



WHZ Westsächsische
Hochschule Zwickau
Hochschule für Mobilität

BACHELORARBEIT

Entwicklung eines automatisierten Logfilter-Systems für Nextcloud und OpenSearch zur Identifikation und Entfernung sensibler Infrastrukturinformationen

Fakultät für Physikalische Technik und Informatik
Fachgruppe Informatik

Bachelorarbeit, vorgelegt von

Matthias Krüger

im April 2026

Betreuer: Prof. Dr. Frank Grimm, Westsächsische Hochschule Zwickau
Einrichtung: Hetzner Online GmbH, Falkenstein

Inhaltsverzeichnis

Selbstständigkeitserklärung	5
Abkürzungsverzeichnis	6
Danksagung	8
Zusammenfassung	9
1 Einleitung	10
1.1 Problemstellung	10
1.2 Motivation und Ziel der Arbeit	10
1.3 Forschungsfrage	11
2 Grundlagen	12
2.1 Logs	12
2.2 Nextcloud	12
2.3 OpenSearch	13
2.4 Pipes and Filters	14
2.5 Model-View-Controller	14
2.6 Model-View-Controller-Service	15
2.7 Composer	15
2.8 Reguläre Ausdrücke (Regex)	16
3 Konzeption	17
3.1 Anforderungsanalyse	17
3.2 Methodischer Ansatz	18
3.3 Architektur	19
3.4 Risiken und Schwierigkeiten	20
4 Implementierung	21
4.1 Systemüberblick	21
4.2 Datenmodell und Schnittstelle	22
4.3 Logdateienabruf	28

4.4	Filterpipeline.....	32
4.5	Filterklassen.....	36
4.6	Integration als Symfony Bundle.....	42
4.7	Tests.....	44
5	Evaluation und Validierung.....	55
5.1	Erfüllung der funktionalen Anforderungen.....	55
5.2	Erfüllung der nicht-funktionalen Anforderungen.....	56
5.3	Bekannte Einschränkungen.....	57
6	Fazit und Ausblick.....	58
6.1	Fazit.....	58
6.2	Ausblick.....	59
7	Quellenverzeichnis.....	61
8	Verwendete Hilfsmittel.....	64
A	Code des Logfiltersystems.....	65
A.1	FieldsToFilterConfig.....	65
A.2	LogEntry.....	71
A.3	LogFilterBundle.....	73
A.4	FilterPipeline.....	75
A.5	LogFilterService.....	77
A.6	OpenSearchApiClient.....	81
A.7	OpenSearchService.....	82
A.8	AbstractFilter.....	85
A.9	FilterInterface.....	87
A.10	CredentialsFilter.....	87
A.11	InfrastructureFilter.....	93
A.12	NetworkFilter.....	96
A.13	LogEntryTest.....	101
A.14	FilterPipelineTest.....	102

A.15 CredentialsFilterTest	107
A.16 InfrastructureFilterTest	123
A.17 NetworkFilterTest	127
A.18 Test-Skript	137
A.19 Config	140



Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit in allen Teilen selbstständig angefertigt und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt habe, und dass die Arbeit in gleicher oder ähnlicher Form in noch keiner anderen Prüfung vorgelegen hat. Mir ist bewusst, dass ich Autor/in der vorliegenden Arbeit bin und volle Verantwortung für den Text trage. ☒

Ich erkläre, dass ich wörtlich oder sinngemäß aus anderen Werken – dazu gehören auch Internetquellen – übernommene Inhalte als solche kenntlich gemacht und die entsprechenden Quellen angegeben habe. ☒

Mir ist bewusst, dass meine Arbeit auf Plagiate überprüft werden kann. Mir ist bekannt, dass es sich bei der Abgabe eines Plagiats um ein schweres akademisches Fehlverhalten handelt und dass Täuschungen nach der für mich gültigen Prüfungsordnung geahndet werden. ☒

Zusätzlich versichere ich, dass ich auf künstlicher Intelligenz (KI) basierende Werkzeuge nur in Absprache mit den Prüfern verwendet habe. Dabei stand meine eigene geistige Leistung im Vordergrund, und ich habe jederzeit den Prozess steuernd bearbeitet. ☒

Diese Werkzeuge habe ich im Quellenverzeichnis in der Rubrik „Übersicht verwendeter Hilfsmittel“ mit ihrem Produktnamen und einer Übersicht des im Rahmen dieser Prüfungs-/Studienarbeit genutzten Funktionsumfangs unter Angabe der Textstelle in der Arbeit vollständig aufgeführt. ☒

Ich versichere, dass ich keine KI-basierten Tools verwendet habe, deren Nutzung die Prüfer explizit schriftlich ausgeschlossen haben. Ich bin mir bewusst, dass die Verwendung von Texten oder anderen Inhalten und Produkten, die durch KI-basierte Tools generiert wurden, keine Garantie für deren Qualität darstellt. ☒

Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Passagen vollumfänglich selbst und trage die Verantwortung für eventuell durch die KI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate. ☒

20.04.2026

Ort, Datum


Unterschrift

Abkürzungsverzeichnis

API	Application Programming Interface
SDK	Software Development Kit
DSL	Domain Specific Language
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVCS	Model-View-Controller-Service
HTTP	Hypertext Transfer Protocol
FIG	Framework Interoperability Group
MAC	Media Access Control
REST	Representational State Transfer
SoC	Separation of Concerns
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
PSR	PHP Standards Recommendation
WebDAV	Web-based Distributed Authoring and Versioning acronym

Abbildungsverzeichnis

1	Überblick der Systemkomponenten	21
2	Ausführung sämtlicher PHPUnit-Tests des Logfiltersystems	48

Danksagung

Hiermit möchte ich mich bei Prof. Dr. Grimm für die Betreuung dieser Bachelorarbeit seitens der Hochschule bedanken.

Des Weiteren bedanke ich mich bei Florian Voit und Nadine Metzger für die Betreuung der Bachelorarbeit seitens der Hetzner Online GmbH.

Zusammenfassung

Die vorliegende Bachelorarbeit untersucht die Konzeption und Implementierung eines automatisierten Logfiltersystems für Nextcloud und OpenSearch. Ziel der Arbeit ist es, ein System zu entwickeln, das infrastrukturspezifische und sicherheitskritische Informationen in Logs zuverlässig erkennt und entfernt, ohne die technische Aussagekraft für die Fehleranalyse zu beeinträchtigen.

Zu Beginn wurden die für das Verständnis der Bachelorarbeit relevanten theoretischen und technischen Grundlagen erläutert sowie konkrete funktionale und nicht-funktionale Anforderungen an das System definiert. Anschließend erfolgte die Konzeption der Systemarchitektur. Das System wurde als wiederverwendbares Composer-Package implementiert, das Logeinträge zeitraumbasiert über die OpenSearch Scroll-API abrufen und durch eine modulare Filterpipeline verarbeitet. Die Filterung sensibler Informationen erfolgt durch zwei komplementäre Mechanismen. Zum einen eine feldnamenbasierte Direkterkennung für strukturierte Felder sowie eine musterbasierte Erkennung mittels vorkompilierter regulärer Ausdrücke für eingebettete Muster in Freitextfeldern.

Die Evaluation erfolgte anhand der durchgeführten Unit-Tests sowie eines Testlaufs gegen eine angebundene OpenSearch-Instanz. Die Ergebnisse zeigen, dass alle definierten Anforderungen erfüllt werden. Als Grenzen des Systems wurden die Abhängigkeit von vordefinierten Feldnamen sowie die eingeschränkte Erkennbarkeit sensibler Informationen in unstrukturierten Freitextfeldern ohne erkennbares Schlüssel-Wert-Muster identifiziert. Beide Einschränkungen sind der Natur musterbasierter Erkennung inhärent.

1 Einleitung

1.1 Problemstellung

Die Hetzner Online GmbH bietet mit den sogenannten Storage Shares einen Managed-Hosting-Dienst an, der die Bereitstellung und den Betrieb von Plattformen zur Dateiablage und Zusammenarbeit übernimmt. Unter einem Managed-Hosting-Dienst wird ein Hosting-Modell verstanden, bei dem der Anbieter neben der Bereitstellung der zugrunde liegenden Server- und Infrastrukturressourcen auch administrative Aufgaben wie Systembetrieb, Wartung, Sicherheitsupdates, Monitoring und grundlegende Konfigurationsmaßnahmen übernimmt. Kunden erhalten dabei Zugriff auf eine vollständig verwaltete Umgebung, während Aufgaben wie Systembetrieb, Wartung, Sicherheitsupdates und grundlegende Infrastrukturkonfigurationen durch den Anbieter erfolgen.

Innerhalb dieser Storage Shares kommt Nextcloud als zentrale Anwendungsplattform zum Einsatz.

Im Rahmen des Storage-Share-Hostings entstehen umfangreiche Logs, die sowohl für die Systemadministration als auch für die Fehlerbehebung und Sicherheitsüberwachung wichtig sind. Somit besteht seitens der Kunden häufig der Bedarf, Einsicht in die Logs ihrer Nextcloud-Instanz zu erhalten, um Anwendungsprobleme eigenständig analysieren oder Fehlverhalten nachvollziehen zu können.

Eine direkte Bereitstellung vollständiger und ungefilterter Logs ist jedoch problematisch, da diese möglicherweise sensible Informationen über die interne Infrastruktur des Hosting-Anbieters enthalten könnten. Dazu zählen beispielsweise interne Server- und Hostnamen, Netzwerkstrukturen oder Hinweise auf eingesetzte Sicherheitsmechanismen. Die Offenlegung solcher Daten kann Rückschlüsse auf die zugrunde liegende Systemarchitektur ermöglichen und daher das Risiko gezielter Angriffe erhöhen.

Die zentrale Herausforderung dieser Arbeit besteht daher darin, ein geeignetes Verhältnis zwischen der Bereitstellung technisch relevanter Loginformationen für Kunden und dem Schutz der infrastrukturenspezifischen und sicherheitskritischen Informationen zu schaffen.

1.2 Motivation und Ziel der Arbeit

Ziel dieser Arbeit ist die Konzeption und Implementierung eines Systems zur Analyse und Filterung von Logs aus Nextcloud-Instanzen [1]. Logs stellen eine zentrale Grundlage für die Untersuchung von Systemverhalten sowie für die Diagnose von Fehlerursachen dar [2], weshalb ihre strukturierte und gezielte Aufbereitung von großer Bedeutung ist. Im Rahmen dieser Arbeit soll daher ein System entwickelt werden, das sensible Informationen zuverlässig erkennt und entfernt, ohne dabei die technische Aussagekraft der Daten für die Fehleranalyse wesentlich zu beeinträchtigen.

Zu diesem Zweck wird eine Anwendung auf Basis des Symfony-Frameworks [3] entwickelt, die Logs aus einer OpenSearch-Instanz bezieht [4]. Die Verarbeitung erfolgt dabei abhängig vom jeweiligen Log-Typ. In dieser Arbeit werden insbesondere folgende Log-Typen betrachtet:

- Nextcloud-Logs
- NGINX-Logs

Für jeden Log-Typ sollen geeignete Filter- und Transformationsmechanismen implementiert werden, um eine sichere und zugleich nutzbare Bereitstellung der Logs für Kunden zu gewährleisten.

1.3 Forschungsfrage

Aus diesen Anforderungen ergibt sich die folgende Forschungsfrage:

"Wie kann ein automatisiertes, erweiterbares Filtersystem für Nextcloud und OpenSearch implementiert werden, das infrastrukturspezifische und sicherheitskritische Informationen in Logs zuverlässig erkennt und entfernt, ohne die technische Aussagekraft der Logs zu beeinträchtigen?"

2 Grundlagen

Dieses Kapitel beschreibt die theoretischen und technischen Grundlagen, die für das Verständnis dieser Arbeit erforderlich sind. Es werden zentrale Konzepte, Werkzeuge und Architekturprinzipien erläutert, auf denen das Logfiltersystem aufbaut.

2.1 Logs

Logs dienen der Protokollierung von Systemereignissen und stellen eine zentrale Grundlage für Fehleranalysen, Systemüberwachung und sicherheitsrelevante Untersuchungen dar. Sie dokumentieren technische Abläufe in chronologischer Reihenfolge und enthalten typischerweise essenzielle Informationen zu Fehlermeldungen, Anwendungsereignissen, Zugriffen und sicherheitsrelevanten Vorgängen.

Nach Nadine Rücker (2022) versteht man unter Logs *„computergenerierte Textdateien, die während der Laufzeit eines Systems erstellt werden und eine große Menge an Informationen über den Systembetrieb enthalten. Sie werden häufig verwendet, um Erkenntnisse über Fehler, Nutzung, Betrugsfälle und weitere Ereignisse zu gewinnen [2] [Übers. durch den Verf.]“*

Logs können sowohl als strukturierte JSON-Objekte als auch als einfache Textzeilenformate vorliegen. Aufgrund ihres Informationsgehalts können Logs neben technisch relevanten Daten auch sensible Informationen wie Benutzererkennungen oder interne Systeminformationen enthalten, wodurch sich die Notwendigkeit ergibt, die enthaltenen Informationen gegebenenfalls zu filtern. In komplexen Anwendungen zur Dateiablage und Synchronisation wie Nextcloud entstehen daher umfangreiche Logs, die gefiltert werden müssen, um sensible Informationen zu schützen.

2.2 Nextcloud

Nach Kariyattin u. a. (2018) bezeichnet Nextcloud *„eine selbst gehostete Open-Source Plattform zur Dateifreigabe und Kommunikation, die einen einheitlichen Zugriff auf Dateien aus unterschiedlichen Speicherquellen ermöglicht. Sie führt Daten aus Cloud-Speichern, Windows-Netzlaufwerken und älteren Speichersystemen in einer gemeinsamen, benutzerfreundlichen Oberfläche zusammen, sodass Nutzer Dateien geräteunabhängig abrufen, synchronisieren und teilen können [1] [Übers. durch den Verf.]“*.

Die Plattform wird als PHP-Webanwendung auf einem Linux-Webserver wie Apache [5] oder NGINX [6] betrieben. Benutzerinformationen, Konfigurationen und Dateimetadaten werden in einer relationalen Datenbank wie MySQL [7] gespeichert, während die eigentlichen Dateien in herkömmlichen Verzeichnisstrukturen abgelegt und über das WebDAV-Protokoll zugänglich gemacht werden [1].

Die von Nextcloud erzeugten Logs enthalten zahlreiche technisch relevante und sensible Informationen, deren sichere Aufbereitung und gezielte Filterung eine

strukturierte Speicherung und effiziente Abfrage erfordert. Hierfür kommt OpenSearch zum Einsatz, eine leistungsfähige Open-Source-Software zur Suche und Analyse von Daten, die eine schnelle, skalierbare und flexible Verarbeitung von Logs ermöglicht.

2.3 OpenSearch

OpenSearch dient der strukturierten Speicherung, Indizierung und Analyse großer Datenmengen, insbesondere von Log- und Ereignisdaten.

Saravanan Kuppusamy (2025) beschreibt OpenSearch als *„eine Open-Source Such- und Analysesoftware, die auf Apache Lucene basiert und für horizontale Skalierbarkeit, Zuverlässigkeit und Echtzeitsuche entwickelt wurde. Sie ermöglicht komplexe Suchabfragen über unterschiedlichste Datentypen hinweg und ist bekannt für ihre leistungsfähige Volltextsuche, strukturierte Suche, Analysefunktionen sowie ihre benutzerfreundliche Handhabung. [4] [Übers. durch den Verf.]“*.

Die in OpenSearch gespeicherten Daten werden indexiert, wodurch eine performante und nahezu echtzeitfähige Verarbeitung ermöglicht wird. Auf dieser Grundlage können Informationen effizient gefiltert, durchsucht und ausgewertet werden. Typische Anwendungsfälle dafür sind unter anderem die Fehleranalyse, das Monitoring von IT-Systemen sowie die Auswertung von Ereignis- und Protokolldaten.

Für die Verarbeitung großer Ergebnismengen stellt OpenSearch die sogenannte Scroll-API bereit. Dieses Verfahren erlaubt es, umfangreiche Suchergebnisse schrittweise abzurufen, anstatt sie vollständig in einer einzelnen Antwort zu bearbeiten. Dadurch wird die Speicherbelastung reduziert und eine stabile sowie eine ressourcenschonende Weiterverarbeitung der Daten ermöglicht.

Die eigentliche Kommunikation mit OpenSearch erfolgt über eine HTTP-basierte REST-API. Suchanfragen werden mithilfe der JSON-basierten Abfragesprache Query DSL formuliert. Diese ermöglicht die präzise Definition komplexer Such- und Filterkriterien und bildet die Grundlage für strukturierte sowie volltextbasierte Abfragen.

Query DSL

Die Query DSL von OpenSearch ist eine JSON-basierte Abfragesprache zur Formulierung strukturierter sowie volltextbasierter Suchanfragen. Sie ermöglicht die Kombination unterschiedlicher Abfragearten und die präzise Definition von Such- und Filterkriterien [8].

Für die zeitliche Eingrenzung von Logs wird häufig der sogenannte `range`-Filter verwendet. Dieser ermöglicht die Filterung von Dokumenten anhand eines definierten Wertebereichs innerhalb eines bestimmten Feldes, beispielsweise eines Zeitstempels. Zur Definition des Intervalls stehen Operatoren wie `gt` (Greater Than), `lt` (Less Than), `gte` (Greater Than or Equal) oder `lte` (Less Than or Equal) zur Verfügung [9]. Auf diese Weise können Logeinträge gezielt innerhalb eines bestimmten Zeitraums abgefragt werden.

Scroll-API

OpenSearch begrenzt die Anzahl der Treffer, die bei einer einzelnen Suchanfrage zurückgegeben werden, standardmäßig auf eine konfigurierbare Höchstzahl. Um dennoch große Ergebnismengen effizient zu verarbeiten, stellt OpenSearch die Scroll-API bereit. Diese ermöglicht es, Suchergebnisse schrittweise abzurufen, indem ein auf dem Server gespeicherter Suchkontext genutzt wird, der für nachfolgende Anfragen aktiv bleibt [10].

Bei der initialen Suchanfrage wird der Parameter `scroll` angegeben, der festlegt, wie lange der Suchkontext bestehen bleiben soll. OpenSearch legt daraufhin den Suchkontext an und liefert zusammen mit den ersten Ergebnissen eine eindeutige `scroll_id` zurück. Diese `scroll_id` muss in allen folgenden Scroll-Anfragen übermittelt werden, um weitere Batches aus demselben Suchkontext abzurufen. Jede neue Anfrage erzeugt eine neue `scroll_id`, die die vorherige ersetzt. Sobald der Suchkontext abläuft, können keine weiteren Ergebnisse abgerufen werden [10].

2.4 Pipes and Filters

Da die Filterung und Weiterverarbeitung der abgerufenen Daten typischerweise nicht in einem einzelnen Verarbeitungsschritt erfolgt, sondern mehrere aufeinanderfolgende Transformationen umfasst, ist eine geeignete Architektur erforderlich. Ein verbreiteter Ansatz hierfür ist das Pipes-and-Filters Architekturmuster.

Nach Doberkat (2002) beschreibt Pipes-and-Filters „eine verbreitete Architektur, bei der rechnerische Komponenten (Filter) über Verbindungselemente (Pipes) miteinander verbunden werden, sodass Berechnungen in einer stream-artigen Weise erfolgen. Die Daten werden durch die Pipes und Filter transportiert und dabei schrittweise von Eingaben (Inputs) zu Ausgaben (Outputs) transformiert. [11] [Übers. durch den Verf.]“.

Ein wesentliches Merkmal dieses Architekturmusters ist die klare Trennung der Verantwortlichkeiten. Jeder Filter erfüllt eine spezifische Aufgabe, wodurch komplexe Gesamtprozesse in überschaubare Einzelschritte zerlegt werden können. Diese Modularisierung fördert die Verständlichkeit, Wartbarkeit und Erweiterbarkeit eines Systems, da einzelne Komponenten unabhängig voneinander betrachtet und angepasst werden können.

2.5 Model-View-Controller

Während Pipes-and-Filters die Strukturierung von Datenverarbeitungsprozessen betrachtet, konzentriert sich das Model-View-Controller (MVC) Architekturmuster auf die systematische Aufteilung einer Anwendung in klar abgegrenzte Verantwortungsbereiche: Model, View und Controller. Ziel ist eine saubere Trennung der Verantwortungsbereiche nach dem Prinzip der Separation of Concerns (SoC), um Wartbarkeit, Erweiterbarkeit und Wiederverwendbarkeit zu verbessern [12]

Das Model repräsentiert den fachlichen Zustand des Systems sowie die Regeln zur Sicherstellung der Datenkonsistenz. Es ist unabhängig von Darstellung und

Benutzerinteraktion und stellt definierte Schnittstellen zur Verfügung, über die andere Komponenten auf seine Daten zugreifen können [13].

Die View ist an das Model gebunden und übernimmt ausschließlich die Darstellung seiner Inhalte. Sie enthält keine eigenständige fachliche Logik, sondern bezieht ihre Informationen über definierten Abfragen vom Model [13].

Der Controller koordiniert den Anwendungsablauf. Er empfängt Benutzereingaben, verarbeitet diese und initiiert entsprechende Änderungen am Model sowie Aktualisierungen der View [13].

2.6 Model-View-Controller-Service

Das Model-View-Controller-Service (MVCS) Architekturmuster stellt eine Weiterentwicklung des klassischen MVC Architekturmusters dar. Während das MVC eine klare Trennung zwischen Model, View und Controller vorsieht, ergänzt MVCS diese Struktur um eine zusätzliche Service-Schicht zur Kapselung fachlicher Geschäftslogik. Ziel dieser Erweiterung ist es, die Verantwortlichkeiten innerhalb eines Softwaresystems noch präziser zu trennen und insbesondere komplexere fachliche Prozesse aus dem Controller auszulagern.

Service

Die Service-Schicht ist innerhalb des MVCS eine eigene Ebene, die die fachliche Logik der Anwendung bündelt. Sie liegt konzeptionell zwischen dem Controller und dem Model und sorgt dafür, dass komplexe Abläufe übersichtlich und klar strukturiert umgesetzt werden. Dadurch werden fachliche Prozesse getrennt von Controller und View in eigenständige, klar abgegrenzte Funktionseinheiten gegliedert.

2.7 Composer

Composer ist ein Abhängigkeitsverwaltungswerkzeug für PHP-Projekte [14]. Es löst externe Bibliotheken und deren Abhängigkeiten auf, lädt sie herunter und registriert sie im Autoloading-System des Projekts. Abhängigkeiten werden dabei deklarativ in einer `composer.json` Datei beschrieben.

Ein zentraler Bestandteil von Composer ist der Autoloader [15], der sicherstellt, dass die heruntergeladenen Bibliotheken und deren Klassen automatisch geladen werden, ohne dass explizite `require`- oder `include`-Anweisungen im Quellcode notwendig sind. Composer generiert hierfür eine `autoload.php` Datei, die einmalig zu Beginn eines Skripts eingebunden wird. Die Zuordnung von Namespaces zu Verzeichnissen erfolgt standardmäßig nach PSR-4, einem von der PHP Framework Interoperability Group (FIG) definierten Standard, der eine einheitliche und strukturierte Klassenverwaltung gewährleistet [16]. Die über Composer verwalteten Abhängigkeiten werden dabei als sogenannte Packages bereitgestellt.

Ein Composer-Package stellt eine durch Composer verwaltete Einheit dar, die typischerweise als Verzeichnis mit PHP-Code sowie einer zugehörigen Paketbeschreibung vorliegt. Diese Beschreibung enthält unter anderem einen Namen und eine Version, die der eindeutigen Identifikation dienen. Darüber hinaus umfasst ein Package Metadaten, wie etwa die Angabe der Bezugsquelle, über die die Inhalte entweder als Distributionsarchiv oder als Quellcode bereitgestellt werden. Packages werden lokal installiert und können als wiederverwendbare Abhängigkeiten in verschiedenen PHP-Projekten eingesetzt werden [17].

2.8 Reguläre Ausdrücke (Regex)

Reguläre Ausdrücke (Regex) sind eine formale Notation zur Beschreibung von Mustern in Zeichenketten. Nach Friedl (2008) sind reguläre Ausdrücke *„eine generelle Notation zur Beschreibung von Textmustern, beinahe wie eine kleine Programmiersprache zum Prüfen und zum Manipulieren von Texten. [18]“*. Sie ermöglichen die deklarative Spezifikation von Mengen von Zeichenketten und werden in der praktischen Anwendung insbesondere zur Suche, Validierung, Extraktion und Ersetzung von Textinhalten eingesetzt.

Formal beschreiben reguläre Ausdrücke Mengen von Zeichenketten und können damit als Spezifikation einer formalen Sprache verstanden werden.

3 Konzeption

Dieses Kapitel beschreibt die methodische Vorgehensweise zur Konzeption des Logfiltersystems. In Die konkrete technische Umsetzung, einschließlich der verwendeten Klassen, Schnittstellen und Datenstrukturen, wird im Kapitel Implementierung genauer erläutert.

3.1 Anforderungsanalyse

Im Rahmen der Konzeption des Logfiltersystems werden die Anforderungen in funktionale und nicht-funktionale Anforderungen unterteilt. Diese strukturierte Anforderungsanalyse bildet dabei die Grundlage für die spätere Systementwicklung, da sie sicherstellt, dass sowohl die gewünschten Funktionen als auch die Qualitätseigenschaften des Systems klar definiert sind, bevor mit der Implementierung begonnen wird. Die folgenden definierten funktionalen und nicht-funktionalen Anforderungen wurden auf Grundlage der Zielsetzung abgeleitet und mit der Hetzner Online GmbH hinsichtlich ihrer fachlichen Relevanz und praktischen Anwendbarkeit abgestimmt. Sie bilden damit die im Rahmen dieser Arbeit zugrunde gelegten und mit dem Praxispartner vereinbarten Anforderungen an das entwickelte Logfiltersystem.

Funktionale Anforderungen

Funktionale Anforderungen sind nach Chazette (2023) "detaillierte Aussagen zu den Fähigkeiten, dem Verhalten und den Informationen, die das System bereitstellen muss. Sie beschreiben, was ein System tun soll. [19] [Übers. durch den Verf.]". Im Folgenden werden die funktionalen Anforderungen für das entwickelte Logfiltersystem spezifiziert.

FA-01 (Zeitraumbasierter Abruf)

Das System muss Logeinträge aus einer OpenSearch-Instanz abrufen können, deren Zeitstempel innerhalb eines vom Benutzer definierten Zeitraums liegen. Voraussetzung ist, dass ein gültiges Start- und Enddatum im ISO-8601-Format [20] angegeben wird und das Startdatum vor dem Enddatum liegt.

FA-02 (Filterung strukturierter Felder)

Das System muss sensible Informationen in strukturierten Logfeldern identifizieren und deren Werte durch definierte Platzhalter ersetzen können. Die zu filternden Feldnamen müssen konfigurierbar sein, um unterschiedliche Logstrukturen unterstützen zu können.

FA-03 (Filterung unstrukturierter Inhalte)

Das System muss sensible Informationen innerhalb unstrukturierter Freitextfelder erkennen und filtern können. Die Erkennung erfolgt auf Grundlage regulärer Ausdrücke, um auch eingebettete Muster in Logs zu berücksichtigen.

FA-04 (Strukturierte Ausgabe)

Das System muss die gefilterten Logeinträge in einem konsistenten JSON-Format

bereitstellen, um eine Weiterverarbeitung oder externe Nutzung der Ergebnisse zu ermöglichen.

FA-05 (Verarbeitungsstatistik)

Das System muss nach Abschluss der Verarbeitung eine Statistik über verarbeitete Logeinträge sowie erkannte und gefilterte Inhalte bereitstellen. Dies dient der Überprüfung von Qualität und Vollständigkeit der Filterung.

Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen hingegen beschreiben Eigenschaften oder Einschränkungen eines Systems. Sie definieren die Bedingungen, unter denen eine Lösung wirksam bleiben muss, sowie die Qualitätsmerkmale, die die Lösung erfüllen soll. [19] [Übers. durch den Verf.]. Zur Definition der nicht-funktionalen Anforderungen wurde sich an der ISO/IEC 25010 orientiert, die ein etabliertes Qualitätsmodell für Softwareprodukte bereitstellt und zentrale Qualitätsmerkmale wie Wartbarkeit, Performance und Testbarkeit strukturiert beschreibt [21]. Im Folgenden werden die nicht-funktionalen Anforderungen für das entwickelte Logfiltersystem spezifiziert.

NFA-01 (Erweiterbarkeit)

Das System muss so gestaltet sein, dass neue Filterkategorien durch Hinzufügen neuer Filterklassen integriert werden können, ohne bestehende Komponenten ändern zu müssen.

(Zuordnung: ISO/IEC 25010 [21] - Maintainability / Modifiability)

NFA-02 (Performance)

Das System muss auch bei großen Mengen an Logeinträgen eine Verarbeitungsgeschwindigkeit von mindestens 1000 Log-Einträgen pro Sekunde aufrechterhalten können.

(Zuordnung: ISO/IEC 25010 [21] - Performance Efficiency / Time Behaviour)

NFA-03 (Testbarkeit)

Die Filterklassen müssen isoliert testbar sein, ohne dass für die Durchführung von Unit-Tests eine aktive OpenSearch-Instanz erforderlich ist.

