better-vulnerability.html

Cheikes, B., Waltermire, D., & Scarfone, K. (2011). *Common Platform Enumeration: Naming Specification Version 2.3*. https://doi.org/10.6028/NIST.IR.7695

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley.

GitHub. (2023a). *About the dependency graph*. Verfügbar 11. August 2023 unter https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph

GitHub. (2023b). *About the GitHub Advisory database*. Verfügbar 20. Mai 2023 unter https://docs.github.com/en/code-security/security-advisories/global-security-advisories/about-the-github-advisory-database

GitHub. (2023c). *GitHub Advisory Database*. Verfügbar 20. Mai 2023 unter https://github.com/advisories

GitHub. (2023d). *Keeping your supply chain secure with Dependabot*. Verfügbar 20. Mai 2023 unter https://docs.github.com/en/code-security/dependabot

GitHub. (2023e). *Understanding GitHub Actions*. Verfügbar 12. August 2023 unter https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Google. (2023a). *About Open Source Insights*. Verfügbar 12. August 2023 unter https://deps.dev/about

Google. (2023b). *Angular: A platform for building mobile and desktop web applications* (Version 16.2). https://angular.io/

Google. (2023c). *Java idiomatic client for Cloud Storage* (Version 2.26.1). https://github.com/googleapis/java-storage

Google. (2023d). *Material Design components for Angular* (Version 16.1.8). https://material.angular.io/

Google. (2023e). *OSV-Scanner*. Verfügbar 28. August 2023 unter https://google.github.io/osv-scanner/

Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K. (2018). Do developers update their library dependencies? *Empirical Software Engineering*, *23*(1), 384–417. https://doi.org/10.1007/s10664-017-9521-5

Mozilla Developer Network. (2023). *SPA (Single-page application)*. Verfügbar 11. August 2023 unter https://developer.mozilla.org/en-US/docs/Glossary/SPA

NIST. (2021). *CVE-2021-44228*. Verfügbar 17. August 2023 unter https://nvd.nist.gov/vuln/detail/CVE-2021-44228

NTIA. (2021). *The Minimum Elements For a Software Bill of Materials (SBOM)*. Verfügbar 12. August 2023 unter https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

Ombredanne, P. (2023). *PURL Specification*. Verfügbar 28. August 2023 unter https://github.com/package-url/purl-spec

O'Neall, G. (2023). *Spdx-Java-Library* (Version 1.1.7). https://github.com/spdx/Spdx-Java-Library

OWASP Foundation. (n. d.). *OWASP CycloneDX*. Verfügbar 28. August 2023 unter https://cyclonedx.org/

OWASP Foundation. (2023a). *CycloneDX Core (Java)* (Version 7.3.2). https://github.com/CycloneDX/cyclonedx-core-java

OWASP Foundation. (2023b). *CycloneDX v1.5 JSON Reference*. Verfügbar 10. August 2023 unter https://cyclonedx.org/docs/1.5/json/#components_items_type

OWASP Foundation. (2023c). *CycloneDX v1.5 JSON Reference*. Verfügbar 12. August 2023 unter https://cyclonedx.org/docs/1.5/json/#components

Parmelee, M., Booth, H., Waltermire, D., & Scarfone, K. (2011). *Common Platform Enumeration: Name Matching Specification Version 2.3*. https://doi.org/10.6028/NIST.IR.7696

Preston-Werner, T. (n. d.). *Semantic Versioning 2.0.0*. Verfügbar 28. August 2023 unter https://semver.org/

SCANOSS. (2023). *PURL to CPE Relationship mapping project.* https://github.com/scanoss/purl2cpe

Sharma, S. (2019). *Mastering Microservices with Java - Third Edition.* Packt.

Sonatype. (2023). *8th Annual State of the Software Supply Chain Report.* Verfügbar 28. August 2023 unter https://www.sonatype.com/state-of-the-software-supply-chain/open-source-dependency-management-trends-and-recommendations

SOPHIST GmbH. (2016). *Schablonen für alle Fälle.* Verfügbar 1. Juli 2023 unter https://www.sophist.de/fileadmin/user_upload/Bilder_zu_Seiten/Publikationen/Wissen_for_free/MASTeR_Broschuere_3-Auflage_interaktiv.pdf

SPDX Workgroup. (2021). *Learn - Software Package Data Exchange (SPDX).* Verfügbar 28. August 2023 unter https://spdx.dev/resources/learn/

Springett, S. (2021). *Support for SPDX BOMs.* Verfügbar 15. August 2023 unter https://github.com/DependencyTrack/dependency-track/discussions/1222#discussioncomment-1466059

Springett, S. (2022). *New Recommendations to Improve The NVD.* Verfügbar 22. August 2023 unter https://owasp.org/blog/2022/09/13/sbom-forum-recommends-improvements-to-nvd

Springett, S. (2023a). *CPE Parser* (Version 2.0.2). https://github.com/stevespringett/CPE-Parser

Springett, S. (2023b). *Package URL (purl) for Java* (Version 1.4.0). https://github.com/package-url/packageurl-java

Springett, S. (2023c, 26. Mai). *Dependency-Track* (Version v4.8). https://github.com/DependencyTrack/dependency-track

Synopsys. (2023). *Open Source Security and Risk Analysis Report.* Verfügbar 17. August 2023 unter https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-ossra-2023.pdf

Tal, L. (2019). *The state of open source security report.* Verfügbar 17. August 2023 unter https://res.cloudinary.com/snyk/image/upload/v1551172581/The-State-Of-Open-Source-Security-Report-2019-Snyk.pdf

The Apache Software Foundation. (2023). *Maven Artifact* (Version 3.9.4). https://maven.apache.org/ref/3.9.4/maven-artifact/index.html

The Linux Foundation. (2022). *SPDX Specification v2.3.0 Annex F: External repository identifiers.* Verfügbar 18. August 2023 unter https://spdx.github.io/spdx-spec/v2.3/external-repository-identifiers/

VMware. (2023a). *Spring Boot* (Version 3.1.2). https://spring.io/projects/spring-boot

VMware. (2023b). *Spring Security* (Version 3.1.2). https://spring.io/projects/spring-security

Wetter, J., & Ringland, N. (2021). *Understanding the Impact of Apache Log4j Vulnerability.* Verfügbar 22. August 2023 unter https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html

Winslow, S. (2023). *SPDX and NTIA Minimum Elements for SBOM HOWTO*. Verfügbar 27. August 2023 unter https://spdx.github.io/spdx-ntia-sbom-howto