(Zuordnung: ISO/IEC 25010 [21] - Maintainability / Testability)

NFA-04 (Wartbarkeit)

Der Quellcode muss klar strukturiert und dokumentiert sein, sodass Anpassungen und Erweiterungen ohne tiefgreifendes Vorwissen über die gesamte Codebasis vorgenommen werden können.

(Zuordnung: ISO/IEC 25010 [21] - Maintainability / Modifiability)

3.2 Methodischer Ansatz

Ziel ist die systematische Ableitung einer geeigneten Architektur sowie die methodisch kontrollierte Umsetzung der Filtermechanismen. Hierbei werden die betrachteten Log-Typen hinsichtlich ihrer syntaktischen Struktur, ihrer Informationen sowie

wiederkehrender Muster untersucht. Die Analyse bildet die Grundlage für die spätere Implementierung der Filterlogik, indem relevante Feldnamen und geeignete reguläre Ausdrücke zur Erkennung sensibler Informationen identifiziert und spezifiziert werden. Auf dieser Basis wird eine Trennung zwischen gewöhnlichen und infrastrukturspezifischen beziehungsweise sicherheitskritischen Informationen vorgenommen. Aus der Analyse der Log-Strukturen ergibt sich, dass sensible Informationen in zwei grundlegend unterschiedlichen Formen auftreten können. Zum einen als eigenständige strukturierte Felder, deren Feldname direkt auf einen sensiblen Inhalt hinweist, zum anderen als eingebettete Muster innerhalb von Freitextfeldern oder zusammengesetzten Zeichenketten.

Diese Unterscheidung bildet die Grundlage für die Wahl zweier komplementärer Erkennungsmechanismen: eine feldnamenbasierte Direkterkennung sowie eine musterbasierte Erkennung mittels regulärer Ausdrücke, deren formale Eigenschaften eine positionsunabhängige Mustererkennung innerhalb von Zeichenketten ermöglichen. Beide Mechanismen werden für jede Filterkategorie kombiniert eingesetzt, um eine möglichst vollständige Abdeckung zu gewährleisten.

3.3 Architektur

MVCS

Die grundlegende Architektur orientiert sich an dem MVCS Architekturmuster. Diese Entscheidung ergibt sich aus der Anforderung, Geschäftslogik unabhängig vom Anfragekontext testbar und wiederverwendbar zu halten. Die Auslagerung der Filter- und Abruflogik in eigenständige Services stellt sicher, dass diese Komponenten isoliert geprüft werden können, ohne eine bestimmte Laufzeitumgebung vorauszusetzen.

Da das System als wiederverwendbares Composer-Package für unterschiedliche Anwendungen konzipiert ist, sind Controller und View allerdings nicht Bestandteil des Composer-Package. Spezifische Eingaben und in welcher Form Ergebnisse ausgegeben werden sollen, hängen vom Verwendungskontext der einbindenden Anwendung ab und sollten daher nicht innerhalb des Composer-Package festgelegt sein. Somit werden ausschließlich das Model und die verschiedenen Services bereitgestellt.

Pipes-and-Filters

Für die Filterlogik innerhalb der Serviceschicht wird das Pipes-and-Filters Architekturmuster eingesetzt. Die Entscheidung ergibt sich unmittelbar aus der Anforderung an die Erweiterbarkeit. Da die zu filternden Informationskategorien thematisch klar voneinander abgrenzbar sind, lässt sich jede Kategorie als eigenständiger Filter realisieren, der unabhängig von den übrigen aktiviert, deaktiviert oder ergänzt werden kann.

Somit definiert MVCS die übergeordnete Schichtenstruktur des Gesamtsystems, während Pipes-and-Filters die interne Ablaufstruktur innerhalb der Serviceschicht beschreibt.

3.4 Risiken und Schwierigkeiten

Im Rahmen der Konzeption und Umsetzung des in dieser Arbeit vorgestellten Systems, ergeben sich verschiedene Herausforderungen, die den Entwicklungsprozess beeinflussen können. Die nachfolgenden Punkte beschreiben zentrale Risiken:

R-01 (unvollständige Filterung)

Eine wesentliche Schwierigkeit besteht darin, dass nicht in allen Fällen sämtliche infrastrukturspezifischen und sicherheitskritischen Informationen innerhalb der Logs zuverlässig identifiziert und gefiltert werden können. Insbesondere bei komplexen oder bislang unbekannten Logstrukturen besteht das Risiko, dass sensible Details unbeabsichtigt in den bereitgestellten Logeinträgen verbleiben.

R-02 (zu starke Filterung)

Es besteht das Risiko, dass durch eine zu restriktive Filterung technisch relevante Loginformationen entfernt werden. Dies kann dazu führen, dass Anwendungsfehler oder unerwartetes Systemverhalten nicht mehr vollständig nachvollzogen werden kann.

R-03 (unterschiedliche Log-Formate)

Die in dieser Arbeit betrachteten Log-Typen weisen unterschiedliche Strukturen und Formate auf. Dies erschwert die Entwicklung einheitlicher Analyse- und Filtermechanismen.

R-04 (Performance)

Eine weitere Herausforderung stellt die Verarbeitung großer Mengen an Logs dar, da das Logfiltersystem auch bei großen Mengen an Logeinträgen eine angemessene Verarbeitungszeit gewährleisten muss.

R-05 (zeitliche Beschränkung)

Die Bearbeitungszeit der Arbeit ist auf einen Zeitraum von zehn Wochen begrenzt [22], was den Umfang der Implementierung einschränkt. Entsprechend ist eine vollständige Abdeckung aller denkbaren Log-Typen, Sonderfälle und Optimierungsansätze nur eingeschränkt möglich.

4 Implementierung

Dieses Kapitel beschreibt die technische Umsetzung des Logfiltersystems. Es wird dargelegt, wie die einzelnen Komponenten realisiert wurden, wie sie zusammenwirken und welche konkreten Implementierungsentscheidungen getroffen wurden.

4.1 Systemüberblick

Das Logfiltersystem ist modular aufgebaut und folgt einer klaren Verantwortungstrennung. Die einzelnen Komponenten sind so gestaltet, dass sie unabhängig voneinander entwickelt, getestet und bei Bedarf erweitert oder ausgetauscht werden können.

Für die Implementierung des Logfiltersystems wurde das Symfony-Framework gewählt. Symfony ist ein verbreitetes PHP-Framework, das sich durch hohe Modularität, Flexibilität und eine klare Strukturierung des Codes auszeichnet. In der vergleichenden Analyse von Laaziri (2018) wird Symfony als etablierter Vertreter moderner PHP-Frameworks beschrieben, der eine breite und robuste Architektur bietet, welche für die Entwicklung komplexer Webanwendungen geeignet ist [3].

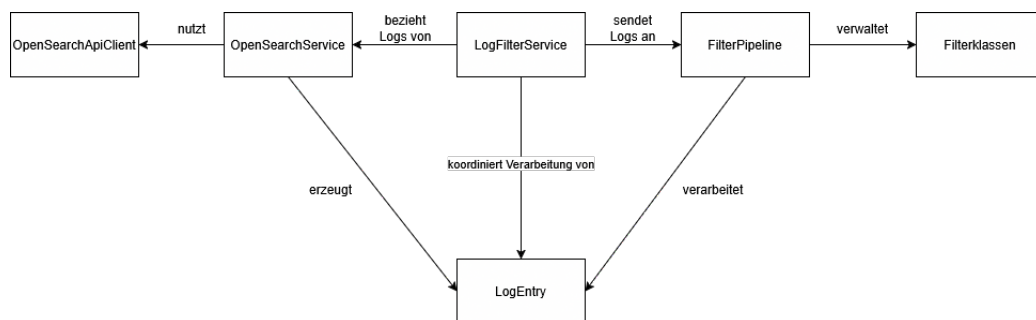


Abbildung 1: Überblick der Systemkomponenten

Im Zentrum des Logfiltersystems steht der **LogFilterService**, der als Koordinator fungiert. Er stellt die zentrale Einstiegsschnittstelle dar und koordiniert den gesamten Ablauf der Logverarbeitung. Der **LogFilterService** nimmt die definierten Parameter wie den zu betrachteten Zeitraum der Logeinträge entgegen und initiiert daraufhin den Abruf sowie die anschließende Filterung der Logeinträge. Dabei übergibt er die eigentlichen Aufgaben an die spezialisierten Komponenten.

Für den Zugriff auf die Logeinträge ist der **OpenSearchService** verantwortlich. Diese Komponente kapselt die gesamte Kommunikation mit der OpenSearch-Instanz und abstrahiert die Details der zugrunde liegenden API. Basierend auf den übergebenen Parametern erstellt sie die entsprechende Query und nutzt die Scroll-API, um große Datenmengen effizient zu verarbeiten. Die Rückgabe erfolgt in Form eines PHP-Generators, wodurch die Logeinträge sequenziell verarbeitet werden können, ohne den Arbeitsspeicher unnötig zu belasten.

Zur Umsetzung der technischen Kommunikation mit OpenSearch wird der **OpenSearchApiClient** eingesetzt. Dieser stellt die eigentliche technische Schnittstelle

zur OpenSearch-API dar und übernimmt die Durchführung der HTTP- bzw. API-Aufrufe. Der `OpenSearchService` nutzt diesen Client ausschließlich für die Kommunikation mit OpenSearch, wodurch die fachliche Logik klar von der technischen Kommunikationsschicht getrennt wird.

Die vom `OpenSearchService` gelieferten Logeinträge werden als `LogEntry`-Objekte gekapselt. Diese dienen als strukturierte Repräsentation eines einzelnen Logeintrags und bilden die Grundlage für die weitere Verarbeitung. Durch die Kapselung in ein Objekt wird ein einheitlicher Zugriff auf die enthaltenen Daten ermöglicht, unabhängig von der ursprünglichen Datenstruktur.

Die eigentliche Filterlogik ist in der `FilterPipeline` gekapselt. Sie verwaltet eine konfigurierbare Liste von Filterklassen und sorgt dafür, dass jeder Logeintrag nacheinander durch alle registrierten Filter verarbeitet wird. Dieses Vorgehen entspricht dem Pipes-and-Filters Architekturmuster und ermöglicht somit eine flexible Erweiterung des Systems, da neue Filter ohne Anpassung bestehender Komponenten integriert werden können.

Die Konfiguration des Systems erfolgt über eine zentrale Konfigurationsdatei `config.local`. Sie enthält Laufzeitparameter für die OpenSearch-Integration sowie die Verarbeitung von Logs. Dazu zählen Verbindungsparameter (URL, Benutzername, Passwort, Indexname) sowie Steuerparameter für die Verarbeitung wie die Batch-Größe und das Scroll-Intervall [10]. Der vollständige Code dieser Klasse ist im Anhang A.19 einsehbar.

Die einzelnen Filterklassen sind für die Erkennung und Filterung der sensiblen Informationen innerhalb der Logeinträge verantwortlich. Hierbei kommen zwei komplementäre Mechanismen zum Einsatz: Die feldbasierte Filterung, bei der bekannte sensible Felder gezielt identifiziert und deren Werten ersetzt werden und die musterbasierte Filterung mittels regulärer Ausdrücke, um sensible Informationen innerhalb von Freitextfeldern zu erkennen.

Während der Verarbeitung der Logeinträge werden zusätzlich statistische Informationen erfasst, beispielsweise die Anzahl verarbeiteter Einträge oder erkannter sensibler Informationen. Nach Abschluss der Verarbeitung werden die gefilterten Logeinträge in strukturierter Form zusammen mit der Verarbeitungsstatistik zurückgegeben.

Der Gesamtprozess lässt sich somit als sequenzielle Verarbeitungskette beschreiben. Der `LogFilterService` initiiert den Ablauf, der `OpenSearchService` liefert die unverarbeiteten Logeinträge, diese werden als `LogEntry`-Objekte gekapselt und anschließend durch die `FilterPipeline` verarbeitet, in der die eigentliche Filterung durch die einzelnen Filterklassen erfolgt. Durch dieses Zusammenspiel wird eine flexible, erweiterbare und wartbare Systemarchitektur realisiert.

4.2 Datenmodell und Schnittstelle

Die Anwendung ist entsprechend dem MVCS Architekturmuster in voneinander abgegrenzte Schichten gegliedert, wobei jede Schicht eine klar definierte

Verantwortung trägt und von den übrigen Schichten entkoppelt ist. Diese Trennung stellt sicher, dass Änderungen an einer Komponente keine unbeabsichtigten Auswirkungen auf andere Teile des Systems haben.

LogEntry

Das Model wird durch die Klasse `LogEntry` repräsentiert. Sie kapselt die Daten eines einzelnen Logeintrags und stellt standardisierte Zugriffsmethoden bereit. Dadurch wird sichergestellt, dass alle weiteren Komponenten unabhängig vom Ursprungsformat mit einer einheitlichen Datenstruktur arbeiten. Sie speichert zwei Pflichtattribute, eine eindeutige `$id` sowie ein assoziatives Array `$data`, das den Inhalt des Logeintrags enthält. Beide Attribute werden über den Konstruktor gesetzt und sind ausschließlich über definierte Methoden zugänglich.

LogEntry Konstruktor

```
1 public function __construct(private readonly string $id,  
2     private array $data)  
3 {  
4 }  
5 }
```

Der Zugriff auf die Kerndaten des Eintrags erfolgt über die Methoden `getId()`, `getData()` und `setData()`. Während `getId()` die eindeutige Identifikation zurückgibt, ermöglicht `getData()` und `setData()` das vollständige Lesen und Überschreiben des Datenarrays.

LogEntry getId(), getData() und setData() Methoden

```
1 public function getId(): string  
2 {  
3     return $this->id;  
4 }  
5  
6 public function getData(): array  
7 {  
8     return $this->data;  
9 }  
10  
11 public function setData(array $data): void  
12 {  
13     $this->data = $data;  
14 }
```

Für den gezielten Zugriff auf einzelne Felder innerhalb des Datenarrays stehen die Methoden `getField()`, `setField()` und `hasField()` zur Verfügung. `getField()` nimmt einen optionalen Standardwert entgegen, der zurückgegeben wird, sofern das gesuchte Feld nicht existiert. `setField()` setzt den Wert eines einzelnen Feldes, ohne das gesamte Datenarray ersetzen zu müssen und `hasField()` prüft, ob ein bestimmtes Feld im Array vorhanden ist.

LogEntry getField(), setField() und hasField() Methoden

```
1 public function getField(string $key, mixed $default = null):  
    mixed  
2 {  
3     return $this->data[$key] ?? $default;  
4 }  
5  
6 public function setField(string $key, mixed $value): void  
7 {  
8     $this->data[$key] = $value;  
9 }  
10  
11 public function hasField(string $key): bool  
12 {  
13     return isset($this->data[$key]);  
14 }
```

Die Methode `toArray()` serialisiert das gesamte `LogEntry`-Objekt in ein einfaches assoziatives Array, bestehend aus `id` und `data`. Diese Darstellung wird am Ende der Verarbeitung verwendet, um die gefilterten Logeinträge in einem OpenSearch-kompatiblen Format zurückzugeben.

LogEntry toArray() Methode

```
1 public function toArray(): array  
2 {  
3     return  
4     [  
5         'id' => $this->id,  
6         'data' => $this->data  
7     ];  
8 }
```

Die `LogEntry`-Struktur bildet damit die Grundlage für die weitere Verarbeitung innerhalb der Service-Schicht, deren zentraler Einstiegspunkt der `LogFilterService` ist.

Der vollständige Code dieser Klasse ist im Anhang A.2 einsehbar.

LogFilterService

Der `LogFilterService` stellt die zentrale Einstiegsschnittstelle zur Logverarbeitung bereit. Er kapselt den vollständigen Verarbeitungsprozess und ist unabhängig von einem konkreten Anfrage- oder Darstellungskontext.

Im Konstruktor von `LogFilterService` werden der `OpenSearchService`, die `FilterPipeline` sowie die Batch-Größe entgegengenommen. Der `OpenSearchService` wird benötigt, um Logeinträge aus OpenSearch abzurufen, während die `FilterPipeline` für die schrittweise Filterung der Logeinträge zuständig

ist. Die Batch-Größe legt fest, wie viele Einträge pro Batch gesammelt werden können und wird als direkter Parameter übergeben und intern gespeichert.

LogFilterService Konstruktor

```
1 public function __construct(private OpenSearchService
    $openSearchService, private FilterPipeline $filterPipeline,
    int $batchSize)
2 {
3     $this->batchSize = $batchSize;
4 }
```

Die zentrale Methode `processLogs()` steuert den Verarbeitungsablauf.

Vor dem Abruf der Logeinträge wird über `assertValidRange()` sichergestellt, dass der angegebene Zeitraum gültig ist, indem überprüft wird, ob das Startdatum vor dem Enddatum liegt. Ist diese Bedingung verletzt, wird eine `InvalidArgumentException` ausgelöst. Auf diese Weise wird eine fehlerhafte Verarbeitung frühzeitig unterbunden, bevor eine Verbindung mit der OpenSearch Instanz aufgebaut ist.

Zur Messung der Verarbeitungszeit wird zu Beginn über `hrtime(true)` ein monotoner Zeitstempel erfasst. Dieser wird am Ende der Verarbeitung zur Berechnung der Gesamtlaufzeit verwendet.

Für die Verarbeitung der Logs nimmt `processLogs()` zunächst einen Zeitraum entgegen, ruft über den `OpenSearchService` alle Logeinträge ab und leitet jeden Eintrag einzeln durch die `FilterPipeline`. Dabei wird zusätzlich zwischen `total_fetched` und `total_processed` unterschieden, wodurch nachvollziehbar wird, dass alle abgerufenen Logeinträge auch tatsächlich verarbeitet werden. Die verarbeiteten Einträge werden dabei nicht sofort in das Ausgabe-Array geschrieben, sondern zunächst in einem Batch gesammelt. Sobald der Batch die konfigurierte Größe erreicht, wird er über die Methode `writeBatch()` geleert. Ein verbleibender unvollständiger Batch am Ende der Schleife wird ebenfalls übertragen.

Zur Absicherung der Stabilität ist der gesamte Verarbeitungsblock in einem `try/catch`-Block gekapselt. Auftretende Fehler werden im `statistics`-Array unter `errors` gespeichert und zusätzlich über `error_log()` protokolliert.

Abschließend gibt `processLogs()` ein assoziatives Array zurück, das unter dem Schlüssel `filtered` die gefilterten Logeinträge und unter `statistics` die erfassten Verarbeitungsinformationen enthält.

Zusätzlich zur Gesamtverarbeitungszeit wird die reine Filterzeit separat erfasst. Zu Beginn jeder Iteration wird über `hrtime(true)` ein Zeitstempel gesetzt, der nach Abschluss der Filterverarbeitung ausgelesen und zu einem Zähler `$totalFilterTime` addiert wird. Am Ende wird dieser Wert in die Statistik eingetragen. Diese Unterscheidung ermöglicht es, den Anteil der reinen Filterverarbeitungszeit von anderen Prozessen abzugrenzen.

LogFilterService processLogs() Methode

```
1 public function processLogs(DateTimeInterface $from,
    DateTimeInterface $to): array
```

```

2 {
3     $this->assertValidRange($from, $to);
4     $startTime = hrtime(true);
5     $totalFilterTime = 0;
6
7     $statistics =
8     [
9         'date_range' => ['from' => $from->format(\
10             DateTimeInterface::ATOM), 'to' => $to->format(\
11                 DateTimeInterface::ATOM)],
12         'total_fetched' => 0,
13         'total_processed' => 0,
14         'batches_processed' => 0,
15         'errors' => [],
16         'processing_time_ms' => 0.0,
17         'filtering_time_ms' => 0.0,
18     ];
19
20     $filteredLogs = [];
21     $batch = [];
22
23     try
24     {
25         foreach($this->openSearchService->getLogs($from, $to)
26             as $logEntry)
27         {
28             $filterTime = hrtime(true);
29             $statistics['total_fetched']++;
30             $filteredLogEntry = $this->filterPipeline->process(
31                 $logEntry);
32             $statistics['total_processed']++;
33
34             $batch[] = $filteredLogEntry;
35
36             if(count($batch) >= $this->batchSize)
37             {
38                 $this->writeBatch($batch, $filteredLogs,
39                     $statistics);
40                 $batch = [];
41             }
42             $totalFilterTime += (hrtime(true) - $filterTime);
43         }
44
45         if(!empty($batch))
46         {
47             $this->writeBatch($batch, $filteredLogs,
48                 $statistics);
49         }
50     }
51 }

```

```

45     }
46     catch(Exception $exception)
47     {
48         $statistics['errors'][] = $exception->getMessage();
49         error_log("Log processing error: " . $exception->
                    getMessage());
50     }
51
52     $statistics['processing_time_ms'] = round((hrtime(true) -
53         $startTime) / 1_000_000, 2);
54     $statistics['filtering_only_ms'] = round($totalFilterTime /
55         1_000_000, 2);
56
57     return
58     [
59         'statistics' => $statistics,
60         'filtered' => $filteredLogs
61     ];
62 }

```

Die private Methode `writeBatch()` nimmt den aktuellen Batch von Logeinträgen sowie Referenzen auf das Ausgabe-Array `$filteredLogs` und das Statistik-Array `$statistics` entgegen. Für jeden Logeintrag im Batch wird die Methode `toArray()` aufgerufen und das Ergebnis an `$filteredLogs` angehängt. Anschließend wird der Zähler `batches_processed` um eins erhöht.

Die Übergabe der Arrays als Referenzparameter `&$filteredLogs` und `&$statistics` stellt sicher, dass die innerhalb von `writeBatch()` vorgenommenen Änderungen auch in der aufrufenden Methode `processLogs()` sichtbar bleiben. In PHP werden Arrays standardmäßig `by value` übergeben, sodass Änderungen innerhalb einer Funktion ohne Referenz keine Auswirkungen auf die ursprüngliche Variable haben. Durch die Verwendung von Referenzen wird diese Einschränkung aufgehoben, wodurch Modifikationen an `$filteredLogs` und `$statistics` direkt auf die im Aufrufer existierenden Arrays angewendet werden [23].

Auf diese Weise gewährleistet `writeBatch()` sowohl die korrekte Aggregation der verarbeiteten Logeinträge als auch die konsistente Aktualisierung der Statistik über verarbeiteten Batches.

LogFilterService writeBatch() Methode

```

1 private function writeBatch(array $batch, array &$filteredLogs,
2     array &$statistics): void
3 {
4     foreach ($batch as $logEntry)
5     {
6         $filteredLogs[] = $logEntry->toArray();
7     }
8     $statistics['batches_processed']++;
9 }

```

Der vollständige Code dieser Klasse ist im Anhang A.5 einsehbar.

4.3 Logdateienabruf

Der Abruf der Logs aus OpenSearch erfolgt über die Klassen `OpenSearchAPIClient` und `OpenSearchService`. Die Aufteilung in zwei Klassen entspricht dem Single-Responsibility-Prinzip und trennt die Konfiguration und Bereitstellung des OpenSearch-Clients von der fachlichen Abruflogik.

OpenSearchApiClient

Der `OpenSearchApiClient` ist ein Wrapper um das offizielle OpenSearch PHP Software Development Kit (SDK). Im Konstruktor werden URL, Benutzername und Passwort der OpenSearch-Instanz entgegengenommen. Über den `OpenSearchClientBuilder` des SDKs wird daraus ein vollständig konfigurierter Client aufgebaut der Basis-Authentifizierung und die explizite Port-Angabe im Host-Header aktiviert.

Der Aufbau des Clients erfolgt innerhalb eines try-catch-Blocks, um mögliche Fehler während der Initialisierung abzufangen, beispielsweise durch ungültige Verbindungsparameter. Tritt eine Exception auf, wird diese in eine `RuntimeException` überführt und mit einer Fehlermeldung erneut ausgelöst. Dadurch werden Initialisierungsfehler kontrolliert behandelt und gleichzeitig für aufrufende Komponenten eindeutig signalisiert.

OpenSearchApiClient Konstruktor

```
1 public function __construct(string $openSearchUrl, string
   $openSearchUser, string $openSearchPassword)
2 {
3     try
4     {
5         $this->osClient = (new OpenSearchClientBuilder())
6             ->setHosts([$openSearchUrl])
7             ->setBasicAuthentication($openSearchUser,
8                 $openSearchPassword)
9             ->includePortInHostHeader(true)
10            ->build();
11    }
12    catch(\Exception $e)
13    {
14        throw new \RuntimeException('Failed to build OpenSearch
15        client: ' . $e->getMessage());
16    }
17 }
```

Die Klasse stellt zwei public Methoden bereit. Die Methode `search()` nimmt einen Indexnamen sowie Anfragedaten entgegen, führt die Suchanfrage gegen den angegebene Index aus und gibt die Antwort des SDKs als assoziatives Array zurück.

Sie bildet somit den primären Einstiegspunkt für alle Suchanfragen des `OpenSearchService`.

OpenSearchApiClient search() Methode

```
1 public function search(string $index, array $params): array
2 {
3     return $this->osClient->search(
4         [
5             'index' => $index,
6             'scroll' => $params['scroll'] ?? null,
7             'body'   => $params['body']
8         ]
9     );
10 }
```

Die Methode `getClient()` gibt den zugrundeliegenden SDK-Client zurück. Dieser wird im `OpenSearchService` für weitergehende Operationen benötigt, wie das Abfragen großer Datenmengen.

OpenSearchApiClient getClient() Methode

```
1 public function getClient(): OSClient
2 {
3     return $this->osClient;
4 }
```

Der vollständige Code dieser Klasse ist im Anhang A.6 einsehbar.

OpenSearchService

Der `OpenSearchService` enthält die fachliche Logik des Logabrufs. Im Konstruktor werden ein `OpenSearchAPIClient`, der Name des Source-Index sowie eine Batch-Größe entgegengenommen, über die festgelegt wird, wie viele Dokumente pro Scroll-Anfrage abgerufen werden.

OpenSearchService Konstruktor

```
1 public function __construct(string $url, string $user, string
2     $password, string $sourceIndex, int $batchSize)
3 {
4     $this->client = new OpenSearchApiClient($url, $user,
5         $password);
6     $this->sourceIndex = $sourceIndex;
7     $this->batchSize = $batchSize;
8 }
```

Die dazugehörigen Getter-Methoden `getSourceIndex()` und `getBatchSize()` ermöglichen den externen Lesezugriff auf diese Konfigurationswerte.

OpenSearchService Getter-Methoden

```
1 public function getSourceIndex(): string
```

```

2 {
3     return $this->sourceIndex;
4 }
5
6 public function getBatchSize(): int
7 {
8     return $this->batchSize;
9 }

```

Die private Methode `buildQuery()` erzeugt eine OpenSearch-Range-Query für den angegebenen Zeitraum. Die Range-Query ist ein Abfragetyp des OpenSearch Query DSL, der Dokumente zurückgibt, deren Feldwert innerhalb eines definierten Bereichs liegt. Die Parameter `gte` (Greater Than or Equal) und `lte` (Less Than or Equal) legen dabei die entsprechenden Grenzen fest. Mittels einem `bool/must`-Kontext stellt die Abfrage sicher, dass ausschließlich Dokumente zurückgegeben werden, die die Zeitraumbedingung erfüllen [9].

OpenSearchService Getter-Methoden

```

1 private function buildQuery(DateTimeInterface $from,
2     DateTimeInterface $to): array
3 {
4     return
5     [
6         'bool' =>
7         [
8             'must' =>
9             [
10                [
11                    'range' =>
12                    [
13                        '@timestamp' =>
14                        [
15                            'gte' => $from->format(
16                                DateTimeInterface::ATOM),
17                            'lte' => $to->format(
18                                DateTimeInterface::ATOM)
19                        ],
20                    ],
21                ],
22            ],
23        ],
24    ];
25 }

```

Die Hilfsmethode `createLogEntry()` übersetzt einen einzelnen OpenSearch-Treffer in ein `LogEntry`-Objekt. Sie liest dazu die Felder `_id` und `_source` aus dem Treffer aus und setzt diese als `id` und `data` des neuen Objekts. Fehlende Felder werden dabei mit leeren Standardwerten belegt.

OpenSearchService createLogEntry() Methode

```
1 private function createLogEntry(array $hit): LogEntry
2 {
3     return new LogEntry(id: $hit['_id'] ?? '', data: $hit['_source'] ?? [],);
4 }
```

Die zentrale Methode `getLogs()` dient dem Abruf von Logeinträgen für einen definierten Zeitraum. Sie gibt die Logeinträge nicht als komplettes Array zurück, sondern nutzt einen PHP-Generator, um die Werte schrittweise zu liefern. Ein PHP-Generator ist eine spezielle Funktion, die Iterationen speichereffizient ermöglicht, indem sie ihre Werte über das Schlüsselwort `yield` zurückgibt. Bei jedem Aufruf von `yield` wird die Ausführung der Funktion unterbrochen, der aktuelle Wert an den Aufrufer übergeben und die Position innerhalb der Funktion gespeichert. Bei der nächsten Iteration wird die Ausführung genau an dieser Stelle fortgesetzt [24].

Durch den Einsatz eines Generators liefert `getLogs()` die Logeinträge einzeln an das aufrufende `foreach`-Konstrukt, sobald sie verfügbar sind. Dies reduziert den Speicherbedarf, da nicht alle Logeinträge gleichzeitig im Arbeitsspeicher gehalten werden müssen, was insbesondere bei großen Datenmengen entscheidend ist.

Um auch Datenmengen zu verarbeiten, die die maximale Trefferzahl einer einzelnen Anfrage übersteigen, verwendet `getLogs()` die OpenSearch Scroll-API. Nach dem ersten Aufruf wird die zurückgegebene `scroll_id` genutzt, um so lange weitere Batches zu laden, bis keine Treffer mehr vorhanden sind. Abschließend wird der Scroll-Kontext über `clearScroll()` freigegeben, um Ressourcen nicht unnötig zu belegen.

OpenSearchService getLogs() Methode

```
1 public function getLogs(DateTimeInterface $from,
2     DateTimeInterface $to): \Generator
3 {
4     $response = $this->client->search($this->sourceIndex, [
5         'scroll' => '5m',
6         'body' =>
7         [
8             'size' => $this->batchSize,
9             'sort' => ['@timestamp' => 'asc'],
10            'query' => $this->buildQuery($from, $to),
11        ]
12    ]);
13
14    $scrollId = $response['_scroll_id'] ?? null;
15
16    foreach ($response['hits']['hits'] ?? [] as $hit)
17    {
18        yield $this->createLogEntry($hit);
19    }
```

```

20     $searchClient = $this->client->getClient();
21
22     while ($scrollId && !empty($response['hits']['hits']))
23     {
24         $response = $searchClient->scroll(
25             [
26                 'scroll_id' => $scrollId,
27                 'scroll' => '5m',
28             ]);
29
30         $scrollId = $response['_scroll_id'] ?? null;
31
32         foreach ($response['hits']['hits'] ?? [] as $hit)
33         {
34             yield $this->createLogEntry($hit);
35         }
36     }
37
38     if ($scrollId)
39     {
40         try
41         {
42             $searchClient->clearScroll(['scroll_id' =>
43                 $scrollId]);
44         }
45         catch (Exception $e)
46         {
47             error_log("Failed to clear scroll: " . $e->
48                 getMessage());
49         }
50     }
51 }

```

Der vollständige Code dieser Klasse ist im Anhang A.7 einsehbar.

4.4 Filterpipeline

Die Filterpipeline bildet das verbindende Element zwischen dem Logabruf und den einzelnen Filterklassen. Sie ist nach dem Pipes-and-Filters Architekturmuster aufgebaut und steuert die sequenzielle Verarbeitung jedes LogEntry-Objekts durch alle registrierten Filter.

FilterInterface

Das Interface `FilterInterface` legt den Vertrag fest, den alle konkreten Filter erfüllen müssen. Es schreibt drei Methoden vor: `process()` zur Verarbeitung und Rückgabe eines Logeintrags, `getName()` zur Identifikation des Filters sowie `isEnabled()` zur

Abfrage des aktuellen Aktivierungsstatus. Durch dieses Interface wird sichergestellt, dass die FilterPipeline jeden Filter einheitlich ansprechen kann, unabhängig von seiner konkreten Implementierung.

Filterinterface

```
1 interface FilterInterface
2 {
3     public function process(LogEntry $logEntry): LogEntry;
4
5     public function getName(): string;
6
7     public function isEnabled(): bool;
8 }
```

Der vollständige Code dieser Klasse ist im Anhang A.9 einsehbar.

AbstractFilter

Die abstrakte Klasse AbstractFilter implementiert FilterInterface und stellt den konkreten Filterklassen gemeinsam genutzte Funktionalität zur Verfügung.

Im Konstruktor von AbstractFilter kann optional ein Konfigurationsarray \$config übergeben werden. Dieses wird mit einer vordefinierten Standardkonfiguration zusammengeführt, bei der der Schlüssel enabled standardmäßig auf true gesetzt ist. Ein Filter ist damit automatisch aktiv, sofern keine abweichende Konfiguration angegeben wird.

AbstractFilter Konstruktor

```
1 public function __construct(array $config = [])
2 {
3     $this->config = array_merge(['enabled' => true], $config);
4 }
```

Soll ein Filter nicht ausgeführt werden, kann dies beim Erzeugen der Instanz explizit festgelegt werden, indem enabled auf false gesetzt wird. Der Filter bleibt in diesem Fall Bestandteil der Pipeline, wird jedoch bei der Verarbeitung nicht beachtet. Auf diese Weise lässt sich das Verhalten einzelner Filter gezielt steuern, ohne ihre Position in der Pipeline verändern oder sie vollständig entfernen zu müssen. Die Methode isEnabled() gibt den aktuellen Aktivierungsstatus der Filter zurück.

AbstractFilter isEnabled() Methode

```
1 public function isEnabled(): bool
2 {
3     return $this->config['enabled'] ?? true;
4 }
```

Während die Aktivierung bestimmt, ob ein Filter ausgeführt wird oder nicht, helfen zusätzliche Hilfsmethoden wie buildFieldNameRegex(), Daten gezielt zu verarbeiten.

Die Methode `buildFieldNameRegex()` erzeugt aus einer Liste von Feldnamen einen regulären Ausdruck, der sämtliche übergebenen Bezeichnungen als Alternativen enthält. Hierzu wird jeder Feldname mithilfe der Funktion `preg_quote` maskiert. Diese Funktion stellt sicher, dass der übergebene Feldname unverändert als Suchbegriff verwendet wird. Zeichen, die in regulären Ausdrücken eine spezielle Funktion haben, werden dabei automatisch so angepasst, dass sie nicht als Bestandteil der Regex-Syntax, sondern als normale Zeichen interpretiert werden [25]. Der resultierende Ausdruck repräsentiert somit eine kompakte Liste von Alternativen aller übergebenen Feldnamen. Dieser kann in verschiedenen Verarbeitungsschritten verwendet werden, um mehrere Feldbezeichnungen effizient innerhalb eines einzigen regulären Ausdrucks zu berücksichtigen.

AbstractFilter buildFieldNameRegex() Methode

```

1 protected function buildFieldNameRegex(array $fieldNames):
    string
2 {
3     $escapedFields = array_map(function($field) {return
        preg_quote($field, '/');}, $fieldNames);
4
5     return implode('|', $escapedFields);
6 }

```

Die Methode `redactData()` durchläuft das Datenarray eines Logeintrags rekursiv und verarbeitet jedes Feld anhand von drei definierten Ansätzen. Zunächst wird jeder Feldname mittels `strtolower()` in Kleinschreibung überführt, um eine case-insensitive Prüfung gegen die in `$fieldLabelMap` hinterlegten Feldnamen zu ermöglichen. Dadurch können Feldnamen unabhängig von abweichender Groß- und Kleinschreibung konsistent erkannt werden. Handelt es sich beim Wert eines Felds um ein verschachteltes Array, erfolgt zunächst ein rekursiver Aufruf der Methode, sodass auch tiefliegendere Strukturen vollständig verarbeitet werden. Liegt kein Array vor, wird geprüft, ob für den normalisierten Feldnamen ein Eintrag in `$fieldLabelMap` existiert. In diesem Fall wird der zugehörige Wert durch den definierten Platzhalter ersetzt. Handelt es sich stattdessen um eine einfache Zeichenkette, wird der übergebene `$redactor`-Callable angewendet, um sensible Informationen innerhalb unstrukturierter Inhalte zu erkennen und zu filtern.

AbstractFilter redactData() Methode

```

1 protected function redactData(array $data, array $fieldLabelMap
    , callable $redactor): array
2 {
3     foreach($data as $key => $value)
4     {
5         $lowerKey = strtolower((string) $key);
6
7         if(is_array($value))
8         {
9             $data[$key] = $this->redactData($value,
                $fieldLabelMap, $redactor);

```

```

10         }
11
12         elseif(isset($fieldLabelMap[$lowerKey]))
13         {
14             $data[$key] = $fieldLabelMap[$lowerKey];
15         }
16
17         elseif(is_string($value))
18         {
19             $data[$key] = $redactor($value);
20         }
21     }
22     return $data;
23 }

```

Der vollständige Code dieser Klasse ist im Anhang A.8 einsehbar.

FilterPipeline

Die Klasse `FilterPipeline` verwaltet eine geordnete Liste von `FilterInterface`-Implementierungen und leitet jeden `LogEntry` sequenziell durch alle aktiven Filter.

Filter werden über die `addFilter()` Methode registriert. Bei der Registrierung wird automatisch ein Statistikeintrag für den neuen Filter angelegt, der dessen Namen, Aktivierungsstatus und die Anzahl verarbeiteter Einträge enthält.

FilterPipeline addFilter() Methode

```

1 public function addFilter(FilterInterface $filter): self
2 {
3     $this->filters[] = $filter;
4     $this->statistics[$filter->getName()] =
5         [
6             'enabled' => $filter->isEnabled(),
7             'processed' => 0
8         ];
9     return $this;
10 }

```

Die Methode `process()` iteriert über alle registrierten Filter und wendet jeden aktiven Filter auf den übergebenen `LogEntry` an. Das Ergebnis jedes Filters dient dabei als Eingabe für den nächsten. Parallel dazu wird der Verarbeitungszähler des jeweiligen Filters erhöht.

Filterpipeline process() Methode

```

1 public function process(LogEntry $logEntry): LogEntry
2 {
3     foreach ($this->filters as $filter)

```

```

4      {
5          if ($filter->isEnabled())
6          {
7              $logEntry = $filter->process($logEntry);
8              $this->statistics[$filter->getName()][ 'processed
              ' ]++;
9          }
10     }
11     return $logEntry;
12 }

```

Über `getFilters()` und `getStatistics()` können die registrierten Filter sowie die gesammelten Verarbeitungsstatistiken abgefragt werden.

Filterpipeline Getter-Methoden

```

1 public function getFilters(): array
2 {
3     return $this->filters;
4 }
5
6 public function getStatistics(): array
7 {
8     return $this->statistics;
9 }

```

Der vollständige Code dieser Klasse ist im Anhang A.4 einsehbar.

4.5 Filterklassen

Die Filterlogik ist modular aufgebaut und in drei spezialisierte Filterklassen gegliedert: `CredentialsFilter`, `InfrastructureFilter` und `NetworkFilter`. Da jede dieser Klassen auf derselben Grundarchitektur basiert und eine einheitliche Implementierungsstruktur über die abstrakte Klasse `AbstractFilter` sowie das `FilterInterface` verwenden, unterscheiden sie sich primär lediglich hinsichtlich der jeweils verarbeiteten Datenkategorien und der spezifischen regulären Ausdrücke. Die grundlegenden Mechanismen der feldbasierten sowie musterbasierten Filterung sowie die Integration in die Filterpipeline sind bei allen drei Filterklassen identisch aufgebaut. Aus diesem Grund wird im Folgenden exemplarisch nur der `CredentialsFilter` detailliert dargestellt. Der vollständige Code aller drei Filterklassen ist im Anhang A dieser Arbeit einsehbar.

FieldsToFilterConfig

Die Klasse `FieldsToFilterConfig` bildet die zentrale Grundlage aller Filterklassen. Sie kapselt sämtliche Feldnamen, die mit sensiblen Informationen in Verbindung stehen können und stellt diese über statische Methoden zur Verfügung. Jede dieser

Methoden gibt ein Array von Strings zurück, das unterschiedliche Schreibweisen eines Feldnamens berücksichtigt.

Die Verwendung statischer Methoden ermöglicht den Zugriff auf die Felddefinitionen, ohne dass ein Objekt der Klasse instanziiert werden muss, was den Speicherverbrauch reduziert und die Nutzung in verschiedenen Filterklassen vereinfacht. Durch die zentrale Konfigurationsklasse ist die Verwaltung der Felder vollständig von der Filterlogik getrennt. Das Hinzufügen oder Entfernen eines Feldnamens erfordert eine Änderung an einer einzigen Stelle, ohne dass eine der Filterklassen angepasst werden muss.

Die Feldnamen sind in drei thematische Kategorien gegliedert: Zugangsdaten, Infrastrukturinformationen und Netzwerkinformationen. Jede Kategorie umfasst dabei mehrere Feldgruppen, die über eigene Methoden bereitgestellt werden. Dabei berücksichtigt jede Gruppe unterschiedliche Schreibweisen desselben Feldnamens. So enthält z. B. die Methode `getPasswordFields()` verschiedene Schreibweisen von Feldnamen, die mit Passwörtern im Zusammenhang stehen.

FieldsToFilterConfig getPasswordFields() Methode

```
1 public static function getPasswordFields(): array
2 {
3     return
4     [
5         'password',
6         'pwd',
7         'passwd',
8         'userPassword',
9         'smtPassword',
10        'smtp_password',
11        'mailPassword',
12        'mail_Password',
13        'passwordSalt',
14        'password_salt',
15        'salts',
16    ];
17 }
```

Die drei Methoden `getSensitiveCredentialFields()`, `getSensitiveInfrastructureFields()` und `getSensitiveNetworkFields()` fassen die jeweiligen Feldgruppen einer Kategorie zusammen. Dies erfolgt über die PHP-Funktion `array_merge`, welche mehrere Arrays zu einem einzigen Array zusammenführt [26]. Da dieselben Feldnamen in mehreren Gruppen vorkommen können, wird das Ergebnis anschließend mit der Funktion `array_unique()` bereinigt, sodass jeder Feldname im zusammengeführten Array nur einmal vorkommt [27].

FieldsToFilterConfig getSensitiveCredentialFields() Methode

```
1 public static function getSensitiveCredentialFields(): array
2 {
3     return array_unique(array_merge
```

```

4      (
5          self::getPasswordFields(),
6          self::getTokenFields(),
7          self::getApiKeyFields(),
8          self::getSessionFields(),
9          self::getSecretFields(),
10         self::getAuthorizationFields(),
11     ));
12 }

```

Die dargestellten Codeausschnitte zeigen hierbei exemplarisch nur einen Teil der Implementierung der Klasse `FieldsToFilterConfig`. Die Klasse umfasst darüber hinaus weitere Methoden zur Definition zusätzlicher Feldgruppen für Token-, API-Key-, Session-, Secret-, Infrastruktur- und Netzwerkinformationen. Diese folgen demselben strukturellen Muster wie die gezeigten Methoden und unterscheiden sich ausschließlich hinsichtlich der konkret hinterlegten Feldnamen. Auch die Methoden für Infrastruktur- und Netzwerkategorien sind dabei analog aufgebaut.

Der vollständige Code dieser Klasse ist im Anhang A.1 einsehbar.

CredentialsFilter

Der `CredentialsFilter` ist für die Filterung zugangs- und authentifizierungsbezogener Informationen verantwortlich. Dazu zählen z. B. Passwörter, Tokens und API-Schlüssel. Ziel des Filters ist es, alle Informationen zu filtern, die einen unmittelbaren Zugriff auf Systeme oder Dienste ermöglichen könnten.

Im Konstruktor der Klasse wird zunächst eine interne Zuordnungstabelle `$fieldLabelMap` aufgebaut, welche jedem sensiblen Feldnamen einen fest definierten Platzhalter zuweist. Grundlage hierfür sind die in der Konfigurationsklasse `FieldsToFilterConfig` definierten Feldgruppen. Zusätzlich werden alle regulären Ausdrücke einmal vorkompiliert, um sicherzustellen, dass diese nicht bei jedem Logeintrag neu erzeugt werden müssen.

CredentialsFilter Konstruktor

```

1 public function __construct(array $config = [])
2 {
3     parent::__construct($config);
4     $this->buildFieldLabelMap();
5     $this->buildCompiledPatterns();
6 }

```

Die Methode `buildFieldLabelMap()` baut die interne Zuordnungstabelle `$fieldLabelMap` auf, die jedem sensiblen Feldnamen einen fest definierten Platzhalter zuweist. Dazu werden zunächst alle Feldkategorien zusammen mit ihrem jeweiligen Typ in einem assoziativen Array `$fieldGroups` gebündelt. Für jeden Typ wird anschließend über `getRedactionLabel()` der zugehörige Platzhalter ermittelt, wie z. B. `[PASSWORD REDACTED]` für die Feldkategorie `PASSWORD`. Innerhalb der

inneren Schleife wird jeder Feldname der Feldkategorie auf Kleinschreibung normalisiert und gemeinsam mit dem Platzhalter als Eintrag in `$fieldLabelMap` gespeichert. Die Normalisierung auf Kleinschreibung ist dabei notwendig, da der spätere Vergleich in der Methode `redactData()` ebenfalls auf Kleinschreibung basiert und Logs in der Praxis unterschiedliche Schreibweisen verwenden können. Das Ergebnis ist hierbei eine Zuordnungstabelle, die einen direkten Vergleich ohne erneuten Schleifendurchlauf über alle Kategorien ermöglicht.

CredentialsFilter buildFieldLabelMap() Methode

```
1 private function buildFieldLabelMap(): void
2 {
3     $fieldGroups = [
4         'PASSWORD' => FieldsToFilterConfig::getPasswordFields(),
5         'TOKEN' => FieldsToFilterConfig::getTokenFields(),
6         'APIKEY' => FieldsToFilterConfig::getApiKeyFields(),
7         'SESSION' => FieldsToFilterConfig::getSessionFields(),
8         'SECRET' => FieldsToFilterConfig::getSecretFields(),
9         'AUTHORIZATION' => FieldsToFilterConfig::
10         getAuthorizationFields(),
11     ];
12     foreach ($fieldGroups as $type => $fields)
13     {
14         $label = $this->getRedactionLabel($type);
15         foreach ($fields as $field)
16         {
17             $this->fieldLabelMap[strtolower($field)] = $label;
18         }
19     }
20 }
```

Die Methode `buildCompiledPatterns()` koordiniert die Vorkompilierung aller regulären Ausdrücke, indem sie für jede Kategorie eine eigene Kompiliermethode aufruft. Die Vorkompilierung zum Zeitpunkt der Instanziierung stellt sicher, dass die regulären Ausdrücke nicht bei jedem Logeintrag neu erzeugt werden müssen, was die Verarbeitungsgeschwindigkeit der Logeinträge erheblich verbessert.

CredentialsFilter buildCompiledPattern() Methode

```
1 private function buildCompiledPatterns(): void
2 {
3     $this->compilePasswordPatterns();
4     $this->compileTokenPatterns();
5     $this->compileApiKeyPatterns();
6     $this->compileSessionPatterns();
7     $this->compileSecretPatterns();
8     $this->compileAuthorizationPatterns();
9 }
```

Jede Kompilierungsmethode erzeugt für ihre Feldkategorie reguläre Ausdrücke und speichert diese zusammen mit dem zugehörigen Platzhalter in den parallelen Arrays `$compiledPatterns` und `$compiledLabels`. Durch die parallele Struktur wird sichergestellt, dass `preg_replace()` jeden Treffer durch den Platzhalter, der der jeweiligen Kategorie zugeordnet ist ersetzt.

CredentialsFilter compileSessionPatterns() Methode

```

1 private function compileSessionPatterns(): void
2 {
3     $field = $this->buildFieldNameRegex(FieldsToFilterConfig::
4         getSessionFields());
5     $label = $this->getRedactionLabel('SESSION');
6
7     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s*[:]=\s
8         *"[a-z0-9-]+"\/i';
9     $this->compiledLabels[] = $label;
10
11     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s*[:]=\s
12         *[a-z0-9-]+\/i';
13     $this->compiledLabels[] = $label;
14 }

```

Die Verarbeitung eines Logeintrags erfolgt über die Methode `process()`, welche zunächst über die Methode `isEnabled()` prüft, ob der Filter aktiviert ist. Ist dies der Fall, werden die Daten des Logeintrags über `getData()` ausgelesen und zusammen mit der Zuordnungstabelle `$fieldLabelMap` sowie dem Filter `redactCredentialPatterns()` an `redactData()` übergeben. Das gefilterte Datenarray wird anschließend über `setData()` zurück in den Logeintrag geschrieben und der veränderte Logeintrag zurückgegeben. Die rekursive Vorgehensweise von `redactData()` stellt dabei sicher, dass auch verschachtelte Datenstrukturen vollständig analysiert werden.

CredentialsFilter process() Methode

```

1 public function process(LogEntry $logEntry): LogEntry
2 {
3     if (!$this->isEnabled())
4     {
5         return $logEntry;
6     }
7
8     $redacted = $this->redactData
9     (
10         $logEntry->getData(),
11         $this->fieldLabelMap,
12         fn(string $value) => $this->redactCredentialPatterns(
13             $value)
14     );
15     $logEntry->setData($redacted);

```

```

16     return $logEntry;
17 }

```

Der Filter `redactCredentialPatterns()` wendet die vorkompilierten Muster auf eine einzelne Zeichenkette an.

Zu Beginn der Methode wird geprüft, ob die Zeichenkette mindestens eines der Zeichen "=" oder ":" enthält. Dies ist erforderlich, da die implementierten und vorkompilierten regulären Ausdrücke auf die Erkennung von Schlüssel-Wert-Zuweisungen ausgelegt sind. Ist keines der Zeichen in der Zeichenkette enthalten, wird der Wert unverändert zurückgegeben. Auf diese Weise wird ein unnötiger Aufruf von `preg_replace` vermieden, wodurch die Effizienz der Methode verbessert wird.

Enthält die zu untersuchende Zeichenkette keines dieser beiden Zeichen, wird davon ausgegangen, dass keine entsprechende Zuweisung vorliegt, sodass ein erfolgreicher Musterabgleich ausgeschlossen werden kann. In einem solchen Fall wird die Eingabe unverändert zurückgegeben, ohne weitere Verarbeitungsschritte durchzuführen.

CredentialsFilter redactCredentialPatterns() Methode

```

1 private function redactCredentialPatterns(string $value):
    string
2 {
3     if (!str_contains($value, '=') && !str_contains($value, ':'))
4     {
5         return $value;
6     }
7
8     return preg_replace($this->compiledPatterns, $this->
        compiledLabels, $value) ?? $value;
9 }

```

Der vollständige Code dieser Klasse ist im Anhang A.10 einsehbar.

NetworkFilter

Der `NetworkFilter` ist für die Filterung netzwerkbezogener Informationen wie IPv4-/IPv6-Adressen, MAC-Adressen und Hostnamen verantwortlich. Diese Daten sind insbesondere in sicherheitskritischen Umgebungen sensibel, da sie Rückschlüsse auf interne Netzstrukturen ermöglichen können.

Beim Erstellen des Filters werden die in der Konfigurationsklasse definierten netzwerkbezogenen Feldnamen übernommen. Diese werden intern mit passenden Platzhaltern verknüpft, sodass erkannte Werte entsprechend gefiltert werden können. Die relevanten Feldnamen sind dabei zentral in der Konfigurationsklasse festgelegt.

Zusätzlich zur feldbasierten Erkennung enthält der `NetworkFilter` spezialisierte Methoden zur Identifikation typischer Netzwerkformate. IPv4-Adressen werden anhand

ihres Dezimalformats und der Trennung durch Punkte erkannt, während IPv6-Adressen durch ihr Hexadezimalformat und Trennung durch Doppelpunkte identifiziert werden. MAC-Adressen werden sowohl im Doppelpunkt- als auch im Bindestrichformat berücksichtigt und Hostnamen werden anhand üblicher Domainstrukturen erfasst.

Um die Verarbeitung großer Mengen an Logs effizient zu gestalten, wird vor der Anwendung bestimmter regulärer Ausdrücke geprüft, ob charakteristische Zeichen wie Punkte oder Doppelpunkte im jeweiligen String vorkommen. Diese Vorprüfung reduziert unnötige Regex-Ausführungen und trägt somit zur Performanceoptimierung bei.

NetworkFilter redactNetworkPatterns() Methode

```
1 private function redactNetworkPatterns(string $value): string
2 {
3     if (!str_contains($value, '.') && !str_contains($value,
4         ':') && !str_contains($value, '-'))
5     {
6         return $value;
7     }
8     return preg_replace($this->compiledPatterns, $this->
9         compiledLabels, $value) ?? $value;
10 }
```

Der vollständige Code dieser Klasse ist im Anhang A.12 einsehbar.

InfrastructureFilter

Der InfrastructureFilter ist für die Filterung infrastrukturbezogener Informationen zuständig. Dazu zählen insbesondere datenbankspezifische Informationen wie Typen, Namen, Benutzernamen oder Passwörter von Datenbanken. Diese Informationen sind sicherheitsrelevant, da sie Rückschlüsse auf die interne Systemarchitektur, eingesetzte Technologien und mögliche Angriffspunkte zulassen.

Der vollständige Code dieser Klasse ist im Anhang A.11 einsehbar.

4.6 Integration als Symfony Bundle

Neben der Bereitstellung als allgemeines Composer-Package wurde das Logfiltersystem zusätzlich als Symfony Bundle integriert, um eine nahtlose Einbindung in Symfony-basierte Anwendungen zu ermöglichen. Ein Symfony Bundle bezeichnet eine wiederverwendbare Einheit von Code zur Bereitstellung spezifischer Funktionalitäten innerhalb des Symfony-Frameworks. Bundles dienen sowohl der Umsetzung von Kernfunktionen des Frameworks als auch der Integration zusätzlicher Erweiterungen. Sie ermöglichen die gemeinsame Nutzung von Code und Funktionalitäten über mehrere Anwendungen hinweg und müssen innerhalb einer Symfony-Anwendung registriert werden, um verwendet zu werden [28].

Die Bundle-Integration verfolgt das Ziel, die verschiedenen Komponenten nicht manuell in jeder Anwendung instanziiieren und konfigurieren zu müssen, sondern deren Registrierung und Konfiguration in die Mechanismen des Frameworks zu überführen. Dadurch wird insbesondere die Wiederverwendbarkeit erhöht und die Einbindung des Logfiltersystems in bestehende Symfony-Anwendungen vereinfacht.

LogFilterBundle

Für die Bundle-Integration des Logfiltersystems wird die Klasse `LogFilterBundle` verwendet, die auf der Symfony-Basisklasse `AbstractBundle` basiert.

`LogFilterBundle` übernimmt zwei zentrale Aufgaben. Zum einen definiert sie über die Methode `configure()` die vom Bundle bereitgestellten Konfigurationsparameter. Zum anderen registriert sie über die Methode `loadExtension()` die zentrale Komponente des Logfiltersystems im Symfony Dependency-Injection-Container [29] und verknüpft deren Abhängigkeiten.

Die Methode `configure()` definiert die extern konfigurierbaren Parameter des Bundles. Dazu gehören insbesondere die Verbindungsparameter zur `OpenSearch`-Instanz, der Name des zu verwendenden Index, die Batch-Größe sowie das Scroll-Intervall.

LogfilterBundle configure() Methode

```
1 public function configure(DefinitionConfigurator $definition):  
    void  
2 {  
3     $definition->rootNode()  
4         ->children()  
5         ->scalarNode('opensearch_url')->isRequired()->end()  
6         ->scalarNode('opensearch_user')->isRequired()->end()  
7         ->scalarNode('opensearch_password')->isRequired()->  
            end()  
8         ->scalarNode('opensearch_index')->isRequired()->end()  
9         ->integerNode('batch_size')->defaultValue(2500)->  
            end()  
10        ->scalarNode('scroll_window')->defaultValue('5m')->  
            end()  
11        ->end();  
12 }
```

Die Registrierung der Komponenten erfolgt über die Methode `loadExtension()`. Diese überführt die zuvor implementierten Kernkomponenten in den Symfony Dependency-Injection-Container und definiert zugleich deren Abhängigkeiten.

LogfilterBundle loadExtension() Methode

```
1 public function loadExtension(array $config,  
    ContainerConfigurator $container, ContainerBuilder $builder)  
    : void
```

```

2 {
3     $container->services()
4         ->set(OpenSearchApiClient::class)
5         ->args(
6             [
7                 $config['opensearch_url'],
8                 $config['opensearch_user'],
9                 $config['opensearch_password'],
10            ])
11         ->set(OpenSearchService::class)
12         ->args([service(OpenSearchApiClient::class),
13                 $config['opensearch_index'], $config['batch_size'],
14                 $config['scroll_window']])
15         ->set(FilterPipeline::class)
16         ->set(LogFilterService::class)
17         ->args([service(OpenSearchService::class), service(
18             FilterPipeline::class), $config['batch_size']])
19         ->public();
20 }

```

Innerhalb dieser Methode erhält der `OpenSearchApiClient` zunächst die erforderlichen Verbindungsparameter. Darauf aufbauend wird der `OpenSearchService` mit dem Client sowie den abrufspezifischen Parametern erzeugt. Der `LogFilterService` erhält anschließend sowohl den `OpenSearchService` als auch die `FilterPipeline` als Abhängigkeit und steht nach der Registrierung innerhalb des Containers als zentraler Einstiegspunkt des Logfiltersystems zur Verfügung.

Die Bundle-Integration ergänzt damit die generische Bereitstellung als Composer-Package um eine framework-spezifische Integrationsschicht. Während die Filterlogik weiterhin unabhängig vom Framework implementiert bleibt, übernimmt das Bundle die standardisierte Konfiguration, Service-Registrierung und Bereitstellung innerhalb der Symfony-Infrastruktur.

Der vollständige Code dieser Klasse ist im Anhang A.3 einsehbar.

4.7 Tests

Zur Sicherstellung der korrekten Funktionsweise des entwickelten Logfiltersystems wurden verschiedene Tests mit PHPUnit durchgeführt. Der Fokus dieser Tests liegt auf der Verifikation der Filterlogik sowie ihres Zusammenspiels innerhalb der Filterpipeline, da die Filterklassen keine aktive OpenSearch-Verbindung voraussetzen. Die Validierung des Gesamtablaufs unter realistischen Bedingungen, einschließlich des Logabrufs über die Scroll-API, erfolgte ergänzend über ein dediziertes Testskript, das direkt gegen eine angebundene OpenSearch-Instanz ausgeführt wurde.

Unit-Tests

Die Unit-Tests sind in fünf Testklassen unterteilt: `LogEntryTest`, `CredentialsFilterTest`, `NetworkFilterTest`, `InfrastructureFilterTest` und `FilterPipelineTest`. Die unterschiedlichen Filterklassen werden jeweils isoliert innerhalb eigenständiger Testklassen geprüft, um eine klare Trennung der Inhalte, Verantwortlichkeiten sowie eine gezielte Validierung der jeweiligen Funktionalitäten zu ermöglichen.

Die in diesem Abschnitt dargestellten Testmethoden stellen exemplarische Ausschnitte der vollständigen Testimplementierung dar. Zur besseren Lesbarkeit werden nicht alle Testfälle im Detail aufgeführt. Der vollständige Code aller Tests ist in Anhang A dieser Arbeit enthalten und ermöglicht dort eine vollständige Nachvollziehbarkeit aller Testfälle.

Die Testklasse `LogEntryTest` konzentriert sich auf die Überprüfung der grundlegenden Funktionalitäten der Model-Klasse `LogEntry`. Im Rahmen dieser Tests werden der Konstruktor sowie die Getter- und Setter-Methoden auf ihre korrekte Implementierung hin validiert. Zusätzlich wird die Funktionsweise der Methode `toArray()` geprüft, da diese eine wesentliche Rolle in der strukturierten Ausgabe spielt und somit für die ordnungsgemäße Serialisierung innerhalb des Models verantwortlich ist.

`LogEntryTest testToArray()` Test-Methode

```
1 public function testToArray(): void
2 {
3     $id = 'test-123';
4     $data = ['message' => 'Test'];
5     $logEntry = new LogEntry($id, $data);
6
7     $array = $logEntry->toArray();
8
9     $this->assertIsArray($array);
10    $this->assertEquals($id, $array['id']);
11    $this->assertEquals($data, $array['data']);
12 }
```

Der vollständige Code dieser Klasse ist im Anhang A.13 einsehbar.

Die Testklasse `FilterPipelineTest` überprüft das Zusammenspiel aller Filterklassen. Hierbei werden Logeinträge mit Inhalten aus verschiedenen Filterkategorien durch die Pipeline verarbeitet, um sicherzustellen, dass die sequenzielle Verarbeitung korrekt durch alle registrierten Filter läuft und dabei jede Kategorie sensibler Daten zuverlässig gefiltert wird, ohne dass es zu Inkonsistenzen zwischen den Filtern kommt.

`FilterPipelineTest testLogWithCredentialsInPipeline()` Test-Methode

```
1 public function testLogWithCredentialsInPipeline(): void
2 {
3     echo str_repeat("=", 40) . PHP_EOL;
4     echo "\nRESULTS FOR: testLogWithCredentialsInPipeline\n";
```

```

5      $originalData =
6          [
7              'timestamp' => '2026-02-19T06:00:00Z',
8              'action' => 'user_login',
9              'user' => 'boris',
10             'password' => 'password123',
11             'ip' => '192.168.1.100',
12             'session' => 'sess_abc123def456'
13         ];
14
15     echo "\nORIGINAL LOG:\n";
16     echo json_encode($originalData, JSON_PRETTY_PRINT) . "\n";
17
18     $logEntry = new LogEntry('1', $originalData);
19
20     echo "\nPROCESSING THROUGH PIPELINE\n";
21     $filtered = $this->filterPipeline->process($logEntry);
22
23     echo "\nFILTERED LOG:\n";
24     echo json_encode($filtered->getData(), JSON_PRETTY_PRINT) .
25         "\n";
26
27     $this->printFilterStats();
28
29     $data = $filtered->getData();
30     $this->assertEquals('boris', $data['user']);
31     $this->assertEquals('[PASSWORD REDACTED]', $data['password
32         ']);
33     $this->assertEquals('[IP ADDRESS REDACTED]', $data['ip']);
34     $this->assertEquals('[SESSION INFORMATION REDACTED]', $data
35         ['session']);
36 }

```

Der vollständige Code dieser Klasse ist im Anhang A.14 einsehbar.

Die Teststruktur für `CredentialsFilterTest`, `NetworkFilterTest` und `InfrastructureFilterTest` ist konsistent aufgebaut, da diese eine einheitliche Architektur über das `FilterInterface` sowie die abstrakte Basisklasse `AbstractFilter` verwenden. Dadurch können Tests für unterschiedliche Filtertypen nach demselben Prinzip implementiert werden, insbesondere hinsichtlich der feldbasierten und musterbasierten Filterung. Die dargestellten Testmethoden sind daher exemplarisch zu verstehen, da sie stellvertretend für eine größere Anzahl ähnlicher Testfälle stehen. Die weiteren Tests folgen demselben Aufbauprinzip und unterscheiden sich primär in den jeweiligen Eingabedaten und erwarteten Filterungen. Der vollständige Code des `CredentialsFilterTest` ist in Anhang A.15, des `NetworkFilterTest` in Anhang A.17 und des `InfrastructureFilterTest` in Anhang A.16 einsehbar.

`CredentialsFilterTest testPreservesNonSensitiveData()` Test-Methode

```

1 public function testPreservesNonSensitiveData(): void
2 {
3     $logEntry = new LogEntry('1',
4         [
5             'username' => 'boris_boris',
6             'email' => 'boris@example.com',
7             'timestamp' => '2026-01-01T12:00:00Z',
8             'password' => 'password123'
9         ]
10    );
11
12    $filtered = $this->filter->process($logEntry);
13    $data = $filtered->getData();
14
15    $this->assertEquals('boris_boris', $data['username']);
16    $this->assertEquals('boris@example.com', $data['email']);
17    $this->assertEquals('2026-01-01T12:00:00Z', $data['timestamp']);
18    $this->assertEquals('[PASSWORD REDACTED]', $data['password']);
19 }

```

Zusätzlich wird geprüft, ob die feldbasierte Filterung auch in verschachtelten Datenstrukturen korrekt funktioniert. Hierzu werden mehrdimensionale Arrays verwendet, in denen sensible Informationen unabhängig von ihrer Position innerhalb der Struktur identifiziert und gefiltert werden müssen.

CredentialsFilterTest testRedactsNestedPasswordFields() Test-Methode

```

1 public function testRedactsNestedPasswordFields(): void
2 {
3     $logEntry = new LogEntry('1',
4         [
5             'user' =>
6                 [
7                     'username' => 'boris',
8                     'password' => 'password123'
9                 ]
10        ]
11    );
12
13    $filtered = $this->filter->process($logEntry);
14    $data = $filtered->getData();
15
16    $this->assertEquals('boris', $data['user']['username']);
17    $this->assertEquals('[PASSWORD REDACTED]', $data['user']['password']);
18 }

```

Ergänzend zur feldbasierten Filterung wird die musterbasierte Filterung sensibler Informationen innerhalb von Zeichenketten getestet. Hierbei soll sichergestellt werden, dass sensible Daten wie z. B. ein Passwort, das als Schlüssel-Wert-Zuordnung in einer

längeren Zeichenkette eingebettet ist, korrekt gefiltert wird, während der umgebene Teil der Zeichenkette unverändert bleibt. Auch diese Tests werden für alle implementierten Filterklassen durchgeführt.

CredentialsFilterTest testRedactsPasswordPatternsInStrings() Test-Methode

```
1 public function testRedactsPasswordPatternsInStrings(): void
2 {
3     $logEntry = new LogEntry('1',
4         [
5             'message' => 'User login with password=password123
6                 failed'
7         ]
8     );
9
10    $filtered = $this->filter->process($logEntry);
11    $data = $filtered->getData();
12
13    $this->assertStringContainsString('[PASSWORD REDACTED]',
14        $data['message']);
15    $this->assertStringNotContainsString('password123', $data['
16        message']);
17 }
```

Sämtliche zusätzlich implementierten, im Anhang A dokumentierten Testfälle wurden erfolgreich mit PHPUnit ausgeführt und bestätigen die korrekte Funktion der jeweiligen Filterklassen analog zu den hier dargestellten exemplarischen Tests. Die erfolgreichen Testausführungen sind in Abbildung 2 dargestellt.

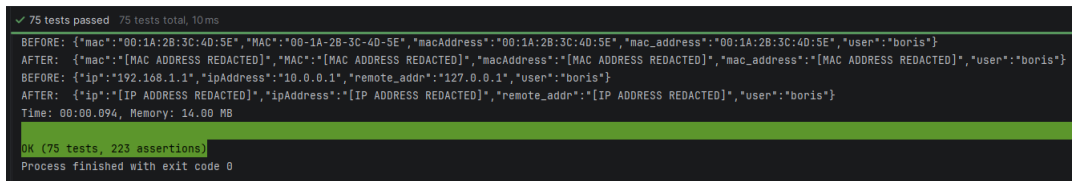


Abbildung 2: Ausführung sämtlicher PHPUnit-Tests des Logfiltersystems

Integrationstest mit OpenSearch

Für die Validierung der Filterpipeline unter realen Bedingungen wurde ein Skript entwickelt, das eine Verbindung zu einer unternehmensintern bereitgestellten OpenSearch-Testinstanz herstellt. Diese Testinstanz dient als dedizierte Testumgebung und liefert die notwendigen Logeinträge, die für die Verarbeitung durch die Filterpipeline abgerufen werden. Ziel dieses Ansatzes ist es, die Funktionsfähigkeit der Pipeline im Zusammenspiel aller Filter unter realistischen Bedingungen mit entsprechenden Abhängigkeiten zu überprüfen und sicherzustellen, dass sensible Informationen zuverlässig gefiltert werden.

Zunächst werden die erforderlichen Klassen in das Skript eingebunden und die Konfigurationsparameter für die Verbindung zur OpenSearch-Testinstanz geladen.

Diese beinhalten die URL der Instanz, den Nutzernamen, das Passwort sowie den Index, aus dem die Logeinträge abgerufen werden.

Test-Skript Konfigurationsparamter

```
1 require_once __DIR__ . '/vendor/autoload.php';
2
3 use Logfilter\Service\OpenSearchService;
4 use Logfilter\Service\FilterPipeline;
5 use Logfilter\Service\LogFilterService;
6 use Logfilter\Service\Filter\CredentialsFilter;
7 use Logfilter\Service\Filter\NetworkFilter;
8 use Logfilter\Service\Filter\InfrastructureFilter;
9
10 $config = require __DIR__ . '/config.local.php';
11 $url = $config['opensearch_url'];
12 $user = $config['opensearch_user'];
13 $password = $config['opensearch_password'];
14 $index = $config['opensearch_index'];
```

Diese Parameter stellen sicher, dass das Skript eine authentifizierte Verbindung zur bereitgestellten Datenbank herstellen kann. Durch die Verwendung einer separaten Konfigurationsdatei wird außerdem die Trennung von Code und sensiblen Zugangsdaten eingehalten.

Für den Logabruf wird ein Zeitintervall definiert. Im Beispiel umfasst diese die letzten zwei Minuten.

Test-Skript Zeitgrenzen

```
1 $from = new DateTime('-2 minutes');
2 $to = new DateTime('now');
```

Die Definition eines konkreten Zeitraums erlaubt eine gezielte Auswahl von Logeinträgen und reduziert die Menge der zu verarbeitenden Logs auf einen kontrollieren Bereich.

Die Instanz des `OpenSearchService` übernimmt die Kommunikation mit OpenSearch. Auf Basis dieser Instanz wird die Filterpipeline erstellt und mit den drei Filterklassen `CredentialsFilter`, `InfrastructureFilter` und `NetworkFilter` konfiguriert. Durch die Konfiguration der Filterpipeline wird sichergestellt, dass jeder Logeintrag sequenziell durch alle Filter verarbeitet wird.

Test-Skript Instanziierung

```
1 $openSearchService = new OpenSearchService(
2     url: $url,
3     user: $user,
4     password: $password,
5     sourceIndex: $index,
6     batchSize: 2500
7 );
8
```

```

9  $pipeline = new FilterPipeline();
10 $pipeline->addFilter(new CredentialsFilter());
11 $pipeline->addFilter(new NetworkFilter());
12 $pipeline->addFilter(new InfrastructureFilter());
13
14 $logFilterService = new LogFilterService(
15     openSearchService: $openSearchService,
16     filterPipeline: $pipeline,
17     batchSize: 2500
18 );

```

Die Methode `processLogs()` führt den Abruf der Logeinträge durch und filtert diese durch die Filterpipeline. Das daraus entstehenden Ergebnis enthält sowohl die gefilterten Logeinträge als auch statistische Informationen über den Filterprozess.

Test-Skript `processLogs()`

```

1  $result = $logFilterService->processLogs($from, $to);

```

Um die gefilterten Inhalte innerhalb der Logeinträge erkenntlich zu machen, werden Platzhalter definiert, die den sensiblen Informationen entsprechen. Diese Platzhalter dienen dazu, trotz der gefilterten Inhalte den Kontext innerhalb der Logeinträge nachvollziehbar zu erhalten.

Test-Skript Platzhalter

```

1  $redactionLabels =
2  [
3      'PASSWORD' => '[PASSWORD REDACTED]',
4      'TOKEN' => '[TOKEN REDACTED]',
5      'API KEY' => '[API KEY REDACTED]',
6      'SESSION' => '[SESSION INFORMATION REDACTED]',
7      'SECRET' => '[SECRET REDACTED]',
8      'AUTHORIZATION' => '[AUTHORIZATION REDACTED]',
9      'IP ADDRESS' => '[IP ADDRESS REDACTED]',
10     'MAC ADDRESS' => '[MAC ADDRESS REDACTED]',
11     'HOSTNAME' => '[HOSTNAME REDACTED]',
12     'DATABASE INFORMATION' => '[DATABASE INFORMATION REDACTED
13     ]',

```

Die anschließende Funktion `scanForRedactions()` durchläuft rekursiv alle Logeinträge, prüft jede Zeichenkette auf die definierten Platzhalter und zählt die Anzahl der auftretenden Platzhalter, um nachvollziehen zu können, wie häufig welche Inhalte gefiltert werden.

Test-Skript `scanForRedactions()` Methode

```

1  function scanForRedactions(array $data, string $path, array
2      $redactionLabels, array &$amp;counters, array &$amp;hits): void
3  {
4      foreach ($data as $key => $value)

```

```

4      {
5          $currentPath = $path ? "$path.$key" : $key;
6
7          if (is_array($value))
8          {
9              scanForRedactions($value, $currentPath,
10                             $redactionLabels, $counters, $hits);
11          }
12          elseif (is_string($value))
13          {
14              foreach ($redactionLabels as $type => $label)
15              {
16                  $count = substr_count($value, $label);
17                  if ($count > 0)
18                  {
19                      $counters[$type] += $count;
20                      $hits[] = "'$currentPath' => $value";
21                  }
22              }
23          }
24      }

```

Abschließend werden statistische Daten zur Filterung ausgegeben. Diese umfassen den untersuchten Zeitrahmen, die Gesamtanzahl der abgerufenen und verarbeiteten Logeinträge, die Anzahl der Batches, die Dauer der Filterung in Millisekunden und Sekunden als auch die isolierte Dauer der reinen Filterung.

Test-Skript Statistik

```

1  echo "\nRedaction Summary\n";
2  $totalRedactions = array_sum($counters);
3  if ($totalRedactions === 0)
4  {
5      echo "No redactions found\n";
6  }
7  else
8  {
9      foreach ($counters as $type => $count)
10      {
11          if ($count > 0)
12          {
13              echo str_pad($type . ': ', 25) . $count . "\n";
14          }
15      }
16      echo str_pad('TOTAL: ', 25) . $totalRedactions . "\n";
17  }
18
19  $stats = $result['statistics'];
20

```

```

21 echo "\nStatistics\n";
22 echo str_pad('Time-Frame:', 25) . $from->format('Y-m-d H:i:s')
   . ' to ' . $to->format('Y-m-d H:i:s') . "\n";
23 echo str_pad('Fetched:', 25) . $stats['total_fetched'] . "\n";
24 echo str_pad('Processed:', 25) . $stats['total_processed'] . "\n";
25 echo str_pad('Batches:', 25) . $stats['batches_processed'] . "\n";
26 $ms = $stats['processing_time_ms'];
27 $seconds = round($ms / 1000, 3);
28 echo str_pad('Duration of Filtering Process: ', 25) . $ms . "
   ms ({ $seconds} s)\n";
29 echo str_pad('Filtering Logic Time:', 25) . $stats['
   filtering_only_ms'] . " ms\n";
30
31 if (!empty($result['statistics']['errors']))
32 {
33     echo "\nErrors:\n";
34     foreach ($stats['errors'] as $error)
35     {
36         echo " - $error\n";
37     }
38 }

```

Der vollständige Code dieser Klasse ist im Anhang A.18 einsehbar.

Zur Validierung der Effizienz und Zuverlässigkeit des Logfiltersystems wurden mittels des beschriebenen Test-Skripts Testläufe 3 verschiedenen Log-Typen durchgeführt: Nextcloud, NGINX Access und NGINX Error Logs. Die Tests wurden jeweils über einen Zeitraum von zwei Minuten durchgeführt.

Nextcloud Statistik

```

1 Redaction Summary
2 PASSWORD: 4
3 TOKEN: 1688
4 API KEY: 3
5 SECRET: 14
6 AUTHORIZATION: 37
7 IP ADDRESS: 69507
8 HOSTNAME: 64406
9 TOTAL: 135659
10
11 Statistics
12 Time-Frame: 2026-04-14 11:24:38 to 2026-04-14
   11:26:38
13 Fetched: 64267
14 Processed: 64267
15 Batches: 26
16 Duration of Filtering Process: 14749.1 ms (14.749 s)
17 Filtering Logic Time: 7419.49 ms

```

NGINX Access Statistik

1	Redaction Summary	
2	PASSWORD:	3
3	TOKEN:	268
4	API KEY:	2
5	SESSION:	10
6	IP ADDRESS:	978239
7	HOSTNAME:	1956478
8	TOTAL:	2935000
9		
10	Statistics	
11	Time-Frame:	2026-04-14 09:04:02 to 2026-04-14 09:06:02
12	Fetched:	978239
13	Processed:	978239
14	Batches:	392
15	Duration of Filtering Process:	153256.12 ms (153.256 s)
16	Filtering Logic Time:	81226.88 ms

NGINX Error Statistik

1	Redaction Summary	
2	IP ADDRESS:	2374
3	HOSTNAME:	7013
4	TOTAL:	9387
5		
6	Statistics	
7	Time-Frame:	2026-04-14 12:46:43 to 2026-04-14 12:48:43
8	Fetched:	2374
9	Processed:	2374
10	Batches:	1
11	Duration of Filtering Process:	323.25 ms (0.323 s)
12	Filtering Logic Time:	96.08 ms

Die Ergebnisse der Testläufe zeigen, dass das entwickelte Logfiltersystem auch bei stark variierenden Mengen an Logeinträgen stabil und effizient arbeitet. In allen drei Szenarien wurden sämtliche abgerufenen Logeinträge vollständig verarbeitet, was daran erkennbar ist, dass die Anzahl Fetched jeweils mit der Anzahl Processed übereinstimmt. Dies weist darauf hin, dass während der Verarbeitung keine Logeinträge verloren gingen und die Filterpipeline zuverlässig arbeitet.

Hinsichtlich der Performance zeigt sich, dass die Verarbeitungsdauer erwartungsgemäß mit der Menge der zu verarbeitenden Logeinträge ansteigt. Während bei den NGINX Error Logs 2.374 Logeinträge in 0,323 Sekunden verarbeitet wurden, wurden bei den Nextcloud-Logs 64.267 Einträge in 14,749 Sekunden verarbeitet. Die umfangreichsten Werte lagen bei den NGINX Access Logs vor, bei denen 978.239 Logeinträge in 153,256 Sekunden verarbeitet wurden. Die Unterschiede in den Laufzeiten korrespondieren mit den jeweils verarbeiteten Mengen

an Logeinträgen und bestätigen, dass das System auch bei steigender Last konsistent arbeitet.

Dies wird zusätzlich durch die ermittelte Verarbeitungsgeschwindigkeit gestützt. Bei den Nextcloud-Logs wurden durchschnittlich etwa 4.357 Logeinträge pro Sekunde verarbeitet, bei den NGINX Access Logs etwa 6.383 Logeinträge pro Sekunde und bei den NGINX Error Logs etwa 7.349 Logeinträge pro Sekunde. Die Unterschiede zwischen den Logtypen lassen sich insbesondere durch Unterschiede in Struktur, Komplexität und Filteraufwand der jeweiligen Logtypen erklären.

Auch die *Redaction Summary* bestätigt die Funktionsfähigkeit der Filtermechanismen. Sensible Informationen wurden in allen relevanten Kategorien erkannt und gefiltert. Insbesondere bei IP-Adressen und Hostnamen zeigen die hohen Filterzahlen, dass netzwerkbezogene Informationen zuverlässig identifiziert wurden. Gleichzeitig belegen erkannte Credentials wie Tokens, Secrets oder Passwörter die korrekte Funktion sensibler credential-bezogener Filter.

Insgesamt bestätigen die Ergebnisse sowohl die funktionale Korrektheit als auch die Skalierbarkeit des Logfiltersystems unter praxisnahen Bedingungen. Die Integrationstests ergänzen damit die zuvor durchgeführten Unit-Tests um einen Nachweis des Verhaltens unter realistischen Bedingungen.

5 Evaluation und Validierung

Dieses Kapitel bewertet, inwieweit das implementierte Logfiltersystem die in Kapitel 3.1 definierten Anforderungen erfüllt. Die Evaluation der funktionalen und nicht-funktionalen Anforderungen stützt sich auf die Ergebnisse der durchgeführten Unit-Tests sowie auf Messungen unter realistischen Bedingungen mit einer angebundenen OpenSearch-Instanz.

5.1 Erfüllung der funktionalen Anforderungen

FA-01 fordert den zeitraumbasierten Abruf von Logeinträgen aus OpenSearch anhand eines Start- und Enddatums im ISO-8601-Format. Die Anforderung wird durch den `OpenSearchService` umgesetzt, der auf Basis der übergebenen Zeitparameter eine entsprechende Range-Query gegen die OpenSearch-Instanz erzeugt. Ausgegeben werde hierbei ausschließlich Logeinträge, deren Zeitstempel innerhalb des definierten Zeitraums liegen. Die Bedingung, dass das Startdatum vor dem Enddatum liegt, wird durch die Semantik des Range-Filters von OpenSearch sichergestellt, der bei umgekehrter Reihenfolge keine Treffer zurückgibt.

FA-02 fordert die Identifikation und Ersetzung sensibler Informationen in strukturierten Feldern durch einen festgelegten Platzhalter, wobei die zu filternden Feldnamen konfigurierbar definiert sein müssen. Die Anforderung wird durch die feldnamenbasierte Direkterkennung in allen Filterklassen erfüllt. Die Feldnamen sind zentral in `FieldsToFilterConfig` konfigurierbar definiert. Die Testklassen `CredentialsFilterTest`, `NetworkFilterTest` und `InfrastructureFilterTest` verifizieren diese Funktionalität für verschiedene Varianten der Feldnamen, einschließlich verschachtelter Datenstrukturen.

FA-03 fordert die Erkennung und Filterung sensibler Informationen innerhalb von unstrukturierten Freitextfeldern mittels regulärer Ausdrücke. Die Anforderung wird durch die Verwendung vorkompilierter regulärer Ausdrücke umgesetzt, wodurch eine effiziente Mustererkennung innerhalb unstrukturierter Logs ermöglicht wird. Die Korrektheit dieser Funktionalität wird durch die Testklassen `CredentialsFilterTest`, `NetworkFilterTest`, `InfrastructureFilterTest` sowie `FilterPipelineTest` mittels Unit-Tests verifiziert, welche sicherstellen, dass eingebettete sensible Informationen erkannt und ersetzt werden, während der übrige Text unverändert bleibt.

FA-04 fordert die Bereitstellung der gefilterten Logeinträge in einem konsistenten JSON-Format zur Weiterverarbeitung. Die Anforderung wird durch den `LogFilterService` erfüllt, der die gefilterten Logeinträge entgegennimmt und in einem JSON-kompatiblen Array zurückgibt.

FA-05 fordert die Bereitstellung einer Verarbeitungsstatistik über die verarbeiteten Logeinträge sowie die erkannten und gefilterten Inhalte zur Sicherstellung der Nachvollziehbarkeit und Qualität der Filterung. Die Anforderung wird durch den `LogFilterService` erfüllt, der nach Abschluss der Verarbeitung eine strukturierte Statistik mit relevanten Metadaten zurückliefert. Diese umfasst die Anzahl abgerufener und verarbeiteter Logeinträge, die Anzahl verarbeiteter Batches, die Gesamtdauer der

Verarbeitung und die Gesamtdauer der reinen Filterung. Zusätzlich enthält die Statistik den betrachteten Zeitraum der Verarbeitung in Form eines Start- und Endzeitpunktes im ISO-8601-Format sowie eine Liste aufgetretener Fehler. Die Bedingung der Korrektheit und Vollständigkeit der Statistik wird durch das direkte Inkrementieren der Zähler innerhalb der Verarbeitungsschleife sowie durch die zeitliche Messung der einzelnen Verarbeitungsschritte sichergestellt. Insbesondere werden sowohl die Gesamtverarbeitungszeit als auch die reine Filterzeit getrennt erfasst, um eine differenzierte Analyse der Performance zu ermöglichen.

5.2 Erfüllung der nicht-funktionalen Anforderungen

NFA-01 fordert, dass neue Filterkategorien hinzugefügt werden können, ohne bestehende Komponenten ändern zu müssen. Die Anforderung wird durch das Zusammenspiel mehrerer Architekturentscheidungen erfüllt. Zentrale Grundlage bildet das `FilterInterface`, das einen gemeinsamen Vertrag für alle Filterklassen definiert. Dadurch können Filter in der `FilterPipeline` unabhängig von ihrer konkreten Implementierung verarbeitet und dynamisch eingebunden werden. Zusätzlich wird die abstrakte Klasse `AbstractFilter` eingesetzt, um wiederverwendbare Funktionalität für alle Filter bereitzustellen. Dadurch beschränkt sich die Implementierung neuer Filter auf die jeweilige fachliche Logik der Filterkategorie. Die zentrale Konfigurationsklasse `FieldsToFilterConfig` ermöglicht es zudem, neue Feldnamen an einer einzigen Stelle zu ergänzen, ohne bestehende Filterklassen anpassen zu müssen. Das MVCS Architekturmuster stellt darüber hinaus sicher, dass die Services von Konfiguration und Model entkoppelt sind, was eine unabhängige Entwicklung und Testbarkeit neuer Filterklassen ermöglicht.

NFA-02 fordert eine Verarbeitungsgeschwindigkeit von mindestens 1000 Logeinträgen pro Sekunde. Die Anforderungen werden durch zwei Maßnahmen erfüllt. Zum einen verwendet der `OpenSearchService` die Scroll-API in Kombination mit einem PHP-Generator, sodass Logeinträge einzeln verarbeitet werden, ohne die gesamten Logdaten im Arbeitsspeicher zu halten. Weiterhin werden die regulären Ausdrücke einmal beim Instanzieren der Filterklassen kompiliert, sodass kein Kompilierungsaufwand pro Logeintrag entsteht. Die Validierung der geforderten Verarbeitungsgeschwindigkeit erfolgte über die in Abschnitt 4.7 beschriebenen Testläufe mit Nextcloud-, NGINX-Access- und NGINX-Error-Logs. Die dort ermittelten Verarbeitungsgeschwindigkeiten von ca. 4357 bis 7349 Logeinträgen pro Sekunde liegen in allen Testfällen deutlich über der geforderten Mindestleistung von 1000 Logeinträgen pro Sekunde und bestätigen somit die Erfüllung von NFA-02.

NFA-03 fordert die isolierte Unit-Testbarkeit der Filterklassen ohne aktive `OpenSearch`-Instanz. Die Anforderung wird durch die klare Trennung der Komponenten erfüllt. Da die Filterklassen ausschließlich `LogEntry`-Objekte entgegennehmen und zurückgeben, können sie vollständig isoliert ohne aktive `OpenSearch`-Instanz getestet werden. Die Abhängigkeit zu `OpenSearch` ist ausschließlich im `OpenSearchService` und `OpenSearchApiClient` gekapselt.

NFA-04 fordert einen klar strukturierten und dokumentierten Quellcode. Die

Anforderung wird durch die konsequente Anwendung des MVCS Architekturmodells sowie die Dokumentation des geschriebenen Codes erfüllt.

5.3 Bekannte Einschränkungen

Trotz der erreichten Funktionalität und der modularen Erweiterbarkeit weist das entwickelte Logfiltersystem einige Einschränkungen auf, die sich sowohl aus dem gewählten methodischen Ansatz als auch aus der gewählten Filterlogik ergeben.

Die erste Einschränkung betrifft die Erkennung sensibler Informationen in unstrukturierten Freitextfeldern. Das System kombiniert eine feldnamenbasierte Direkterkennung mit einer musterbasierten Erkennung auf Grundlage regulärer Ausdrücke. Während beide Verfahren für strukturierte Daten sowie syntaktisch klar erkennbare Muster zuverlässig arbeiten, stößt dieser Ansatz bei der Filterung von sensiblen Informationen in Freitextfeldern ohne eindeutig identifizierbare Struktur an seine Grenzen. Dies betrifft insbesondere Inhalte wie Passwörter und Tokens, sofern diese nicht über bekannte Feldnamen adressiert werden können noch aufgrund fehlender fester Struktur zuverlässig durch Muster erkannt werden können. Dies liegt daran, dass Passwörter und vergleichbare Inhalte im Gegensatz zu Informationen wie IPv4- oder IPv6-Adressen keiner festen syntaktisch klar beschreibbaren Struktur folgen, sondern grundsätzlich aus einer beliebigen Kombination und Anzahl von Zeichen bestehen können. Eine eindeutige Unterscheidung zwischen einem Passwort und einer gewöhnlichen Zeichenfolge ist daher anhand rein syntaktischer beziehungsweise musterbasierter Verfahren nicht zuverlässig möglich.

Die zweite Einschränkung betrifft die feldnamenbasierte Erkennung sensibler Daten. Diese basiert auf einer vordefinierten Konfiguration in der Klasse `FieldToFilterConfig`, in der bekannte sensible Feldnamen zentral hinterlegt werden. Dieses Vorgehen ermöglicht eine gezielte Direkterkennung, setzt jedoch voraus, dass relevante Feldnamen im Vorfeld bekannt und hinterlegt sind. Treten in neuen oder veränderten Logstrukturen bislang unbekannte Feldbezeichnungen auf, werden diese nicht automatisch gefiltert. Die Qualität dieses Mechanismus hängt somit von der Vollständigkeit und Pflege der Konfiguration ab.

Darüber hinaus ist bei der Änderung von Logformaten oder bei der Ergänzung von zusätzlichen Logarten eine regelmäßige Überprüfung und Pflege der hinterlegten Feldnamen erforderlich. Diese Einschränkung ist jedoch zugleich eine bewusste Folge des gewählten Architekturansatzes, da die zentrale Konfiguration eine gezielte Erweiterung ermöglicht, ohne bestehende Filterlogik anpassen zu müssen.

6 Fazit und Ausblick

6.1 Fazit

Im Rahmen dieser Arbeit wurde folgende Forschungsfrage untersucht:

„Wie kann ein automatisiertes, erweiterbares Filtersystem für Nextcloud und OpenSearch implementiert werden, das infrastrukturspezifische und sicherheitskritische Informationen in Logs zuverlässig erkennt und entfernt, ohne die technische Aussagekraft der Logs zu beeinträchtigen?“

Die Umsetzung zeigt, dass ein automatisiertes und erweiterbares Filtersystem durch eine modulare Pipeline-Architektur realisiert werden kann, in der Logs über OpenSearch effizient abgerufen, verarbeitet und gefiltert werden. Durch die Kombination aus feldbasierten und musterbasierten Filtermechanismen können sowohl strukturierte als auch unstrukturierte sensible Informationen erkannt und entfernt werden, ohne die strukturelle und technische Aussagekraft der Logs wesentlich zu beeinträchtigen.

Ziel dieser Bachelorarbeit bestand in der Konzeption und Umsetzung eines modularen Logfiltersystems zur automatisierten Erkennung und Filterung sensibler Informationen in Logs. Dieses Ziel konnte im Rahmen der Bachelorarbeit erreicht werden, indem ein System entwickelt wurde, das die in der Anforderungsanalyse definierten funktionalen und nicht-funktionalen Anforderungen adressiert und in eine technische Lösung überführt.

Das entwickelte Logfiltersystem ermöglicht einen zeitraumbasierten Abruf von Logeinträgen aus OpenSearch, verarbeitete diese über eine sequenzielle Pipeline und stellt die gefilterten Ergebnisse in einem strukturierten JSON-Format bereit. Durch die Kombination aus felddatenbasierter Direkterkennung und musterbasierter Erkennung mittels regulärer Ausdrücke können sowohl strukturierte sensible Daten als auch eingebettete Muster in Freitextfeldern erkannt und gefiltert werden.

Die Entscheidung für das Pipes-and-Filters Architekturmusters sowie das MVCS Architekturmusters für die Gesamtarchitektur hat sich als geeignet erwiesen. Die klare Trennung der Komponenten und Verantwortlichkeiten erleichtert sowohl die Erweiterbarkeit des Logfiltersystems um neue Filterkategorien als auch die Testbarkeit der einzelnen Komponenten ohne Abhängigkeit zu externen Systemen.

Die im Rahmen der Konzeption identifizierten Risiken und Schwierigkeiten (R-01 bis R-05) konnten unterschiedlich stark adressiert werden.

Das Risiko der unvollständigen Filterung (R-01) lässt sich nicht vollständig ausschließen. Zwar können strukturierte Informationen zuverlässig erkannt werden, jedoch ist eine vollständige Erfassung sensibler Informationen insbesondere bei stark variierenden oder unbekannten Logstrukturen nicht garantiert, da rein syntaktische Verfahren keine vollständige semantische Erkennung ermöglichen.

Dem gegenüber steht das Risiko einer zu starken Filterung (R-02), das durch eine bewusste Einschränkung der Erkennungslogik auf eindeutig identifizierbare Muster

reduziert wurde. Dadurch wird vermieden, dass technisch relevante Informationen unbeabsichtigt entfernt werden.

Die Unterstützung unterschiedlicher Log-Formate (R-03) wurde durch einen modularen Aufbau des Filtersystems berücksichtigt. Diese ermöglicht es, unterschiedliche Logstrukturen durch spezialisierte Komponenten zu verarbeiten, ohne eine einheitliche Datenstruktur vorauszusetzen.

Das Performance-Risiko (R-04) wurde durch Maßnahmen wie der Verwendung der OpenSearch Scroll-API reduziert. Dadurch werden Logeinträge in kleineren, sequentiellen Einheiten verarbeitet, was insbesondere bei großen Logdatenmengen eine speichereffiziente Verarbeitung ermöglicht. Zusätzlich werden Filteroperationen durch vorkompilierte reguläre Ausdrücke umgesetzt, wodurch der Aufwand der Mustererkennung pro Logeintrag reduziert wird. Die Verarbeitung erfolgt damit ohne wiederholte Initialisierungskosten innerhalb der Filterpipeline.

Die zeitliche Begrenzung des Projekts (R-05) hat den Implementierungsumfang eingeschränkt, dennoch konnte ein funktionsfähiges Logfiltersystem innerhalb des vorgegebenen Zeitrahmens erfolgreich entwickelt und umgesetzt werden. Die grundlegende Architektur wurde dabei so gestaltet, dass zentrale Anforderungen erfüllt werden und gleichzeitig eine spätere Erweiterung des Systems weiterhin möglich bleibt.

Mit der Umsetzung als Composer-Package wurde zudem das Ziel erreicht, das entwickelte System unabhängig von einer spezifischen Laufzeitumgebung nutzbar zu machen. Das ergänzende `LogfilterBundle` bietet zudem für Symfony-Anwendungen eine direkt einsetzbare Integrationsmöglichkeit inklusive konfigurierbarer Parameter.

Zusammenfassend zeigt die Arbeit, dass ein regelbasiertes, modular aufgebautes Filtersystem einen praktikablen Ansatz zur automatisierten Filterung sensibler Informationen in Logs darstellt. Die gewählte Architektur bildet eine solide Grundlage, auf der weiterführende Erkennungsverfahren und zusätzliche Log-Typen ergänzt werden können.

6.2 Ausblick

Das entwickelte System bietet mehrere Ansatzpunkte für zukünftige Erweiterungen und Verbesserungen.

Eine naheliegende Erweiterung ist die Unterstützung zusätzlicher Log-Typen. Im Rahmen dieser Arbeit wurden ausschließlich Nextcloud- und NGINX-Logs betrachtet. Da die modulare Architektur des Logfiltersystems auf klar getrennten Komponenten aufbaut, lassen sich weitere Log-Formate integrieren, indem entsprechende Filterklassen ergänzt werden, ohne bestehende Komponenten grundlegend verändern zu müssen.

Ein weiterer Ansatz betrifft die Weiterentwicklung der Erkennung sensibler Informationen außerhalb strukturierter Felder. Da Informationen wie Passwörter oder Tokens sich syntaktisch nicht eindeutig von gewöhnlichen Zeichenfolgen unterscheiden, lassen diese sich nicht allein mit regelbasierten Verfahren zuverlässig

erfassen. Der Einsatz von Verfahren des maschinellen Lernens könnte hier einen Ansatz bieten, bei dem sowohl Bedeutung als auch Kontext einbezogen werden kann, anstatt sich ausschließlich auf syntaktischen Merkmalen zu stützen.

Schließlich wäre die Weiterentwicklung der konfigurationsbasierten Felderkennung sinnvoll. Derzeit setzt die zuverlässige Erkennung sensibler Feldnamen voraus, dass diese vorab in der zentralen Konfiguration hinterlegt sind. Insbesondere beim Einsatz in dynamischen Umgebungen, in denen Log-Formate häufig variieren oder neue Felder hinzukommen, entsteht dadurch ein kontinuierlicher Pflegeaufwand. Ergänzend zur manuellen Konfiguration wäre eine automatische Erkennung potenziell sensibler Felder denkbar.

7 Quellenverzeichnis

Literatur

- [1] S. Kariyattin, S. Marru und M. Pierce. „Evaluating NextCloud as a File Storage for Apache Airavata“. In: *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18)*. Association for Computing Machinery, 2018, S. 2. DOI: 10.1145/3219104.3229270. URL: <https://dl.acm.org/doi/epdf/10.1145/3219104.3229270> (besucht am 09.02.2026).
- [2] N. Rücker. „Log File Processing - Changes, Challenges, and Chances“. Diss. Friedrich-Alexander-Universität, Erlangen-Nürnberg, 2022, S. 2, 15. URL: <https://open.fau.de/server/api/core/bitstreams/3d33d66e-ddd0-4115-84ef-adb993191a5a/content> (besucht am 10.02.2026).
- [3] Laaziri et al. „A Comparative Study of Laravel and Symfony PHP Frameworks“. In: *International Journal of Electrical and Computer Engineering* (2018), S. 705. ISSN: 2088-8708. DOI: 10.11591/ijece.v9i1.pp704-712. URL: <https://ijece.iaescore.com/index.php/IJECE/article/view/11601/11103> (besucht am 16.03.2026).
- [4] S. Kuppusamy. *Mastering OpenSearch: A Comprehensive Guide*. 1. Aufl. 2025, S. 1.
- [5] The Apache Software Foundation. *Apache HTTP Server*. URL: <https://httpd.apache.org/> (besucht am 19.04.2026).
- [6] NGINX Team. *NGINX*. URL: <https://nginx.org/en/> (besucht am 19.04.2026).
- [7] Oracle Corporation. *MySQL*. URL: <https://www.mysql.com/de/> (besucht am 19.04.2026).
- [8] OpenSearch Team. *Query DSL*. URL: <https://docs.opensearch.org/latest/query-dsl/> (besucht am 27.02.2026).
- [9] OpenSearch Team. *Range Query*. URL: <https://docs.opensearch.org/latest/query-dsl/term/range/> (besucht am 27.02.2026).
- [10] OpenSearch Team. *Scroll API*. URL: <https://docs.opensearch.org/latest/api-reference/search-apis/scroll/> (besucht am 27.02.2026).
- [11] E. Doberkat. *Pipes and filters: Modelling a software architecture through relations*. URL: <https://eldorado.tu-dortmund.de/server/api/core/bitstreams/da2d8ef9-5ab6-4b6a-8ea5-9600e2fc92c4/content> (besucht am 09.02.2026).

- [12] A. Bubeck. „Modellbasierte Softwareentwicklung für mobile Manipulatoren im industriellen Einsatz“. Deutsch. Diss. Universität Stuttgart, 2015, S. 32. DOI: 10.18419/opus-8823. URL: <https://d-nb.info/1118370791/34>.
- [13] T. Reenskaug. *MODELS - VIEWS - CONTROLLERS*. 1979. URL: https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf (besucht am 09. 02. 2026).
- [14] Composer Team. *Composer Introduction*. URL: <https://getcomposer.org/doc/00-intro.md> (besucht am 27. 02. 2026).
- [15] Composer Team. *Basic usage*. URL: <https://getcomposer.org/doc/01-basic-usage.md> (besucht am 19. 04. 2026).
- [16] PHP-FIG. *PHP PSR-4*. URL: <https://www.php-fig.org/psr/psr-4/> (besucht am 16. 03. 2026).
- [17] Composer Team. *Repositories - Composer*. URL: https://getcomposer.org/doc/05-repositories.md?utm_source=chatgpt.com#package (besucht am 09. 04. 2026).
- [18] Jeffrey E. F. Friedl. *Reguläre Ausdrücke*. 3. Aufl. O'Reilly, 2008, S. 1.
- [19] L. Chazette. „Requirements Engineering for Explainable Systems“. Diss. Gottfried Wilhelm Leibniz Universität Hannover, 2023, S. 10. URL: <https://repo.uni-hannover.de/items/e822e738-42fd-4ccf-99c7-4a289de1a652> (besucht am 16. 03. 2026).
- [20] International Organization for Standardization. *ISO 8601-1:2019 Date and time — Representations for information interchange — Part 1: Basic rules*. Techn. Ber. 2019. URL: <https://www.iso.org/obp/ui/en/#iso:std:iso:8601:-1:ed-1:v1:en> (besucht am 07. 04. 2026).
- [21] International Organization for Standardization. *ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model*. Techn. Ber. 2023. URL: <https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en> (besucht am 07. 04. 2026).
- [22] Westsächsischen Hochschule Zwickau. *Prüfungsordnung für den Bachelorstudiengang Informatik*. 2013. URL: https://www.whz.de/fileadmin/lehre/kurskatalog/docs/his/PTI/IF_B/BP0_IF_2013.pdf (besucht am 20. 04. 2026).
- [23] PHP Team. *Passing by Reference*. URL: <https://www.php.net/manual/en/language.references.pass.php> (besucht am 24. 03. 2026).
- [24] PHP Team. *Generators overview*. URL: <https://www.php.net/manual/en/language.generators.overview.php> (besucht am 02. 03. 2026).

- [25] PHP Team. *PHP preg_quote*. URL: <https://www.php.net/manual/de/function.preg-quote.php> (besucht am 02.03.2026).
- [26] PHP Team. *PHP array_merge*. URL: <https://www.php.net/manual/en/function.array-merge.php> (besucht am 03.03.2026).
- [27] PHP Team. *PHP array_unique*. URL: <https://www.php.net/manual/de/function.array-unique.php> (besucht am 03.03.2026).
- [28] Symfony Team. *The Bundle System*. URL: <https://symfony.com/doc/current/bundles.html> (besucht am 19.04.2026).
- [29] Symfony Team. *The DependencyInjection Component*. URL: https://symfony.com/doc/current/components/dependency_injection.html (besucht am 19.04.2026).
- [30] Digital Science. *Overleaf – Online LaTeX Editor*. URL: <https://www.overleaf.com/> (besucht am 24.03.2026).
- [31] LanguageTooler GmbH. *LanguageTool: Kostenlose KI-Rechtschreibprüfung*. URL: <https://languagetool.org/de> (besucht am 24.03.2026).
- [32] JetBrains. *PhpStorm*. URL: <https://www.jetbrains.com/de-de/phpstorm/> (besucht am 24.03.2026).
- [33] GitLab Inc. *GitLab*. URL: <https://about.gitlab.com/> (besucht am 30.03.2026).
- [34] JabRef Team. *JabRef*. URL: <https://www.jabref.org/> (besucht am 19.04.2026).
- [35] draw.io Ltd. *draw.io*. URL: <https://www.drawio.com/> (besucht am 20.04.2026).

8 Verwendete Hilfsmittel

Die Bachelorarbeit wurde mit Overleaf [30] geschrieben.

Für die Rechtschreibprüfung wurde LanguageTool [31] verwendet.

Das Logfiltersystem wurde mit PhpStorm [32] entwickelt.

Für die Versionsverwaltung wurde GitLab [33] verwendet.

Für die Verwaltung der Quellen wurde JabRef [34] verwendet.

Für die Erstellung des Systemkomponenten-Überblicks wurde draw.io [35] verwendet.

A Code des Logfiltersystems

A.1 FieldsToFilterConfig

FieldsToFilterConfig

```
1 <?php
2
3 namespace Logfilter\Config;
4
5 /**
6  * central configuration class defining sensitive field names
7  * that should be detected and redacted by the log filtering
8  * system.
9  */
10 class FieldsToFilterConfig
11 {
12     /**
13      * returns all password-related field names.
14      *
15      * @return string[]
16      */
17     public static function getPasswordFields(): array
18     {
19         return
20         [
21             'password',
22             'pwd',
23             'passwd',
24             'userPassword',
25             'smtPassword',
26             'smtp_password',
27             'mailPassword',
28             'mail_Password',
29             'passwordSalt',
30             'password_salt',
31             'salts',
32         ];
33     }
34
35     /**
36      * returns all token-related field names.
37      *
38      * @return string[]
39      */
40     public static function getTokenFields(): array
41     {
42         return
43         [
```

```

43         'token',
44         'authToken',
45         'auth_token',
46         'accessToken',
47         'access_token',
48         'bearer',
49         'bearerToken',
50         'shareToken',
51         'share_token',
52         'isTokenLogin',
53     ];
54 }
55
56 /**
57  * returns all API key-related field names.
58  *
59  * @return string[]
60  */
61 public static function getApiKeyFields(): array
62 {
63     return
64     [
65         'apiKey',
66         'api_key',
67         'key'
68     ];
69 }
70
71 /**
72  * returns all session-related field names.
73  *
74  * @return string[]
75  */
76 public static function getSessionFields(): array
77 {
78     return
79     [
80         'session',
81         'sessionId',
82         'session_id',
83         'PHPSESSID',
84     ];
85 }
86
87 /**
88  * returns all secret-related field names.
89  *
90  * @return string[]
91  */

```

```

92 public static function getSecretFields(): array
93 {
94     return
95     [
96         'secret',
97         'secretKey',
98         'secret_key',
99         'encryptionKey',
100        'encryption_key',
101    ];
102 }
103
104 /**
105  * returns all authorization-related field names.
106  *
107  * @return string[]
108  */
109 public static function getAuthorizationFields(): array
110 {
111     return
112     [
113         'authorization',
114         'auth',
115    ];
116 }
117
118 /**
119  * returns all database-related field names.
120  *
121  * @return string[]
122  */
123 public static function getDatabaseFields(): array
124 {
125     return
126     [
127         'dbPassword',
128         'db_password',
129         'databasePassword',
130         'database_password',
131         'dbUser',
132         'db_user',
133         'databaseUser',
134         'database_user',
135         'dbHost',
136         'db_host',
137         'databaseHost',
138         'database_host',
139         'dbPort',
140         'db_port',

```

```

141         'dbType',
142         'db_type',
143         'dbName',
144         'db_name',
145     ];
146 }
147
148 /**
149  * returns all IP address-related field names.
150  *
151  * @return string[]
152  */
153 public static function getIpAddressFields(): array
154 {
155     return
156     [
157         'ip',
158         'ipAddress',
159         'ip_address',
160         'remoteAddress',
161         'remote_address',
162         'remoteAddr',
163         'remote_addr',
164         'upstreamAddr',
165         'upstream_addr',
166         'remoteIP',
167         'remote_ip',
168         'remote ip',
169         'client'
170     ];
171 }
172
173 /**
174  * returns all MAC address-related field names.
175  *
176  * @return string[]
177  */
178 public static function getMacFields(): array
179 {
180     return
181     [
182         'macAddress',
183         'mac_address',
184         'macAddr',
185         'mac_addr',
186         'mac'
187     ];
188 }
189

```

```

190  /**
191  * Returns all host-related field names.
192  *
193  * @return string[]
194  */
195  public static function getHostFields(): array
196  {
197      return
198          [
199              'host',
200              'hostname',
201              'host_name',
202              'serverHost',
203              'server_host',
204              'server',
205          ];
206  }
207
208  /**
209  * returns all sensitive credential-related fields.
210  *
211  * @return string[]
212  */
213  public static function getSensitiveCredentialFields():
214      array
215  {
216      return array_unique(array_merge
217          (
218              self::getPasswordFields(),
219              self::getTokenFields(),
220              self::getApiKeyFields(),
221              self::getSessionFields(),
222              self::getSecretFields(),
223              self::getAuthorizationFields(),
224          ));
225  }
226
227  /**
228  * returns all sensitive infrastructure-related fields.
229  *
230  * @return string[]
231  */
232  public static function getSensitiveInfrastructureFields():
233      array
234  {
235      return array_unique(array_merge
236          (
237              self::getDatabaseFields(),

```

```

237     }
238
239     /**
240      * returns all sensitive network-related fields.
241      *
242      * @return string[]
243      */
244     public static function getSensitiveNetworkFields(): array
245     {
246         return array_unique(array_merge
247         (
248             self::getIpAddressFields(),
249             self::getMacFields(),
250             self::getHostFields(),
251         ));
252     }
253
254     /**
255      * determines whether a field is considered sensitive.
256      *
257      * @param string $fieldName Field name to evaluate
258      * @return bool true if the field is sensitive, false
259      *         otherwise
260      */
261     public static function isFieldSensitive(string $fieldName):
262         bool
263     {
264         $allFields = array_unique(array_merge
265         (
266             self::getSensitiveCredentialFields(),
267             self::getSensitiveInfrastructureFields(),
268             self::getSensitiveNetworkFields()
269         ));
270
271         return in_array(strtolower($fieldName), array_map('
272             strtolower', $allFields), true);
273     }
274
275     /**
276      * returns the classification type of a sensitive field.
277      *
278      * @param string $fieldName Field name to classify
279      */
280     public static function getFieldTypes(string $fieldName): ?
281         string
282     {
283         $standardizedField = strtolower($fieldName);

```

```

281         $credentialFields = array_map('strtolower', self::
            getSensitiveCredentialFields());
282         if(in_array($standardizedField, $credentialFields, true
            ))
283         {
284             return 'credential';
285         }
286
287         $infrastructureFields = array_map('strtolower', self::
            getSensitiveInfrastructureFields());
288         if(in_array($standardizedField, $infrastructureFields,
            true))
289         {
290             return 'infrastructure';
291         }
292
293         $networkFields = array_map('strtolower', self::
            getSensitiveNetworkFields());
294         if(in_array($standardizedField, $networkFields, true))
295         {
296             return 'network';
297         }
298         return null;
299     }
300 }

```

A.2 LogEntry

LogEntry

```

1 <?php
2
3 namespace Logfilter\Model;
4
5 /**
6  * represents a single log entry
7  */
8
9 class LogEntry
10 {
11     /**
12      * creates a new log entry instance
13      * @param string $id unique identifier of the log entry
14      * @param array $data array containing log data
15      */
16     public function __construct(private readonly string $id,
17                                 private array $data)
18     {

```

```

18     }
19
20
21     /**
22      * retrieves the unique identifier of the log entry
23      * @return string the log entry id
24      */
25     public function getId(): string
26     {
27         return $this->id;
28     }
29
30     /**
31      * retrieves the complete log data array
32      * @return array the log data
33      */
34     public function getData(): array
35     {
36         return $this->data;
37     }
38
39     /**
40      * overwrites the log data array
41      * @param array $data the new log data
42      */
43     public function setData(array $data): void
44     {
45         $this->data = $data;
46     }
47
48     /**
49      * @param string $key the field name
50      * @return mixed the field value
51      */
52     public function getField(string $key, mixed $default = null
53         ): mixed
54     {
55         return $this->data[$key] ?? $default;
56     }
57
58     /**
59      * set a specific field in the log data
60      * @param string $key the field name
61      * @param mixed $value the value to assign
62      */
63     public function setField(string $key, mixed $value): void
64     {
65         $this->data[$key] = $value;
66     }

```

```

66
67     /**
68      * check if a specific field exists in the log data
69      * @param string $key the field name
70      * @return bool true if field exists, otherwise false
71      */
72     public function hasField(string $key): bool
73     {
74         return isset($this->data[$key]);
75     }
76
77     /**
78      * converts the log entry into an array
79      * @return array the array representation containing id and
80      *             data
81      */
82     public function toArray(): array
83     {
84         return
85         [
86             'id' => $this->id,
87             'data' => $this->data
88         ];
89     }

```

A.3 LogFilterBundle

LogFilterBundle

```

1  <?php
2
3  namespace Logfilter;
4
5  use Logfilter\Service\FilterPipeline;
6  use Logfilter\Service\LogFilterService;
7  use Logfilter\Service\OpenSearchApiClient;
8  use Logfilter\Service\OpenSearchService;
9  use Symfony\Component\Config\Definition\Configurator\
10     DefinitionConfigurator;
11 use Symfony\Component\DependencyInjection\ContainerBuilder;
12 use Symfony\Component\DependencyInjection\Loader\Configurator\
13     ContainerConfigurator;
14 use Symfony\Component\HttpKernel\Bundle\AbstractBundle;
15 use function Symfony\Component\DependencyInjection\Loader\
    Configurator\service;

```

```

16  * LogfilterBundle
17  *
18  * this Symfony bundle integrates the Logfiltersystem into
    Symfony applications.
19  * It provides configuration handling and registers all
    required services into
20  * the Dependency Injection container.
21  */
22  class LogfilterBundle extends AbstractBundle
23  {
24      /**
25       * defines the configuration structure for the Logfilter
        bundle.
26       *
27       * specifies all configurable parameters that must or can
        be
28       *
29       * @param DefinitionConfigurator $definition the
        configuration definition builder.
30       *
31       * @return void
32       */
33      public function configure(DefinitionConfigurator
        $definition): void
34      {
35          $definition->rootNode()
36              ->children()
37                  ->scalarNode('opensearch_url')->isRequired()->
                    end()
38                  ->scalarNode('opensearch_user')->isRequired()->
                    end()
39                  ->scalarNode('opensearch_password')->isRequired()
                    ->end()
40                  ->scalarNode('opensearch_index')->isRequired()
                    ->end()
41                  ->integerNode('batch_size')->defaultValue(2500)
                    ->end()
42                  ->scalarNode('scroll_window')->defaultValue('5m')
                    ->end()
43              ->end();
44      }
45
46
47      /**
48       * registers and configures all services required by the
        Logfiltersystem.
49       *
50       * @param array $config configuration values defined in the
        configure() method.

```

```

51      * @param ContainerConfigurator $container the Symfony
        service configurator.
52      * @param ContainerBuilder $builder the container builder
        instance.
53      *
54      * @return void
55      */
56      public function loadExtension(array $config,
        ContainerConfigurator $container, ContainerBuilder
        $builder): void
57      {
58          $container->services()
59              ->set(OpenSearchApiClient::class)
60                  ->args(
61                      [
62                          $config['opensearch_url'],
63                          $config['opensearch_user'],
64                          $config['opensearch_password'],
65                      ])
66              ->set(OpenSearchService::class)
67                  ->args([service(OpenSearchApiClient::class),
68                          $config['opensearch_index'], $config['
69                          batch_size'], $config['scroll_window']])
68              ->set(FilterPipeline::class)
69              ->set(LogFilterService::class)
70                  ->args([service(OpenSearchService::class),
71                          service(FilterPipeline::class), $config['
72                          batch_size']])
71                  ->public();
72      }
73  }

```

A.4 FilterPipeline

FilterPipeline

```

1  <?php
2
3  namespace Logfilter\Service;
4
5  use Logfilter\Service\Filter\FilterInterface;
6  use Logfilter\Model\LogEntry;
7
8  /**
9   * executes a sequence of filters on LogEntry objects
10  * filters are executed sequentially -> each filter receives
        the output of the previous filter
11  */

```

```

12 class FilterPipeline
13 {
14     private array $filters = [];
15     private array $statistics = [];
16
17     /**
18      * registers a filter for the pipeline
19      * @param FilterInterface $filter the filter to append
20      */
21     public function addFilter(FilterInterface $filter): self
22     {
23         $this->filters[] = $filter;
24         $this->statistics[$filter->getName()] =
25             [
26                 'enabled' => $filter->isEnabled(),
27                 'processed' => 0
28             ];
29         return $this;
30     }
31
32     /**
33      * processes a single LogEntry through all enabled filters
34      * @param LogEntry $logEntry the unfiltered input log entry
35      * @return LogEntry the log entry after being processed by
36      *         all the filters
37      */
38     public function process(LogEntry $logEntry): LogEntry
39     {
40         foreach ($this->filters as $filter)
41         {
42             if ($filter->isEnabled())
43             {
44                 $logEntry = $filter->process($logEntry);
45                 $this->statistics[$filter->getName()]['
46                     processed']++;
47             }
48         }
49         return $logEntry;
50     }
51
52     /**
53      * returns all registered filters
54      */
55     public function getFilters(): array
56     {
57         return $this->filters;
58     }
59
60     /**

```

```

59     * returns the defined statistics following the processing
60     */
61     public function getStatistics(): array
62     {
63         return $this->statistics;
64     }
65
66     /**
67     * resets filter processed counters back to zero
68     */
69     public function resetCounters(): void
70     {
71         foreach ($this->statistics as $filterName => $stats)
72         {
73             $this->statistics[$filterName]['processed'] = 0;
74         }
75     }
76
77     /**
78     * clears the tracked statistics
79     */
80     public function clearStatistics(): void
81     {
82         $this->statistics = [];
83     }
84 }

```

A.5 LogFilterService

LogFilterService

```

1  <?php
2
3  namespace Logfilter\Service;
4
5  use Exception;
6  use InvalidArgumentException;
7  use Logfilter\Service\FilterPipeline;
8  use Logfilter\Model\LogEntry;
9  use DateTimeInterface;
10
11  class LogFilterService
12  {
13      private int $batchSize;
14
15      /**
16       * @param OpenSearchService $openSearchService
17       * @param \Logfilter\Service\FilterPipeline $filterPipeline

```

```

18     * @param int $batchSize number of LogEntry objects than
        can be written per batch
19     */
20     public function __construct(private OpenSearchService
        $openSearchService, private FilterPipeline
        $filterPipeline, int $batchSize)
21     {
22         $this->batchSize = $batchSize;
23     }
24
25     /**
26     * processes logs within a given time frame
27     * fetches the logs, processes each log entry through the
        pipeline, writes the result in batches and collects the
        corresponding statistics
28     * @param DateTimeInterface $from the start time stamp
29     * @param DateTimeInterface $to the end stime stamp
30     * @return array
31     */
32     public function processLogs(DateTimeInterface $from,
        DateTimeInterface $to): array
33     {
34         $this->assertValidRange($from, $to);
35         $startTime = hrtime(true);
36         $totalFilterTime = 0;
37
38         $statistics =
39         [
40             'date_range' => ['from' => $from->format(\
                DateTimeInterface::ATOM), 'to' => $to->format(\
                DateTimeInterface::ATOM)],
41             'total_fetched' => 0,
42             'total_processed' => 0,
43             'batches_processed' => 0,
44             'errors' => [],
45             'processing_time_ms' => 0.0,
46             'filtering_time_ms' => 0.0,
47         ];
48
49         $filteredLogs = [];
50         $batch = [];
51
52         try
53         {
54             foreach($this->openSearchService->getLogs($from,
                $to) as $logEntry)
55             {
56                 $filterTime = hrtime(true);
57                 $statistics['total_fetched']++;

```

```

58         $filteredLogEntry = $this->filterPipeline->
59             process($logEntry);
60         $statistics['total_processed']++;
61
62         $batch[] = $filteredLogEntry;
63
64         if(count($batch) >= $this->batchSize)
65         {
66             $this->writeBatch($batch, $filteredLogs,
67                 $statistics);
68             $batch = [];
69         }
70
71         $totalFilterTime += (hrtime(true) - $filterTime
72             );
73     }
74
75     if(!empty($batch))
76     {
77         $this->writeBatch($batch, $filteredLogs,
78             $statistics);
79     }
80
81 }
82
83 catch(Exception $exception)
84 {
85     $statistics['errors'][] = $exception->getMessage();
86     error_log("Log processing error: " . $exception->
87         getMessage());
88 }
89
90 $statistics['processing_time_ms'] = round((hrtime(true)
91     - $startTime) / 1_000_000, 2);
92 $statistics['filtering_only_ms'] = round(
93     $totalFilterTime / 1_000_000, 2);
94
95 return
96 [
97     'statistics' => $statistics,
98     'filtered' => $filteredLogs
99 ];
100 }
101
102 /**
103  * writes a processed batch of log entries into the result
104  * collection.
105  *
106  * each LogEntry is converted into an array representation
107  * before being stored.

```

```

98     * the batch counter in the statistics array is incremented
99     *
100    * @param array $batch current batch of processed LogEntry
    objects.
101    * @param array &$filteredLogs reference to the final
    output array of filtered logs.
102    * @param array &$statistics reference to the processing
    statistics array.
103    *
104    * @return void
105    */
106    private function writeBatch(array $batch, array &
        $filteredLogs, array &$statistics): void
107    {
108        foreach ($batch as $logEntry)
109        {
110            $filteredLogs[] = $logEntry->toArray();
111        }
112        $statistics['batches_processed']++;
113    }
114
115    /**
116     * validates the provided time range.
117     *
118     *
119     * @param DateTimeInterface $from start timestamp.
120     * @param DateTimeInterface $to end timestamp.
121     *
122     * @return void
123     *
124     * @throws InvalidArgumentException if the start date is
        not before the end date.
125     */
126    private function assertValidRange(DateTimeInterface $from,
        DateTimeInterface $to): void
127    {
128        if ($from >= $to)
129        {
130            throw new InvalidArgumentException
131            (
132                sprintf('Invalid date range: start (%s) must be
                    before end (%s)', $from->format(
                        DateTimeInterface::ATOM), $to->format(
                            DateTimeInterface::ATOM))
133            );
134        }
135    }
136 }

```

A.6 OpenSearchApiClient

OpenSearchApiClient

```
1 <?php
2 namespace Logfilter\Service;
3
4 use OpenSearch\Client as OSClient;
5 use OpenSearch\ClientBuilder as OpenSearchClientBuilder;
6
7 class OpenSearchApiClient
8 {
9     private OSClient $osClient;
10
11     public function __construct(string $openSearchUrl, string
12         $openSearchUser, string $openSearchPassword)
13     {
14         try
15         {
16             $this->osClient = (new OpenSearchClientBuilder())
17                 ->setHosts([$openSearchUrl])
18                 ->setBasicAuthentication($openSearchUser,
19                     $openSearchPassword)
20                 ->includePortInHostHeader(true)
21                 ->build();
22         }
23         catch(\Exception $e)
24         {
25             throw new \RuntimeException('Failed to build
26                 OpenSearch client: ' . $e->getMessage());
27         }
28     }
29
30     public function getClient(): OSClient
31     {
32         return $this->osClient;
33     }
34
35     /**
36      * executes a search query against a specified OpenSearch
37      * index.
38      *
39      * @param string $index the name of the OpenSearch index to
40      * query.
41      * @param array $params query parameters including body and
42      * optional scroll settings.
43      *
44      * @return array the raw response returned by the
45      * OpenSearch client.
46      */
```

```

40     public function search(string $index, array $params): array
41     {
42         return $this->osClient->search(
43             [
44                 'index' => $index,
45                 'scroll' => $params['scroll'] ?? null,
46                 'body'   => $params['body']
47             ]
48         );
49     }

```

A.7 OpenSearchService

OpenSearchService

```

1  <?php
2
3  namespace Logfilter\Service;
4
5  use Exception;
6  use Logfilter\Model\LogEntry;
7  use DateTimeInterface;
8
9  /**
10   * handles the communication with OpenSearch
11   */
12  class OpenSearchService
13  {
14      private OpenSearchApiClient $client;
15      private string $sourceIndex;
16      private int $batchSize;
17
18      /**
19       * @param string $url the URL of the OpenSearch instance
20       * @param string $user the username for basic
21       * authentication
22       * @param string $password the password for basic
23       * authentication
24       * @param string $sourceIndex the index containing the
25       * unfiltered logs
26       * @param int $batchSize number of documents to fetch and
27       * write per request
28       */
29      public function __construct(string $url, string $user,
30                                  string $password, string $sourceIndex, int $batchSize)
31      {
32          $this->client = new OpenSearchApiClient($url, $user,
33                                                  $password);

```

```

28         $this->sourceIndex = $sourceIndex;
29         $this->batchSize = $batchSize;
30     }
31
32     /**
33      * fetches the log documents within the given time frame
34      * from the source index
35      * @param DateTimeInterface $from the start of the time
36      * range
37      * @param DateTimeInterface $to the end of the time range
38      */
39     public function getLogs(DateTimeInterface $from,
40         DateTimeInterface $to): \Generator
41     {
42         $response = $this->client->search($this->sourceIndex, [
43             'scroll' => '5m',
44             'body' =>
45             [
46                 'size' => $this->batchSize,
47                 'sort' => ['@timestamp' => 'asc'],
48                 'query' => $this->buildQuery($from, $to),
49             ]
50         ]);
51
52         $scrollId = $response['_scroll_id'] ?? null;
53
54         foreach ($response['hits']['hits'] ?? [] as $hit)
55         {
56             yield $this->createLogEntry($hit);
57         }
58
59         $searchClient = $this->client->getClient();
60
61         while ($scrollId && !empty($response['hits']['hits']))
62         {
63             $response = $searchClient->scroll(
64             [
65                 'scroll_id' => $scrollId,
66                 'scroll' => '5m',
67             ]
68         );
69
70             $scrollId = $response['_scroll_id'] ?? null;
71
72             foreach ($response['hits']['hits'] ?? [] as $hit)
73             {
74                 yield $this->createLogEntry($hit);
75             }
76         }
77     }

```

```

74         if ($scrollId)
75         {
76             try
77             {
78                 $searchClient->clearScroll(['scroll_id' =>
79                     $scrollId]);
80             }
81             catch (Exception $e)
82             {
83                 error_log("Failed to clear scroll: " . $e->
84                     getMessage());
85             }
86         }
87     }
88
89     /**
90     * builds an OpenSearch query that matches logs within
91     * given time frame
92     * @param DateTimeInterface $from the start of the time
93     * range
94     * @param DateTimeInterface $to the end of the time range
95     */
96     private function buildQuery(DateTimeInterface $from,
97         DateTimeInterface $to): array
98     {
99         return
100         [
101             'bool' =>
102             [
103                 'must' =>
104                 [
105                     [
106                         'range' =>
107                         [
108                             '@timestamp' =>
109                             [
110                                 'gte' => $from->format(
111                                     DateTimeInterface::ATOM),
112                                 'lte' => $to->format(
113                                     DateTimeInterface::ATOM)
114                             ],
115                         ],
116                     ],
117                 ],
118             ],
119         ];
120     }
121
122     /**

```

```

116     * creates a LogEntry instance from OpenSearch hit
117     * @param array $hit the single hit from an OpenSearch
        response
118     * @return LogEntry
119     */
120     private function createLogEntry(array $hit): LogEntry
121     {
122         return new LogEntry(id: $hit['_id'] ?? '', data: $hit['_source'] ?? [],);
123     }
124
125     public function getSourceIndex(): string
126     {
127         return $this->sourceIndex;
128     }
129
130     public function getBatchSize(): int
131     {
132         return $this->batchSize;
133     }
134 }

```

A.8 AbstractFilter

AbstractFilter

```

1 <?php
2
3 namespace Logfilter\Service\Filter;
4
5 /**
6  * specifies that all filters must implement process() and
        getName()
7  * general base class for all filters
8  */
9 use Logfilter\Model\LogEntry;
10
11 abstract class AbstractFilter implements FilterInterface
12 {
13
14     protected array $config;
15
16     public function __construct(array $config = [])
17     {
18         $this->config = array_merge(['enabled' => true],
            $config);
19     }
20

```

```

21  /**
22   * function used to define how the logs are being filtered
23   * / processed
24   */
25  abstract public function process(LogEntry $logEntry):
26      LogEntry;
27
28  /**
29   * retrieves the name of the filter
30   * @return string return the name of the filter
31   */
32  abstract public function getName(): string;
33
34  /**
35   * check if filter is enabled
36   * @return true or false
37   */
38  public function isEnabled(): bool
39  {
40      return $this->config['enabled'] ?? true;
41  }
42
43  protected function buildFieldNameRegex(array $fieldNames):
44      string
45  {
46      $escapedFields = array_map(function($field) {return
47          preg_quote($field, '/');}, $fieldNames);
48
49      return implode('|', $escapedFields);
50  }
51
52  /**
53   * recursively walks the data array
54   * if the key matches a known sensitive field, it gets
55   * redacted
56   * if the value is a plain string, it gets run through the
57   * $redactor callable for pattern-based redaction
58   * if the value is a nested array, recurse
59   *
60   * @param array $data the data to redact
61   * @param array $fieldLabelMap the field-to-label lookup
62   * map
63   * @param callable $redactor called with a string value
64   * when no direct key match is found
65   */
66  protected function redactData(array $data, array
67      $fieldLabelMap, callable $redactor): array
68  {
69      foreach($data as $key => $value)

```

```

61         {
62             $lowerKey = strtolower((string) $key);
63
64             if(is_array($value))
65             {
66                 $data[$key] = $this->redactData($value,
67                     $fieldLabelMap, $redactor);
68             }
69
70             elseif(isset($fieldLabelMap[$lowerKey]))
71             {
72                 $data[$key] = $fieldLabelMap[$lowerKey];
73             }
74
75             elseif(is_string($value))
76             {
77                 $data[$key] = $redactor($value);
78             }
79         }
80         return $data;
81     }

```

A.9 FilterInterface

FilterInterface

```

1  <?php
2
3  namespace Logfilter\Service\Filter;
4
5  use Logfilter\Model\LogEntry;
6
7  /**
8   * defines a contract which the filters must fulfill
9   */
10 interface FilterInterface
11 {
12     public function process(LogEntry $logEntry): LogEntry;
13
14     public function getName(): string;
15
16     public function isEnabled(): bool;
17 }

```

A.10 CredentialsFilter

CredentialsFilter

```
1 <?php
2
3 namespace Logfilter\Service\Filter;
4
5 use Logfilter\Config\FieldsToFilterConfig;
6 use Logfilter\Model\LogEntry;
7
8 /**
9  * redacts sensitive credential-related information from log
10  * entries
11  * covers: passwords, tokens, API keys, session identifiers,
12  * secrets, authorization headers
13  */
14 class CredentialsFilter extends AbstractFilter
15 {
16     private array $fieldLabelMap = [];
17
18     /**
19      * regex patterns precompiled once in the constructor
20      */
21     private array $compiledPatterns = [];
22
23     /**
24      * redaction labels corresponding to each compiled pattern
25      */
26     private array $compiledLabels = [];
27
28     /**
29      * creates the filter, builds the field-to-label lookup map
30      * and precompiles all regex patterns
31      */
32     public function __construct(array $config = [])
33     {
34         parent::__construct($config);
35         $this->buildFieldLabelMap();
36         $this->buildCompiledPatterns();
37     }
38
39     public function getName(): string
40     {
41         return 'CredentialsFilter';
42     }
43
44     /**
45      * maps each lowercase field name to its redaction label
46      * used for direct key-based replacement without regex
47      */
48     private function buildFieldLabelMap(): void
```

```

46 {
47     $fieldGroups = [
48         'PASSWORD' => FieldsToFilterConfig::
49             getPasswordFields(),
50         'TOKEN' => FieldsToFilterConfig::getTokenFields(),
51         'APIKEY' => FieldsToFilterConfig::getApiKeyFields()
52         ,
53         'SESSION' => FieldsToFilterConfig::getSessionFields
54         (),
55         'SECRET' => FieldsToFilterConfig::getSecretFields()
56         ,
57         'AUTHORIZATION' => FieldsToFilterConfig::
58             getAuthorizationFields(),
59     ];
60
61     foreach ($fieldGroups as $type => $fields)
62     {
63         $label = $this->getRedactionLabel($type);
64         foreach ($fields as $field)
65         {
66             $this->fieldLabelMap[strtolower($field)] =
67                 $label;
68         }
69     }
70
71     /**
72      * builds all regex patterns once at construction time so
73      * they are never rebuilt per log entry
74      */
75     private function buildCompiledPatterns(): void
76     {
77         $this->compilePasswordPatterns();
78         $this->compileTokenPatterns();
79         $this->compileApiKeyPatterns();
80         $this->compileSessionPatterns();
81         $this->compileSecretPatterns();
82         $this->compileAuthorizationPatterns();
83     }
84
85     private function compilePasswordPatterns(): void
86     {
87         $field = $this->buildFieldNameRegex(
88             FieldsToFilterConfig::getPasswordFields());
89         $label = $this->getRedactionLabel('PASSWORD');
90
91         //used for format like "password": "12345"
92         $this->compiledPatterns[] = '/'(?:' . $field . ')"\s*:\s*'["~"]+/i';
93     }
94 }

```

```

86         $this->compiledLabels[] = $label;
87
88         //used for format like password=12345 / password:12345
89         $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
          *[:]\s*["\s",;)&]+/i';
90         $this->compiledLabels[] = $label;
91     }
92
93     private function compileTokenPatterns(): void
94     {
95         $field = $this->buildFieldNameRegex(
96             FieldsToFilterConfig::getTokenFields());
97         $label = $this->getRedactionLabel('TOKEN');
98
99         //used for format like "token": "abc123"
100        $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
          *[:]\s*"[""]+"/i';
101        $this->compiledLabels[] = $label;
102
103        //used for format like token=abc123
104        $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
          *[:]\s*[A-Za-z0-9\-\._~+\/=]+/i';
105        $this->compiledLabels[] = $label;
106
107        // used for format like "share_token\":"abc123\"
108        $this->compiledPatterns[] = '/\\\\\\"(?:' . $field . ')\s
          *[:]\s*\\\\\\"[""]+"/i';
109        $this->compiledLabels[] = $label;
110    }
111
112     private function compileApiKeyPatterns(): void
113     {
114         $field = $this->buildFieldNameRegex(
115             FieldsToFilterConfig::getApiKeyFields());
116         $label = $this->getRedactionLabel('APIKEY');
117
118         //used for format like "apiKey": "sk-1234567890"
119        $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
          *[:]\s*"[""]+"/i';
120        $this->compiledLabels[] = $label;
121
122        //used for format like apikey=sk-1234567890 / apikey:sk
123        -1234567890
124        $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
          *[:]\s*["\s",;)&]+/i';
125        $this->compiledLabels[] = $label;
126    }
127
128     private function compileSessionPatterns(): void

```

```

126 {
127     $field = $this->buildFieldNameRegex(
128         FieldsToFilterConfig::getSessionFields());
129     $label = $this->getRedactionLabel('SESSION');
130
131     //used for format like "session": "abc123def"
132     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
133         *[:=]\s*[a-z0-9-]+/i';
134     $this->compiledLabels[] = $label;
135
136     //used for format like session=abc123def
137     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
138         *[:=]\s*[a-z0-9-]+/i';
139     $this->compiledLabels[] = $label;
140 }
141
142 private function compileSecretPatterns(): void
143 {
144     $field = $this->buildFieldNameRegex(
145         FieldsToFilterConfig::getSecretFields());
146     $label = $this->getRedactionLabel('SECRET');
147
148     //used for format like "secret": "mysecret123"
149     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
150         *[:=]\s*["^"]+/i';
151     $this->compiledLabels[] = $label;
152
153     //used for format like secret=mysecret123 / secret:
154         mysecret123
155     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
156         *[:=]\s*["^\\s",;)&]+/i';
157     $this->compiledLabels[] = $label;
158 }
159
160 private function compileAuthorizationPatterns(): void
161 {
162     $field = $this->buildFieldNameRegex(
163         FieldsToFilterConfig::getAuthorizationFields());
164     $label = $this->getRedactionLabel('AUTHORIZATION');
165
166     //used for format like "authorization": "Bearer abc123"
167     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
168         *[:=]\s*["^"]+/i';
169     $this->compiledLabels[] = $label;
170
171     //used for format like authorization=Bearer abc123
172     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
173         *[:=]\s*["^\\s",;)&]+(?:\s+["^\\s",;)&]+)?/i';
174     $this->compiledLabels[] = $label;

```

```

165     }
166
167     private function getRedactionLabel(string $type): string
168     {
169         return match ($type)
170         {
171             'PASSWORD' => '[PASSWORD REDACTED]',
172             'TOKEN' => '[TOKEN REDACTED]',
173             'APIKEY' => '[API KEY REDACTED]',
174             'SESSION' => '[SESSION INFORMATION REDACTED]',
175             'SECRET' => '[SECRET REDACTED]',
176             'AUTHORIZATION' => '[AUTHORIZATION REDACTED]',
177         };
178     }
179
180     /**
181     * processes a log entry and applies credential redaction.
182     *
183     * If the filter is disabled, the log entry is returned
184     * unchanged.
185     * Otherwise, both field-based and pattern-based redaction
186     * strategies
187     * are applied recursively to the log data.
188     *
189     * @param LogEntry $logEntry the log entry to process.
190     *
191     * @return LogEntry the filtered log entry.
192     */
193     public function process(LogEntry $logEntry): LogEntry
194     {
195         if (!$this->isEnabled())
196         {
197             return $logEntry;
198         }
199
200         $redacted = $this->redactData
201         (
202             $logEntry->getData(),
203             $this->fieldLabelMap,
204             fn(string $value) => $this->
205                 redactCredentialPatterns($value)
206         );
207
208         $logEntry->setData($redacted);
209         return $logEntry;
210     }
211
212     /**

```

```

210     * applies all precompiled patterns to a plain string value
        in a single preg_replace() call
211     */
212     private function redactCredentialPatterns(string $value):
        string
213     {
214         if (!str_contains($value, '=') && !str_contains($value,
            ':'))
215         {
216             return $value;
217         }
218
219         return preg_replace($this->compiledPatterns, $this->
            compiledLabels, $value) ?? $value;
220     }
221 }

```

A.11 InfrastructureFilter

InfrastructureFilter

```

1  <?php
2
3  namespace Logfilter\Service\Filter;
4
5  use Logfilter\Config\FieldsToFilterConfig;
6  use Logfilter\Model\LogEntry;
7
8  /**
9   * redacts sensitive infrastructure-related information from
        log entries
10   * covers: database credentials, hostnames, ports, types and
        names
11   */
12  class InfrastructureFilter extends AbstractFilter
13  {
14      private array $fieldLabelMap = [];
15
16      /**
17       * regex patterns precompiled once in the constructor
18       */
19      private array $compiledPatterns = [];
20
21      /**
22       * redaction labels corresponding to each compiled pattern
23       */
24      private array $compiledLabels = [];
25

```

```

26  /**
27   * creates the filter, builds the field-to-label lookup map
    and precompiles all regex patterns
28   */
29  public function __construct(array $config = [])
30  {
31      parent::__construct($config);
32      $this->buildFieldLabelMap();
33      $this->buildCompiledPatterns();
34  }
35
36  public function getName(): string
37  {
38      return 'InfrastructureFilter';
39  }
40
41  /**
42   * maps each lowercase field name to its redaction label
43   * used for direct key-based replacement without regex
44   */
45  private function buildFieldLabelMap(): void
46  {
47      $fieldGroups =
48      [
49          'DATABASE' => FieldsToFilterConfig::
            getDatabaseFields(),
50      ];
51
52      foreach ($fieldGroups as $type => $fields)
53      {
54          $label = $this->getRedactionLabel($type);
55          foreach ($fields as $field)
56          {
57              $this->fieldLabelMap[strtolower($field)] =
                $label;
58          }
59      }
60  }
61
62  /**
63   * builds all regex patterns once at construction time so
    they are never rebuilt per log entry
64   */
65  private function buildCompiledPatterns(): void
66  {
67      $this->compileDatabasePatterns();
68  }
69
70  private function compileDatabasePatterns(): void

```

```

71 {
72     $field = $this->buildFieldNameRegex(
73         FieldsToFilterConfig::getDatabaseFields());
74     $label = $this->getRedactionLabel('DATABASE');
75
76     //used for format like "dbPassword": "secretPassword123
77     " / "db_host": "localhost"
78     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
79         *[:]\s*"^"/i';
80     $this->compiledLabels[] = $label;
81
82     //used for format like dbPassword=secretPassword123 /
83     dbPassword:secretPassword123
84     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
85         *[:]\s*"^[\s",;)&]+/i';
86     $this->compiledLabels[] = $label;
87 }
88
89 private function getRedactionLabel(string $type): string
90 {
91     return match ($type)
92     {
93         'DATABASE' => '[DATABASE INFORMATION REDACTED]',
94     };
95 }
96
97 /**
98  * processes a log entry and applies infrastructure
99  * redaction.
100  *
101  * if the filter is disabled, the log entry is returned
102  * unchanged.
103  * otherwise, both field-based and pattern-based redaction
104  * strategies
105  * are applied recursively to the log data.
106  *
107  * @param LogEntry $logEntry the log entry to process.
108  *
109  * @return LogEntry the filtered log entry.
110  */
111 public function process(LogEntry $logEntry): LogEntry
112 {
113     if (!$this->isEnabled())
114     {
115         return $logEntry;
116     }
117
118     $redacted = $this->redactData
119     (

```

```

112         $logEntry->getData(),
113         $this->fieldLabelMap,
114         fn(string $value) => $this->
            redactInfrastructurePatterns($value)
115     );
116
117     $logEntry->setData($redacted);
118     return $logEntry;
119 }
120
121 /**
122  * applies all precompiled patterns to a plain string value
123  * in a single preg_replace() call
124  */
125 private function redactInfrastructurePatterns(string $value
126 ): string
127 {
128     if (!str_contains($value, '=') && !str_contains($value,
129         ':'))
130     {
131         return $value;
132     }
133
134     return preg_replace($this->compiledPatterns, $this->
        compiledLabels, $value) ?? $value;
135 }
136 }

```

A.12 NetworkFilter

NetworkFilter

```

1 <?php
2
3 namespace Logfilter\Service\Filter;
4
5 use Logfilter\Config\FieldsToFilterConfig;
6 use Logfilter\Model\LogEntry;
7
8 /**
9  * redacts sensitive network-related information from log
10  * entries
11  * covers: IPv4 addresses, IPv6 addresses, MAC addresses,
12  * hostnames
13  */
14 class NetworkFilter extends AbstractFilter
15 {
16     private array $fieldLabelMap = [];

```

```

15
16     /**
17      * regex patterns precompiled once in the constructor
18      */
19     private array $compiledPatterns = [];
20
21     /**
22      * redaction labels corresponding to each compiled pattern
23      */
24     private array $compiledLabels = [];
25
26     /**
27      * creates the filter, builds the field-to-label lookup map
28      * and precompiles all regex patterns
29      */
30     public function __construct(array $config = [])
31     {
32         parent::__construct($config);
33         $this->buildFieldLabelMap();
34         $this->buildCompiledPatterns();
35     }
36
37     public function getName(): string
38     {
39         return 'NetworkFilter';
40     }
41
42     /**
43      * maps each lowercase field name to its redaction label
44      * used for direct key-based replacement without regex
45      */
46     private function buildFieldLabelMap(): void
47     {
48         $fieldGroups = [
49             'IP' => FieldsToFilterConfig::getIpAddressFields(),
50             'MAC' => FieldsToFilterConfig::getMacFields(),
51             'HOST' => FieldsToFilterConfig::getHostFields(),
52         ];
53
54         foreach ($fieldGroups as $type => $fields)
55         {
56             $label = $this->getRedactionLabel($type);
57             foreach ($fields as $field)
58             {
59                 $this->fieldLabelMap[strtolower($field)] =
60                     $label;
61             }
62         }
63     }

```

```

62
63 /**
64  * builds all regex patterns once at construction time so
65  * they dont have to rebuilt per log entry
66  * IPv6 patterns are compiled before IPv4 to avoid partial
67  * collisions
68  */
69 private function buildCompiledPatterns(): void
70 {
71     $this->compileIpv6Patterns();
72     $this->compileIpv4Patterns();
73     $this->compileMacPatterns();
74     $this->compileHostnamePatterns();
75 }
76
77 private function compileIpv6Patterns(): void
78 {
79     $field = $this->buildFieldNameRegex(
80         FieldsToFilterConfig::getIpAddressFields());
81     $label = $this->getRedactionLabel('IP');
82
83     //used for format like "ipaddress": "2001:0db8:85a3
84     :0000:0000:8a2e:0370:7334"
85     $this->compiledPatterns[] = '/["\'](?:' . $field . '
86     ["\']\s*[:=]\s*["\']((?:[a-f0-9]{0,4}:){2,7}[a-f0
87     -9]{0,4}(?:%[0-9a-z]+)?)["\']/i';
88     $this->compiledLabels[] = $label;
89
90     //used for format like ipaddress=2001:0db8:85a3
91     :0000:0000:8a2e:0370:7334
92     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
93     *[:=]?[a-f0-9]{0,4}:){2,7}[a-f0
94     -9]{0,4}(?:%[0-9a-z]+)?(?:=$|[\s"\';:,\\])/i';
95     $this->compiledLabels[] = $label;
96 }
97
98 private function compileIpv4Patterns(): void
99 {
100     $field = $this->buildFieldNameRegex(
101         FieldsToFilterConfig::getIpAddressFields());
102     $label = $this->getRedactionLabel('IP');
103
104     //used for format like "ipaddress": "127.0.0.1"
105     $this->compiledPatterns[] = '/["\'](?:' . $field . '
106     ["\']\s*[:=]\s*["\']((?:25[0-5]|2[0-4]\d|1?\d
107     {1,2})\.){3}(?:25[0-5]|2[0-4]\d|1?\d{1,2}))["\']/i';
108     $this->compiledLabels[] = $label;
109
110     //used for format like ipaddress=127.0.0.1

```

```

99     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
    *[:=]\s*((?:25[0-5]|2[0-4]\d|1?\d{1,2})\.)
    {3}(?:25[0-5]|2[0-4]\d|1?\d{1,2}))\b/i';
100     $this->compiledLabels[] = $label;
101 }
102
103 private function compileMacPatterns(): void
104 {
105     $field = $this->buildFieldNameRegex(
106         FieldsToFilterConfig::getMacFields());
107     $label = $this->getRedactionLabel('MAC');
108
109     //used for format like "mac": "00:1A:2B:3C:4D:5E" / "
110     mac": "00-1A-2B-3C-4D-5E"
111     $this->compiledPatterns[] = '/"(?:' . $field . ')"\s
112     *[:=]\s*(([0-9a-f]{2}[-:]){5}[0-9a-f]{2})"/i';
113     $this->compiledLabels[] = $label;
114
115     //used for format like mac=00:1A:2B:3C:4D:5E / mac=00-1
116     A-2B-3C-4D-5E
117     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
118     *[:=]\s*(([0-9a-f]{2}[-:]){5}[0-9a-f]{2})/i';
119     $this->compiledLabels[] = $label;
120 }
121
122 private function compileHostnamePatterns(): void
123 {
124     $field = $this->buildFieldNameRegex(
125         FieldsToFilterConfig::getHostFields());
126     $label = $this->getRedactionLabel('HOST');
127
128     //used for format like "host": "example.com"
129     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
130     *[:=]\s*\\\\\\?"(?:[a-z0-9-]+\\.)+[a-z]{2,}\\\\\\?"?/i';
131     $this->compiledLabels[] = $label;
132
133     //used for format like host=example.com
134     $this->compiledPatterns[] = '/\b(?:' . $field . ')\s
135     *[:=]\s*\\\\\\?"(?:[a-z0-9-]+\\.)+[a-z]{2,}\\\\\\?"?/i';
136     $this->compiledLabels[] = $label;
137 }
138
139 private function getRedactionLabel(string $type): string
140 {
141     return match ($type)
142     {
143         'IP' => '[IP ADDRESS REDACTED]',
144         'MAC' => '[MAC ADDRESS REDACTED]',
145         'HOST' => '[HOSTNAME REDACTED]',

```

```

138         };
139     }
140
141     /**
142     * processes a log entry and applies network redaction.
143     *
144     * if the filter is disabled, the log entry is returned
145     * unchanged.
146     * otherwise, both field-based and pattern-based redaction
147     * strategies
148     * are applied recursively to the log data.
149     *
150     * @param LogEntry $logEntry the log entry to process.
151     *
152     * @return LogEntry the filtered log entry.
153     */
154     public function process(LogEntry $logEntry): LogEntry
155     {
156         if (!$this->isEnabled())
157         {
158             return $logEntry;
159         }
160
161         $redacted = $this->redactData
162         (
163             $logEntry->getData(),
164             $this->fieldLabelMap,
165             fn(string $value) => $this->redactNetworkPatterns(
166                 $value)
167         );
168
169         $logEntry->setData($redacted);
170         return $logEntry;
171     }
172
173     /**
174     * applies all precompiled patterns to a plain string value
175     * in a single preg_replace() call
176     */
177     private function redactNetworkPatterns(string $value):
178     string
179     {
180         if (!str_contains($value, '.') && !str_contains($value,
181             ':')) && !str_contains($value, '-'))
182         {
183             return $value;
184         }
185     }

```

```

180         return preg_replace($this->compiledPatterns, $this->
            compiledLabels, $value) ?? $value;
181     }
182 }

```

A.13 LogEntryTest

LogEntryTest

```

1  <?php
2
3  namespace tests\Unit;
4
5  use PHPUnit\Framework\TestCase;
6  use Logfilter\Model\LogEntry;
7
8  class LogEntryTest extends TestCase
9  {
10
11     public function testConstructorAndGetters(): void
12     {
13         $id = 'test-123';
14         $data = ['test' => 'test data'];
15
16         $logEntry = new LogEntry($id, $data);
17
18         $this->assertEquals($id, $logEntry->getId());
19         $this->assertEquals($data, $logEntry->getData());
20     }
21
22     public function testGetData(): void
23     {
24         $data = ['test' => 'test data'];
25         $logEntry = new LogEntry('id', $data);
26
27         $this->assertEquals($data, $logEntry->getData());
28     }
29
30     public function testSetData(): void
31     {
32         $logEntry = new LogEntry('1', ['old' => 'data']);
33         $newData = ['new' => 'data'];
34
35         $logEntry->setData($newData);
36
37         $this->assertEquals($newData, $logEntry->getData());
38     }
39 }

```

```

40 public function testGetField(): void
41 {
42     $logEntry = new LogEntry('1', ['key' => 'value', '
43         nested' => ['inner' => 'data']]');
44
45     $this->assertEquals('value', $logEntry->getField('key')
46         );
47     $this->assertEquals(['inner' => 'data'], $logEntry->
48         getField('nested'));
49     $this->assertNull($logEntry->getField('nonexistent'));
50     $this->assertEquals('default', $logEntry->getField('
51         nonexistent', 'default'));
52 }
53
54 public function testSetField(): void
55 {
56     $logEntry = new LogEntry('1', []);
57
58     $logEntry->setField('newKey', 'newValue');
59
60     $this->assertEquals('newValue', $logEntry->getField('
61         newKey'));
62 }
63
64 public function testHasField(): void
65 {
66     $logEntry = new LogEntry('1', ['exists' => 'value']);
67
68     $this->assertTrue($logEntry->hasField('exists'));
69     $this->assertFalse($logEntry->hasField('notExists'));
70 }
71
72 public function testToArray(): void
73 {
74     $id = 'test-123';
75     $data = ['message' => 'Test'];
76     $logEntry = new LogEntry($id, $data);
77
78     $array = $logEntry->toArray();
79
80     $this->assertIsArray($array);
81     $this->assertEquals($id, $array['id']);
82     $this->assertEquals($data, $array['data']);
83 }
84 }

```

A.14 FilterPipelineTest

FilterPipelineTest

```
1 <?php
2
3 namespace tests\Unit;
4
5 use PHPUnit\Framework\TestCase;
6 use Logfilter\Service\FilterPipeline;
7 use Logfilter\Service\Filter\CredentialsFilter;
8 use Logfilter\Service\Filter\NetworkFilter;
9 use Logfilter\Service\Filter\InfrastructureFilter;
10 use Logfilter\Model\LogEntry;
11
12 class FilterPipelineTest extends TestCase
13 {
14     private FilterPipeline $filterPipeline;
15
16     protected function setUp(): void
17     {
18         $this->filterPipeline = new FilterPipeline();
19         $this->filterPipeline->addFilter(new CredentialsFilter
20             ());
21         $this->filterPipeline->addFilter(new NetworkFilter());
22         $this->filterPipeline->addFilter(new
23             InfrastructureFilter());
24     }
25
26     private function printFilterStats(): void
27     {
28         echo "FILTER STATISTICS:\n";
29         foreach ($this->filterPipeline->getStatistics() as
30             $filterName => $stats)
31         {
32             $status = $stats['enabled'] ? 'ENABLED' : 'DISABLED
33                 ';
34             echo sprintf("%-30s %s (Processed: %d)\n",
35                 $filterName, $status, $stats['processed']);
36         }
37     }
38
39     public function testLogWithCredentialsInPipeline(): void
40     {
41         echo str_repeat("=", 40) . PHP_EOL;
42         echo "\nRESULTS FOR: testLogWithCredentialsInPipeline\n"
43             . "\n";
44         $originalData =
45             [
46                 'timestamp' => '2026-02-19T06:00:00Z',
47                 'action' => 'user_login',
48                 'user' => 'boris',
```

```

43         'password' => 'password123',
44         'ip' => '192.168.1.100',
45         'session' => 'sess_abc123def456'
46     ];
47
48     echo "\nORIGINAL LOG:\n";
49     echo json_encode($originalData, JSON_PRETTY_PRINT) . "\n";
50
51     $logEntry = new LogEntry('1', $originalData);
52
53     echo "\nPROCESSING THROUGH PIPELINE\n";
54     $filtered = $this->filterPipeline->process($logEntry);
55
56     echo "\nFILTERED LOG:\n";
57     echo json_encode($filtered->getData(),
58         JSON_PRETTY_PRINT) . "\n";
59
60     $this->printFilterStats();
61
62     $data = $filtered->getData();
63     $this->assertEquals('boris', $data['user']);
64     $this->assertEquals('[PASSWORD REDACTED]', $data['password']);
65     $this->assertEquals('[IP ADDRESS REDACTED]', $data['ip']);
66     $this->assertEquals('[SESSION INFORMATION REDACTED]', $data['session']);
67 }
68
69 public function testLogWithDatabaseInformationInPipeline(): void
70 {
71     echo str_repeat("=", 40) . PHP_EOL;
72     echo "\nRESULTS FOR:
73         testLogWithDatabaseInformationInPipeline\n";
74     $originalData =
75     [
76         'timestamp' => '2026-02-19T06:00:00Z',
77         'action' => 'database_action',
78         'db_user' => 'borisDbAdmin',
79         'db_password' => 'password123DB',
80         'dbHost' => '10.0.0.0',
81         'dbPort' => '1234',
82         'status' => 'success'
83     ];
84
85     echo "\nORIGINAL LOG:\n";

```

```

84         echo json_encode($originalData, JSON_PRETTY_PRINT) . "\n";
85
86         $logEntry = new LogEntry('1', $originalData);
87
88         echo "\nPROCESSING THROUGH PIPELINE\n";
89
90         $filtered = $this->filterPipeline->process($logEntry);
91
92         echo "\nFILTERED LOG:\n";
93         echo json_encode($filtered->getData(),
94             JSON_PRETTY_PRINT) . "\n";
95
96         $this->printFilterStats();
97
98         $data = $filtered->getData();
99         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
100             $data['db_user']);
101         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
102             $data['db_password']);
103         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
104             $data['db_host']);
105         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
106             $data['db_port']);
107         $this->assertEquals('success', $data['status']);
108         $this->assertEquals('database_action', $data['action'])
109         ;
110     }
111
112     public function testLogWithNetworkInformationInPipeline():
113         void
114     {
115         echo str_repeat("=", 40) . PHP_EOL;
116         echo "\nRESULTS FOR:
117             testLogWithNetworkInformationInPipeline\n";
118         $originalData =
119             [
120                 'timestamp' => '2026-02-19T06:00:00Z',
121                 'action' => 'access',
122                 'client' =>
123                     [
124                         'user' => 'boris',
125                         'ip_address' => '192.168.1.100',
126                         'port' => 8080
127                     ],
128                 'server' =>
129                     [
130                         'ip' => '10.0.0.1',
131                         'hostname' => 'server.example.com'
132                     ]
133             ]
134     }

```

```

124         ]
125     ];
126
127     echo "\nORIGINAL LOG:\n";
128     echo json_encode($originalData, JSON_PRETTY_PRINT) . "\n";
129
130     $logEntry = new LogEntry('1', $originalData);
131
132     echo "\nPROCESSING THROUGH PIPELINE\n";
133
134     $filtered = $this->filterPipeline->process($logEntry);
135
136     echo "\nFILTERED LOG:\n";
137     echo json_encode($filtered->getData(),
138         JSON_PRETTY_PRINT) . "\n";
139
140     $this->printFilterStats();
141
142     $data = $filtered->getData();
143     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
144         client']['ip_address']);
145     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
146         server']['ip']);
147     $this->assertEquals('[HOSTNAME REDACTED]', $data['
148         server']['hostname']);
149 }
150
151 public function testProcessingMultipleLogs(): void
152 {
153     echo str_repeat("=", 40) . PHP_EOL;
154     echo "\nRESULTS FOR: testProcessingMultipleLogs\n";
155     $logs =
156     [
157         ['id' => '1', 'user' => 'boris', 'password' =>
158             'password123', 'ip' => '192.168.1.10'],
159         ['id' => '2', 'user' => 'sirob', 'password' =>
160             'password321', 'ip' => '192.168.1.20'],
161         ['id' => '3', 'user' => 'sibor', 'password' =>
162             'pass123word', 'ip' => '192.168.1.30']
163     ];
164
165     foreach ($logs as $originalData) {
166         $id = $originalData['id'];
167
168         echo json_encode($originalData, JSON_PRETTY_PRINT)
169             . "\n";
170
171         $logEntry = new LogEntry($id, $originalData);

```

```

164         $filtered = $this->filterPipeline->process(
165             $logEntry);
166         $data = $filtered->getData();
167         echo json_encode($data, JSON_PRETTY_PRINT) . "\n";
168
169         $this->assertEquals('[PASSWORD REDACTED]', $data['
            password']);
170         $this->assertEquals('[IP ADDRESS REDACTED]', $data
            ['ip']);
171     }
172 }
173
174 public function testCredentialsInString(): void
175 {
176     $originalData =
177     [
178         'timestamp' => '2026-02-19T06:00:00Z',
179         'message'    => 'Login attempt failed password=
            hunter2 from user boris',
180     ];
181
182     $logEntry = new LogEntry('1', $originalData);
183     $filtered = $this->filterPipeline->process($logEntry);
184     $data = $filtered->getData();
185
186     $this->assertStringContainsString('[PASSWORD REDACTED
        ]', $data['message']);
187     $this->assertStringNotContainsString('hunter2', $data['
        message']);
188     $this->assertStringContainsString('Login attempt failed
        ', $data['message']);
189     $this->assertStringContainsString('from user boris',
        $data['message']);
190 }
191 }

```

A.15 CredentialsFilterTest

CredentialsFilterTest

```

1 <?php
2
3 namespace tests\Unit;
4
5 use PHPUnit\Framework\TestCase;
6 use Logfilter\Service\Filter\CredentialsFilter;
7 use Logfilter\Model\LogEntry;

```

```

8
9 class CredentialsFilterTest extends TestCase
10 {
11     private CredentialsFilter $filter;
12
13     /**
14      * initializes an credentialsFilter instance
15      */
16     protected function setUp(): void
17     {
18         $this->filter = new CredentialsFilter();
19     }
20
21     // -----
22     // General Tests
23     // -----
24
25     public function testGetName(): void
26     {
27         $this->assertEquals('CredentialsFilter', $this->filter
28             ->getName());
29     }
30
31     /**
32      * test to ensure, that the filter is enabled by default
33      */
34     public function testIsEnabledByDefault(): void
35     {
36         $this->assertTrue($this->filter->isEnabled());
37     }
38
39     /**
40      * test to ensure, that the filter can be disabled
41      */
42     public function testCanBeDisabled(): void
43     {
44         $filter = new CredentialsFilter(['enabled' => false]);
45         $this->assertFalse($filter->isEnabled());
46     }
47
48     /**
49      * test to ensure, that the filter does not behave
50      * unexpectedly when disabled
51      */
52     public function testDoesNotProcessWhenDisabled(): void
53     {
54         $filter = new CredentialsFilter(['enabled' => false]);
55         $data =
56             [

```

```

55         'password' => 'password123',
56         'token' => 'abc123'
57     ];
58
59     $logEntry = new LogEntry('1', $data);
60     $filtered = $filter->process($logEntry);
61
62     $this->assertEquals($data, $filtered->getData());
63 }
64
65 public function testHandlesEmptyData(): void
66 {
67     $logEntry = new LogEntry('1', []);
68     $filtered = $this->filter->process($logEntry);
69
70     $this->assertEquals([], $filtered->getData());
71 }
72
73 /**
74  * test to ensure, that non-sensitive fields are preserved
75  */
76 public function testPreservesNonSensitiveData(): void
77 {
78     $logEntry = new LogEntry('1',
79     [
80         'username' => 'boris_boris',
81         'email' => 'boris@example.com',
82         'timestamp' => '2026-01-01T12:00:00Z',
83         'password' => 'password123'
84     ]);
85
86     $filtered = $this->filter->process($logEntry);
87     $data = $filtered->getData();
88
89     $this->assertEquals('boris_boris', $data['username']);
90     $this->assertEquals('boris@example.com', $data['email
91         ']);
92     $this->assertEquals('2026-01-01T12:00:00Z', $data['
93         timestamp']);
94     $this->assertEquals('[PASSWORD REDACTED]', $data['
95         password']);
96 }
97
98 private function printRedactionResult(array $before, array
99     $after): void
100 {
101     echo 'BEFORE: ' . json_encode($before) . PHP_EOL;
102     echo 'AFTER: ' . json_encode($after) . PHP_EOL;
103 }

```

```

100
101 // -----
102 // Password Tests
103 // -----
104
105 public function testRedactsCaseInsensitiveFieldNames():
106     void
107 {
108     $logEntry = new LogEntry('1',
109         [
110             'PASSWORD' => 'password123',
111             'PaSswOrd' => 'another_Password',
112             'TOKEN' => 'token123'
113         ]
114     );
115
116     $filtered = $this->filter->process($logEntry);
117     $data = $filtered->getData();
118
119     $this->assertEquals('[PASSWORD REDACTED]', $data['
120         PASSWORD']);
121     $this->assertEquals('[PASSWORD REDACTED]', $data['
122         PaSswOrd']);
123     $this->assertEquals('[TOKEN REDACTED]', $data['TOKEN']);
124
125 }
126
127 public function testRedactsPasswordFields(): void
128 {
129     $logEntry = new LogEntry('1',
130         [
131             'password' => 'password123',
132             'user' => 'boris',
133             'pwd' => 'another_Password'
134         ]
135     );
136
137     $before =
138     [
139         'password' => 'password123',
140         'user' => 'boris',
141         'pwd' => 'another_Password'
142     ];
143
144     $filtered = $this->filter->process($logEntry);
145     $data = $filtered->getData();
146     $this->printRedactionResult($before, $data);
147
148     $this->assertEquals('[PASSWORD REDACTED]', $data['
149         password']);

```

```

143         $this->assertEquals('[PASSWORD REDACTED]', $data['pwd
144             ']);
145     $this->assertEquals('boris', $data['user']);
146 }
147
148 public function
149     testRedactsAllCredentialPasswordFieldVariations(): void
150 {
151     $before =
152         [
153             'password' => 'password123',
154             'pwd' => 'password123',
155             'passwd' => 'password123',
156             'userPassword' => 'password123',
157             'smtPassword' => 'password123',
158             'smtp_password' => 'password123',
159             'mailPassword' => 'password123',
160             'mail_Password' => 'password123',
161             'passwordSalt' => 'password123',
162             'password_salt' => 'password123'
163         ];
164
165     $logEntry = new LogEntry('1', $before);
166     $after = $this->filter->process($logEntry)->getData();
167
168     $this->printRedactionResult($before, $after);
169
170     foreach ($before as $field => $value)
171     {
172         $this->assertEquals('[PASSWORD REDACTED]', $after[
173             $field]);
174     }
175 }
176
177 public function testRedactsNestedPasswordFieldVariations(): void
178 {
179     $logEntry = new LogEntry('1',
180         [
181             'user' =>
182                 [
183                     'username' => 'boris',
184                     'password' => 'password123'
185                 ]
186         ]
187     );
188
189     $filtered = $this->filter->process($logEntry);
190     $data = $filtered->getData();

```

```

188         $this->assertEquals('boris', $data['user']['username'])
189         ;
190         $this->assertEquals('[PASSWORD REDACTED]', $data['user
191         ']['password']);
192     }
193
194     public function testRedactsPasswordPatternsInStrings():
195     void
196     {
197         $logEntry = new LogEntry('1',
198         [
199             'message' => 'User login with password=
200             password123 failed'
201         ]);
202
203         $filtered = $this->filter->process($logEntry);
204         $data = $filtered->getData();
205
206         $this->assertStringContainsString('[PASSWORD REDACTED
207         ]', $data['message']);
208         $this->assertStringNotContainsString('password123',
209         $data['message']);
210     }
211
212     public function testRedactsJsonPasswordPatterns(): void
213     {
214         $logEntry = new LogEntry('1',
215         [
216             'log' => '{"username":"boris","password":"
217             password123","email":"boris@example.com"}'
218         ]);
219
220         $filtered = $this->filter->process($logEntry);
221         $data = $filtered->getData();
222
223         $this->assertStringContainsString('[PASSWORD REDACTED
224         ]', $data['log']);
225         $this->assertStringNotContainsString('password123',
226         $data['log']);
227         $this->assertStringContainsString('boris@example.com',
228         $data['log']);
229     }
230
231     // -----
232     // Token Tests
233     // -----
234
235     public function testRedactsTokenFields(): void
236     {

```

```

227     $logEntry = new LogEntry('1',
228         [
229             'token' => 'abc123token',
230             'authToken' => '321cba',
231             'bearerToken' => 'bearer123',
232         ]
233     );
234
235     $filtered = $this->filter->process($logEntry);
236     $data = $filtered->getData();
237
238     $this->assertEquals('[TOKEN REDACTED]', $data['token'])
239     ;
240     $this->assertEquals('[TOKEN REDACTED]', $data['
241         authToken']);
242     $this->assertEquals('[TOKEN REDACTED]', $data['
243         bearerToken']);
244 }
245
246 public function
247     testRedactsAllCredentialTokenFieldVariations(): void
248 {
249     $before =
250     [
251         'token' => 'abc123token',
252         'authToken' => 'abc123token',
253         'auth_token' => 'abc123token',
254         'accessToken' => 'abc123token',
255         'access_token' => 'abc123token',
256         'bearer' => 'abc123token',
257         'bearerToken' => 'abc123token',
258     ];
259
260     $logEntry = new LogEntry('1', $before);
261     $after = $this->filter->process($logEntry)->getData();
262
263     $this->printRedactionResult($before, $after);
264
265     foreach ($before as $field => $value)
266     {
267         $this->assertEquals('[TOKEN REDACTED]', $after[
268             $field]);
269     }
270 }
271
272 public function testRedactsNestedTokenFields(): void
273 {
274     $logEntry = new LogEntry('1',
275         [
276             'action' =>

```

```

271         [
272             'user' => 'boris',
273             'accessToken' => 'abc123token'
274         ]
275     ));
276
277     $filtered = $this->filter->process($logEntry);
278     $data = $filtered->getData();
279
280     $this->assertEquals('boris', $data['action']['user']);
281     $this->assertEquals('[TOKEN REDACTED]', $data['action']
282         ['accessToken']);
283 }
284
285 public function testRedactsTokenPatternsInStrings(): void
286 {
287     $logEntry = new LogEntry('1',
288         [
289             'message' => 'test message with bearer=
290                 abc123token'
291         ]
292     );
293
294     $filtered = $this->filter->process($logEntry);
295     $data = $filtered->getData();
296
297     $this->assertStringContainsString('[TOKEN REDACTED]',
298         $data['message']);
299     $this->assertStringNotContainsString('abc123token',
300         $data['message']);
301 }
302
303 public function testRedactsComplexTokenPatternsInStrings():
304     void
305 {
306     $before =
307         [
308             'message' => 'test message with \"share_token
309                 \":\"abc123\" included as a complex token
310                 pattern'
311         ]
312     ;
313
314     $logEntry = new LogEntry('1', $before);
315     $after = $this->filter->process($logEntry)->getData();
316
317     $this->printRedactionResult($before, $after);
318
319     $this->assertStringContainsString('[TOKEN REDACTED]',
320         $after['message']);

```

```

312         $this->assertStringNotContainsString('\\"share_token
           \":\\"abc123\\"', $after['message']);
313     }
314
315     public function testRedactsJsonTokenPatterns(): void
316     {
317         $logEntry = new LogEntry('1',
318             [
319                 'log' => '{"username":"boris","token":"
                        abc123token","email":"boris@example.com"}',
320             ]);
321
322         $filtered = $this->filter->process($logEntry);
323         $data = $filtered->getData();
324
325         $this->assertStringContainsString('[TOKEN REDACTED]',
            $data['log']);
326         $this->assertStringNotContainsString('abc123token',
            $data['log']);
327         $this->assertStringContainsString('boris@example.com',
            $data['log']);
328     }
329
330     // -----
331     // API Tests
332     // -----
333
334     public function testRedactsApiKeyFields(): void
335     {
336         $logEntry = new LogEntry('1',
337             [
338                 'apiKey' => 'sk-1234567890',
339                 'api_key' => 'ak_test_123'
340             ]);
341
342         $filtered = $this->filter->process($logEntry);
343         $data = $filtered->getData();
344
345         $this->assertEquals('[API KEY REDACTED]', $data['apiKey
            ']);
346         $this->assertEquals('[API KEY REDACTED]', $data['
            api_key']);
347     }
348
349     public function testRedactsAllCredentialAPIFieldVariations
        (): void
350     {
351         $before =
352         [

```

```

353         'apiKey' => 'sk-1234567890',
354         'api_key' => 'sk-1234567890',
355         'key' => 'sk-1234567890',
356     ];
357
358     $logEntry = new LogEntry('1', $before);
359     $after = $this->filter->process($logEntry)->getData();
360
361     $this->printRedactionResult($before, $after);
362
363     foreach ($before as $field => $value)
364     {
365         $this->assertEquals('[API KEY REDACTED]', $after[
366             $field]);
367     }
368
369     public function testRedactsNestedAPIFields(): void
370     {
371         $logEntry = new LogEntry('1',
372             [
373                 'action' =>
374                 [
375                     'user' => 'boris',
376                     'apiKey' => 'abc123token'
377                 ]
378             ]);
379
380         $filtered = $this->filter->process($logEntry);
381         $data = $filtered->getData();
382
383         $this->assertEquals('boris', $data['action']['user']);
384         $this->assertEquals('[API KEY REDACTED]', $data['action']
385             ['apiKey']);
386     }
387
388     public function testRedactsApiKeyPatternsInStrings(): void
389     {
390         $logEntry = new LogEntry('1',
391             [
392                 'message' => 'Request failed with apiKey=sk
393                     -1234567890 for user boris'
394             ]);
395
396         $filtered = $this->filter->process($logEntry);
397         $data = $filtered->getData();
398
399         $this->assertStringContainsString('[API KEY REDACTED]',
400             $data['message']);

```

```

398         $this->assertStringNotContainsString('sk-1234567890',
399             $data['message']);
400     }
401
402     public function testRedactJsonApiKeyPatterns(): void
403     {
404         $logEntry = new LogEntry('1',
405             [
406                 'log' => '{"username":"boris","apiKey":"sk-1234567890","email":"boris@example.com"}',
407             ]);
408
409         $filtered = $this->filter->process($logEntry);
410         $data = $filtered->getData();
411
412         $this->assertStringContainsString('[API KEY REDACTED]',
413             $data['log']);
414         $this->assertStringNotContainsString('sk-1234567890',
415             $data['log']);
416         $this->assertStringContainsString('boris@example.com',
417             $data['log']);
418     }
419
420     // -----
421     // Session Tests
422     // -----
423
424     public function
425     testRedactsAllCredentialSessionFieldVariations(): void
426     {
427         $before =
428             [
429                 'session' => 'abc123',
430                 'sessionId' => 'abc123',
431                 'session_id' => 'abc123',
432                 'PHPSESSID' => 'abc123',
433             ];
434
435         $logEntry = new LogEntry('1', $before);
436         $after = $this->filter->process($logEntry)->getData();
437
438         $this->printRedactionResult($before, $after);
439
440         foreach ($before as $field => $value)
441         {
442             $this->assertEquals('[SESSION INFORMATION REDACTED]', $after[$field]);
443         }
444     }

```

```

440
441 public function testRedactsSessionFields(): void
442 {
443     $logEntry = new LogEntry('1',
444         [
445             'session' => 'abc123',
446             'sessionId' => 'sessionid-abc123',
447             'PHPSESSID' => 'phpSess_123'
448         ]
449     );
450     $filtered = $this->filter->process($logEntry);
451     $data = $filtered->getData();
452
453     $this->assertEquals('[SESSION INFORMATION REDACTED]',
454         $data['session']);
455     $this->assertEquals('[SESSION INFORMATION REDACTED]',
456         $data['sessionId']);
457     $this->assertEquals('[SESSION INFORMATION REDACTED]',
458         $data['PHPSESSID']);
459 }
460
461 public function testRedactsNestedSessionsFields(): void
462 {
463     $logEntry = new LogEntry('1',
464         [
465             'action' =>
466             [
467                 'user' => 'boris',
468                 'PHPSESSID' => 'phpSess_123'
469             ]
470         ]
471     );
472     $filtered = $this->filter->process($logEntry);
473     $data = $filtered->getData();
474
475     $this->assertEquals('boris', $data['action']['user']);
476     $this->assertEquals('[SESSION INFORMATION REDACTED]',
477         $data['action']['PHPSESSID']);
478 }
479
480 public function testRedactSessionPatternsInStrings(): void
481 {
482     $logEntry = new LogEntry('1',
483         [
484             'message' => 'Request failed with session=
485                 abc123 for user boris with sessionId:
486                 sessionid-abc123'
487         ]
488     );

```

```

483         $filtered = $this->filter->process($logEntry);
484         $data = $filtered->getData();
485
486         $this->assertStringContainsString('[SESSION INFORMATION
487             REDACTED]', $data['message']);
488         $this->assertStringNotContainsString('abc123', $data['
489             message']);
490         $this->assertStringNotContainsString('sessionid-abc123
491             ', $data['message']);
492     }
493
494     public function testRedactsJsonSessionPatterns(): void
495     {
496         $logEntry = new LogEntry('1',
497             [
498                 'log' => '{"username":"boris","session":"abc123
499                     ","email":"boris@example.com"}'
500             ]
501         );
502
503         $filtered = $this->filter->process($logEntry);
504         $data = $filtered->getData();
505
506         $this->assertStringContainsString('[SESSION INFORMATION
507             REDACTED]', $data['log']);
508         $this->assertStringNotContainsString('abc123', $data['
509             log']);
510         $this->assertStringContainsString('boris@example',
511             $data['log']);
512     }
513
514     // -----
515     // Secret Tests
516     // -----
517
518     public function
519     testRedactsAllCredentialSecretFieldVariations(): void
520     {
521         $before =
522             [
523                 'secret' => 'mysecret123',
524                 'secretKey' => 'mysecret123',
525                 'secret_key' => 'mysecret123',
526                 'encryptionKey' => 'mysecret123',
527                 'encryption_key' => 'mysecret123',
528             ]
529         ;
530
531         $logEntry = new LogEntry('1', $before);
532         $after = $this->filter->process($logEntry)->getData();
533

```

```

524         $this->printRedactionResult($before, $after);
525
526         foreach ($before as $field => $value)
527         {
528             $this->assertEquals('[SECRET REDACTED]', $after[
529                 $field]);
530         }
531     }
532
533     public function testRedactsSecretFields(): void
534     {
535         $logEntry = new LogEntry('1',
536             [
537                 'secret' => 'mysecret123',
538                 'secret_key' => 'mysecret123',
539                 'encryptionKey' => 'mysecret123'
540             ]);
541
542         $filtered = $this->filter->process($logEntry);
543         $data = $filtered->getData();
544
545         $this->assertEquals('[SECRET REDACTED]', $data['secret
546             ']);
547         $this->assertEquals('[SECRET REDACTED]', $data['
548             secret_key']);
549         $this->assertEquals('[SECRET REDACTED]', $data['
550             encryptionKey']);
551     }
552
553     public function testRedactsNestedSecretFields(): void
554     {
555         $logEntry = new LogEntry('1',
556             [
557                 'action' =>
558                 [
559                     'user' => 'boris',
560                     'secret' => 'mysecret123'
561                 ]
562             ]);
563
564         $filtered = $this->filter->process($logEntry);
565         $data = $filtered->getData();
566
567         $this->assertEquals('boris', $data['action']['user']);
568         $this->assertEquals('[SECRET REDACTED]', $data['action
569             ']['secret']);
570     }
571
572     public function testRedactSecretPatternsInStrings(): void

```

```

568 {
569     $logEntry = new LogEntry('1',
570     [
571         'message' => 'user boris with "secret=
                    mysecret123"'
572     ]);
573
574     $filtered = $this->filter->process($logEntry);
575     $data = $filtered->getData();
576
577     $this->assertStringContainsString('[SECRET REDACTED]',
                    $data['message']);
578     $this->assertStringNotContainsString('mysecret123',
                    $data['message']);
579 }
580
581 public function testRedactsJsonSecretPatterns(): void
582 {
583     $logEntry = new LogEntry('1',
584     [
585         'log' => '{"username":"boris","secret":"
                    mysecret123","email":"boris@example.com"}'
586     ]);
587
588     $filtered = $this->filter->process($logEntry);
589     $data = $filtered->getData();
590
591     $this->assertStringContainsString('[SECRET REDACTED]',
                    $data['log']);
592     $this->assertStringNotContainsString('mysecret123',
                    $data['log']);
593     $this->assertStringContainsString('boris@example',
                    $data['log']);
594 }
595
596 // -----
597 // Authorization Tests
598 // -----
599
600 public function testRedactsAuthorizationField(): void
601 {
602     $logEntry = new LogEntry('1',
603     [
604         'authorization' => 'Bearer
                    abcdefghijklmnopqrstuvwxyz0123456789'
605     ]);
606
607     $filtered = $this->filter->process($logEntry);
608     $data = $filtered->getData();

```

```

609         $this->assertEquals('[AUTHORIZATION REDACTED]', $data['
610             authorization']);
611         $this->assertStringNotContainsString('
        abcdefghijklmnopqrstuvwxyz0123456789', $data['
        authorization']);
612     }
613
614     public function
        testRedactsAllCredentialAuthorizationFieldVariations():
        void
615     {
616         $before =
617             [
618                 'authorization' => '
        abcdefghijklmnopqrstuvwxyz0123456789',
619                 'auth' => 'abcdefghijklmnopqrstuvwxyz0123456789
        ',
620             ];
621
622         $logEntry = new LogEntry('1', $before);
623         $after = $this->filter->process($logEntry)->getData();
624
625         $this->printRedactionResult($before, $after);
626
627         foreach ($before as $field => $value)
628         {
629             $this->assertEquals('[AUTHORIZATION REDACTED]',
        $after[$field]);
630         }
631     }
632
633     public function testRedactsBearerTokensInStrings(): void
634     {
635         $logEntry = new LogEntry('1',
636             [
637                 'message' => 'Request received with
        Authorization: Bearer
        abcdefghijklmnopqrstuvwxyz0123456789',
638             ]);
639
640         $before =
641             [
642                 'message' => 'Request received with
        Authorization: Bearer
        abcdefghijklmnopqrstuvwxyz0123456789',
643             ];
644
645         $filtered = $this->filter->process($logEntry);

```

```

646         $data = $filtered->getData();
647         $this->printRedactionResult($before, $data);
648
649         $this->assertStringNotContainsString('
            abcdefghijklmnopqrstuvwxyz0123456789', $data['
            message']);
650     }
651
652     public function testRedactsJsonAuthorizationPatterns():
        void
653     {
654         $logEntry = new LogEntry('1',
655             [
656                 'log' => '{"username":"boris","authorization":"
                    abcdefghijklmnopqrstuvwxyz0123456789","email
                    ":"boris@example.com"}'
657             ]
658         );
659
660         $filtered = $this->filter->process($logEntry);
661         $data = $filtered->getData();
662
663         $this->assertStringContainsString('[AUTHORIZATION
            REDACTED]', $data['log']);
664         $this->assertStringNotContainsString('
            abcdefghijklmnopqrstuvwxyz0123456789', $data['log'])
            ;
665         $this->assertStringContainsString('boris@example',
            $data['log']);
666     }
}

```

A.16 InfrastructureFilterTest

InfrastructureFilterTest

```

1  <?php
2
3  namespace tests\Unit;
4
5  use PHPUnit\Framework\TestCase;
6  use Logfilter\Service\Filter\InfrastructureFilter;
7  use Logfilter\Model\LogEntry;
8
9  class InfrastructureFilterTest extends TestCase
10 {
11     private InfrastructureFilter $filter;
12
13     /**

```

```

14     * initializes an infrastructureFilter instance
15     */
16 protected function setUp(): void
17 {
18     $this->filter = new InfrastructureFilter();
19 }
20
21 private function printRedactionResult(array $before, array
    $after): void
22 {
23     echo 'BEFORE: ' . json_encode($before) . PHP_EOL;
24     echo 'AFTER: ' . json_encode($after) . PHP_EOL;
25 }
26
27 // -----
28 // General Tests
29 // -----
30
31 public function testGetName(): void
32 {
33     $this->assertEquals('InfrastructureFilter', $this->
        filter->getName());
34 }
35
36 /**
37  * test to ensure, that the filter is enabled by default
38  */
39 public function testIsEnabledByDefault(): void
40 {
41     $this->assertTrue($this->filter->isEnabled());
42 }
43
44 /**
45  * test to ensure, that the filter can be disabled
46  */
47 public function testCanBeDisabled(): void
48 {
49     $filter = new InfrastructureFilter(['enabled' => false
        ]);
50     $this->assertFalse($filter->isEnabled());
51 }
52
53 public function testHandlesEmptyData(): void
54 {
55     $logEntry = new LogEntry('1', []);
56
57     $filtered = $this->filter->process($logEntry);
58
59     $this->assertEquals([], $filtered->getData());

```

```

60     }
61
62     // -----
63     // DB Tests
64     // -----
65
66     public function testRedactsAllDatabaseFieldVariations():
67         void
68     {
69         $before =
70             [
71                 'db_host' => 'localhost',
72                 'db_user' => 'boris',
73                 'db_password' => 'password123',
74                 'db_name' => 'db',
75                 'db_port' => '1234',
76                 'database_host' => 'localhost',
77                 'database_user' => 'sirob',
78                 'database_password' => 'password321'
79             ];
80
81         $logEntry = new LogEntry('1', $before);
82         $after = $this->filter->process($logEntry)->getData();
83
84         $this->printRedactionResult($before, $after);
85
86         foreach ($before as $field => $value)
87         {
88             $this->assertEquals('[DATABASE INFORMATION REDACTED
89                 ]', $after[$field]);
90         }
91     }
92
93     public function testRedactsDatabaseFields(): void
94     {
95         $logEntry = new LogEntry('1',
96             [
97                 'dbhost' => '127.0.0.1',
98                 'dbuser' => 'boris',
99                 'db_password' => 'another_Password',
100                 'dbType' => 'mariaDB'
101             ]
102         );
103
104         $before =
105             [
106                 'dbhost' => '127.0.0.1',
107                 'dbuser' => 'boris',
108                 'db_password' => 'another_Password',
109                 'dbType' => 'mariaDB'

```

```

107         ];
108
109         $filtered = $this->filter->process($logEntry);
110         $data = $filtered->getData();
111         $this->printRedactionResult($before, $data);
112
113         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
114             $data['dbhost']);
115         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
116             $data['dbuser']);
117         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
118             $data['db_password']);
119         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
120             $data['dbType']);
121     }
122
123     public function testRedactsCaseInsensitiveFieldNames():
124         void
125     {
126         $logEntry = new LogEntry('1',
127             [
128                 'dbhost' => '127.0.0.1',
129                 'DBhost' => '127.0.0.1',
130                 'db_passWORd' => 'password123',
131                 'dbPassWord' => 'password123'
132             ]
133         );
134
135         $filtered = $this->filter->process($logEntry);
136         $data = $filtered->getData();
137
138         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
139             $data['dbhost']);
140         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
141             $data['DBhost']);
142         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
143             $data['db_passWORd']);
144         $this->assertEquals('[DATABASE INFORMATION REDACTED]',
145             $data['dbPassWord']);
146     }
147
148     public function testRedactsDatabasePatternsInStrings():
149         void
150     {
151         $logEntry = new LogEntry('1',
152             [
153                 'message' => 'Connection failed: db_host=
154                     localhost, db_user=boris, db_password:
155                     secret123'
156             ]
157         );

```

```

144         $filtered = $this->filter->process($logEntry);
145         $data = $filtered->getData();
146
147
148         $this->assertStringContainsString('[DATABASE
149             INFORMATION REDACTED]', $data['message']);
150         $this->assertStringNotContainsString('boris', $data['
151             message']);
152         $this->assertStringNotContainsString('localhost', $data
153             ['message']);
154         $this->assertStringNotContainsString('secret123', $data
155             ['message']);
156     }
157
158     public function testRedactsJsonDatabasePatterns(): void
159     {
160         $logEntry = new LogEntry('1',
161             [
162                 'log' => '{"dbuser":"boris","db_password": "
163                     password123","dbhost" = "127.0.0.1"}'
164             ]);
165
166         $filtered = $this->filter->process($logEntry);
167         $data = $filtered->getData();
168
169         $this->assertStringContainsString('[DATABASE
170             INFORMATION REDACTED]', $data['log']);
171         $this->assertStringNotContainsString('boris', $data['
172             log']);
173         $this->assertStringNotContainsString('password123',
174             $data['log']);
175         $this->assertStringNotContainsString('127.0.0.1', $data
176             ['log']);
177     }
178 }

```

A.17 NetworkFilterTest

NetworkFilterTest

```

1 <?php
2
3 namespace tests\Unit;
4
5 use PHPUnit\Framework\TestCase;
6 use Logfilter\Service\FILTER\NetworkFilter;
7 use Logfilter\Model\LogEntry;
8

```

```

9 class NetworkFilterTest extends TestCase
10 {
11     private NetworkFilter $filter;
12
13     /**
14      * initializes an networkFilter instance
15      */
16     protected function setUp(): void
17     {
18         $this->filter = new NetworkFilter();
19     }
20
21     private function printRedactionResult(array $before, array
22         $after): void
23     {
24         echo 'BEFORE: ' . json_encode($before) . PHP_EOL;
25         echo 'AFTER: ' . json_encode($after) . PHP_EOL;
26     }
27
28     // -----
29     // General Tests
30     // -----
31
32     public function testGetName(): void
33     {
34         $this->assertEquals('NetworkFilter', $this->filter->
35             getName());
36     }
37
38     /**
39      * test to ensure, that the filter is enabled by default
40      */
41     public function testIsEnabledByDefault(): void
42     {
43         $this->assertTrue($this->filter->isEnabled());
44     }
45
46     /**
47      * test to ensure, that the filter can be disabled
48      */
49     public function testCanBeDisabled(): void
50     {
51         $filter = new NetworkFilter(['enabled' => false]);
52         $this->assertFalse($filter->isEnabled());
53     }
54
55     public function testRedactsAllNetworkFieldVariations():
56         void
57     {

```

```

55     $logEntry = new LogEntry('1', [
56         'ip' => '192.168.1.1',
57         'ipAddress' => '10.0.0.1',
58         'ip_address' => '127.0.0.1',
59         'remoteAddress' => '192.168.1.1',
60         'remote_address' => '10.0.0.1',
61         'remoteAddr' => '127.0.0.1',
62         'remote_addr' => '192.168.1.1',
63         'upstreamAddr' => '10.0.0.1',
64         'upstream_addr' => '127.0.0.1',
65         'macAddress' => '00:1A:2B:3C:4D:5E',
66         'mac_address' => '00-1A-2B-3C-4D-5E',
67         'macAddr' => '00:1A:2B:3C:4D:5E',
68         'mac_addr' => '00-1A-2B-3C-4D-5E',
69         'mac' => '00:1A:2B:3C:4D:5E',
70         'host' => 'test.example.com',
71         'hostname' => 'test.example.com',
72         'host_name' => 'test.example.com',
73     ]);
74
75     $filtered = $this->filter->process($logEntry);
76     $data = $filtered->getData();
77
78     $ipFields = ['ip', 'ipAddress', 'ip_address', '
79         remoteAddress', 'remote_address', 'remoteAddr', '
80         remote_addr', 'upstreamAddr', 'upstream_addr'];
81     $macFields = ['macAddress', 'mac_address', 'macAddr',
82         'mac_addr', 'mac'];
83     $hostFields = ['host', 'hostname', 'host_name'];
84
85     foreach ($ipFields as $field) {
86         $this->assertEquals('[IP ADDRESS REDACTED]', $data[
87             $field]);
88     }
89
90     foreach ($macFields as $field) {
91         $this->assertEquals('[MAC ADDRESS REDACTED]', $data[
92             $field]);
93     }
94
95     foreach ($hostFields as $field) {
96         $this->assertEquals('[HOSTNAME REDACTED]', $data[
97             $field]);
98     }
99
100 }
101
102 // -----
103 // IP Tests
104 // -----

```

```

98
99 public function testRedactsCaseInsensitiveFieldNames():
    void
100 {
101     $logEntry = new LogEntry('1',
102         [
103             'ip_Address' => '127.0.0.1',
104             'ip_AddresS' => '23:123:123:123',
105             'mAc_AdDr' => '00:1A:2B:3C:4D:5E',
106         ]);
107
108     $filtered = $this->filter->process($logEntry);
109     $data = $filtered->getData();
110
111     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
        ip_Address']);
112     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
        ip_AddresS']);
113     $this->assertEquals('[MAC ADDRESS REDACTED]', $data['
        mAc_AdDr']);
114 }
115
116 public function testRedactsIP4Fields(): void
117 {
118     $logEntry = new LogEntry('1',
119         [
120             'ip' => '192.168.1.1',
121             'ipAddress' => '10.0.0.1',
122             'remote_addr' => '127.0.0.1',
123             'user' => 'boris'
124         ]);
125
126     $before =
127         [
128             'ip' => '192.168.1.1',
129             'ipAddress' => '10.0.0.1',
130             'remote_addr' => '127.0.0.1',
131             'user' => 'boris'
132         ];
133
134     $filtered = $this->filter->process($logEntry);
135     $data = $filtered->getData();
136     $this->printRedactionResult($before, $data);
137
138     $this->assertEquals('[IP ADDRESS REDACTED]', $data['ip
        ']);
139     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
        ipAddress']);

```

```

140         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
            remote_addr']);
141     }
142
143     public function testRedactsNestedIPv4Fields(): void
144     {
145         $logEntry = new LogEntry('1',
146             [
147                 'request' => [
148                     'client' => [
149                         'ip' => '192.168.1.1',
150                         'port' => 8080
151                     ],
152                     'server' => [
153                         'ip' => '10.0.0.1',
154                         'hostname' => 'server.example.com'
155                     ]
156                 ]
157             ]
158         );
159
160         $filtered = $this->filter->process($logEntry);
161         $data = $filtered->getData();
162
163         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
            request']['client']['ip']);
164         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
            request']['server']['ip']);
165         $this->assertEquals('[HOSTNAME REDACTED]', $data['
            request']['server']['hostname']);
166     }
167
168     public function testRedactsIP6Fields(): void
169     {
170         $logEntry = new LogEntry('1',
171             [
172                 'ip' => '2001:0db8:85a3:0000:0000:8a2e
                    :0370:7334',
173                 'ipAddress' => '2001:db8::1',
174                 'remote_addr' => '2001:4860:4860::8888',
175                 'user' => 'boris'
176             ]
177         );
178
179         $before =
180             [
181                 'ip' => '2001:0db8:85a3:0000:0000:8a2e
                    :0370:7334',
182                 'ipAddress' => '2001:db8::1',
183                 'remote_addr' => '2001:4860:4860::8888',
184                 'user' => 'boris'

```

```

183         ];
184
185         $filtered = $this->filter->process($logEntry);
186         $data = $filtered->getData();
187         $this->printRedactionResult($before, $data);
188
189         $this->assertEquals('[IP ADDRESS REDACTED]', $data['ip
190             ']);
191         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
192             ipAddress']);
193         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
194             remote_addr']);
195     }
196
197     public function testRedactsNestedIPv6Fields(): void
198     {
199         $logEntry = new LogEntry('1',
200             [
201                 'request' => [
202                     'client' => [
203                         'ip' => '::1',
204                     ],
205                     'server' => [
206                         'ip' => '2001:db8::/32',
207                         'hostname' => 'server.example.com',
208                     ]
209                 ]
210             ]
211         );
212
213         $filtered = $this->filter->process($logEntry);
214         $data = $filtered->getData();
215
216         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
217             request']['client']['ip']);
218         $this->assertEquals('[IP ADDRESS REDACTED]', $data['
219             request']['server']['ip']);
220         $this->assertEquals('[HOSTNAME REDACTED]', $data['
221             request']['server']['hostname']);
222     }
223
224     public function testRedactsIPv4InStrings(): void
225     {
226         $logEntry = new LogEntry('1',
227             [
228                 'message' => 'access from user with ipAddress:
229                     192.168.1.1'
230             ]
231         );
232
233         $before =

```

```

225         [
226             'message' => 'access from user with ipAddress:
                               192.168.1.1'
227         ];
228
229         $filtered = $this->filter->process($logEntry);
230         $data = $filtered->getData();
231         $this->printRedactionResult($before, $data);
232
233         $this->assertStringNotContainsString('192.168.1.1',
            $data['message']);
234     }
235
236     public function testRedactsJsonIpv4Patterns(): void
237     {
238         $logEntry = new LogEntry('1',
239             [
240                 'log' => '{"username":"boris","ip_address
                               ":"192.168.1.1","email":"boris@example.com
                               "}',
241             ]
242         );
243
244         $filtered = $this->filter->process($logEntry);
245         $data = $filtered->getData();
246
247         $this->assertStringContainsString('[IP ADDRESS REDACTED
            ]', $data['log']);
248         $this->assertStringNotContainsString('192.168.1.1',
            $data['log']);
249         $this->assertStringContainsString('boris@example',
            $data['log']);
250         $this->assertStringContainsString('boris', $data['log
            ']);
251     }
252
253     public function testRedactsIPv6InStrings(): void
254     {
255         $logEntry = new LogEntry('1',
256             [
257                 'message' => 'access from user with ipAddress:
                               2001:0db8:85a3:0000:0000:8a2e:0370:7334
                               bitte bleibe stehen'
258             ]
259         );
260
261         $before =
            [
                'message' => 'access from user with ipAddress:
                               2001:0db8:85a3:0000:0000:8a2e:0370:7334
                               bitte bleibe stehen'

```

```

262         ];
263
264         $filtered = $this->filter->process($logEntry);
265         $data = $filtered->getData();
266         $this->printRedactionResult($before, $data);
267
268         $this->assertStringNotContainsString('2001:0db8:85a3
          :0000:0000:8a2e:0370:7334', $data['message']);
269     }
270
271     public function testRedactsJsonIpv6Patterns(): void
272     {
273         $logEntry = new LogEntry('1',
274             [
275                 'log' => '{"username":"boris","ip_address
          ":"2001:0db8:85a3:0000:0000:8a2e
          :0370:7334","email":"boris@example.com"}',
276             ]);
277
278         $filtered = $this->filter->process($logEntry);
279         $data = $filtered->getData();
280
281         $this->assertStringContainsString('[IP ADDRESS REDACTED
          ]', $data['log']);
282         $this->assertStringNotContainsString('2001:0db8:85a3
          :0000:0000:8a2e:0370:7334', $data['log']);
283         $this->assertStringContainsString('boris@example',
          $data['log']);
284         $this->assertStringContainsString('boris', $data['log
          ']);
285     }
286
287     // -----
288     // MAC Tests
289     // -----
290
291     public function testRedactsMACFields(): void
292     {
293         $logEntry = new LogEntry('1',
294             [
295                 'mac' => '00:1A:2B:3C:4D:5E',
296                 'MAC' => '00-1A-2B-3C-4D-5E',
297                 'macAddress' => '00:1A:2B:3C:4D:5E',
298                 'mac_address' => '00:1A:2B:3C:4D:5E',
299                 'user' => 'boris'
300             ]);
301
302         $before =
303         [

```

```

304         'mac' => '00:1A:2B:3C:4D:5E',
305         'MAC' => '00-1A-2B-3C-4D-5E',
306         'macAddress' => '00:1A:2B:3C:4D:5E',
307         'mac_address' => '00:1A:2B:3C:4D:5E',
308         'user' => 'boris'
309     ];
310
311     $filtered = $this->filter->process($logEntry);
312     $data = $filtered->getData();
313     $this->printRedactionResult($before, $data);
314
315     $this->assertEquals('[MAC ADDRESS REDACTED]', $data['
        mac']);
316     $this->assertEquals('[MAC ADDRESS REDACTED]', $data['
        MAC']);
317     $this->assertEquals('[MAC ADDRESS REDACTED]', $data['
        mac_address']);
318     $this->assertEquals('[MAC ADDRESS REDACTED]', $data['
        macAddress']);
319 }
320
321 public function testRedactsMACPatternsInStrings(): void
322 {
323     $logEntry = new LogEntry('1',
324     [
325         'message' => 'User login with MAC=00:1A:2B:3C:4
        D:5E failed'
326     ]);
327
328     $filtered = $this->filter->process($logEntry);
329     $data = $filtered->getData();
330
331     $this->assertStringContainsString('[MAC ADDRESS
        REDACTED]', $data['message']);
332     $this->assertStringNotContainsString('password123',
        $data['message']);
333 }
334
335 // -----
336 // Host Tests
337 // -----
338
339 public function testRedactsHostFields(): void
340 {
341     $logEntry = new LogEntry('1',
342     [
343         'ip' => '192.168.1.1',
344         'ipAddress' => '10.0.0.1',
345         'remote_addr' => '127.0.0.1',

```

```

346         'user' => 'boris'
347     ]);
348
349     $before =
350     [
351         'ip' => '192.168.1.1',
352         'ipAddress' => '10.0.0.1',
353         'remote_addr' => '127.0.0.1',
354         'user' => 'boris'
355     ];
356
357     $filtered = $this->filter->process($logEntry);
358     $data = $filtered->getData();
359     $this->printRedactionResult($before, $data);
360
361     $this->assertEquals('[IP ADDRESS REDACTED]', $data['ip
362         ']);
363     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
364         ipAddress']);
365     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
366         remote_addr']);
367 }
368
369 public function testRedactsNestedHostFields(): void
370 {
371     $logEntry = new LogEntry('1',
372     [
373         'request' => [
374             'client' => [
375                 'ip' => '192.168.1.100',
376                 'port' => 8080
377             ],
378             'server' => [
379                 'ip' => '10.0.0.1',
380                 'hostname' => 'server.example.com'
381             ]
382         ]
383     ]);
384
385     $filtered = $this->filter->process($logEntry);
386     $data = $filtered->getData();
387
388     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
389         request']['client']['ip']);
390     $this->assertEquals('[IP ADDRESS REDACTED]', $data['
391         request']['server']['ip']);
392     $this->assertEquals('[HOSTNAME REDACTED]', $data['
393         request']['server']['hostname']);
394 }

```

A.18 Test-Skript

test_r.un

```

1  <?php
2
3  require_once __DIR__ . '/vendor/autoload.php';
4
5  use Logfilter\Service\OpenSearchService;
6  use Logfilter\Service\FilterPipeline;
7  use Logfilter\Service\LogFilterService;
8  use Logfilter\Service\Filter\CredentialsFilter;
9  use Logfilter\Service\Filter\NetworkFilter;
10 use Logfilter\Service\Filter\InfrastructureFilter;
11
12 $config = require __DIR__ . '/config.local.php';
13 $url = $config['opensearch_url'];
14 $user = $config['opensearch_user'];
15 $password = $config['opensearch_password'];
16 $index = $config['opensearch_index'];
17
18 $from = new DateTime('-2 minutes');
19 $to = new DateTime('now');
20
21 $openSearchService = new OpenSearchService(
22     url: $url,
23     user: $user,
24     password: $password,
25     sourceIndex: $index,
26     batchSize: 2500
27 );
28
29 $pipeline = new FilterPipeline();
30 $pipeline->addFilter(new CredentialsFilter());
31 $pipeline->addFilter(new NetworkFilter());
32 $pipeline->addFilter(new InfrastructureFilter());
33
34 $logFilterService = new LogFilterService(
35     openSearchService: $openSearchService,
36     filterPipeline: $pipeline,
37     batchSize: 2500
38 );
39
40 echo "Getting Logs from {$from->format('Y-m-d H:i:s')} to {$to
    ->format('Y-m-d H:i:s')}...\n\n";
41

```

```

42 $result = $logFilterService->processLogs($from, $to);
43
44 echo "\nFirst 10 filtered entries\n";
45 foreach (array_slice($result['filtered'], 0, 10) as $entry)
46 {
47     echo json_encode($entry, JSON_PRETTY_PRINT) . "\n\n";
48 }
49
50 $redactionLabels =
51 [
52     'PASSWORD' => '[PASSWORD REDACTED]',
53     'TOKEN' => '[TOKEN REDACTED]',
54     'API KEY' => '[API KEY REDACTED]',
55     'SESSION' => '[SESSION INFORMATION REDACTED]',
56     'SECRET' => '[SECRET REDACTED]',
57     'AUTHORIZATION' => '[AUTHORIZATION REDACTED]',
58     'IP ADDRESS' => '[IP ADDRESS REDACTED]',
59     'MAC ADDRESS' => '[MAC ADDRESS REDACTED]',
60     'HOSTNAME' => '[HOSTNAME REDACTED]',
61     'DATABASE INFORMATION' => '[DATABASE INFORMATION REDACTED
62     ]',
63 ];
64 $counters = array_fill_keys(array_keys($redactionLabels), 0);
65 $hits = [];
66
67 function scanForRedactions(array $data, string $path, array
68     $redactionLabels, array &$counters, array &$hits): void
69 {
70     foreach ($data as $key => $value)
71     {
72         $currentPath = $path ? "$path.$key" : $key;
73
74         if (is_array($value))
75         {
76             scanForRedactions($value, $currentPath,
77                 $redactionLabels, $counters, $hits);
78         }
79         elseif (is_string($value))
80         {
81             foreach ($redactionLabels as $type => $label)
82             {
83                 $count = substr_count($value, $label);
84                 if ($count > 0)
85                 {
86                     $counters[$type] += $count;
87                     $hits[] = "'$currentPath' => $value";
88                 }
89             }
90         }
91     }
92 }

```

```

88         }
89     }
90 }
91
92 foreach ($result['filtered'] as $entry)
93 {
94     scanForRedactions($entry['data'], '', $redactionLabels,
95         $counters, $hits);
96 }
97
98 echo "\nRedaction Summary\n";
99 $totalRedactions = array_sum($counters);
100 if ($totalRedactions === 0)
101 {
102     echo "No redactions found\n";
103 }
104 else
105 {
106     foreach ($counters as $type => $count)
107     {
108         if ($count > 0)
109         {
110             echo str_pad($type . ': ', 25) . $count . "\n";
111         }
112     }
113     echo str_pad('TOTAL: ', 25) . $totalRedactions . "\n";
114 }
115
116 $stats = $result['statistics'];
117
118 echo "\nStatistics\n";
119 echo str_pad('Time-Frame: ', 25) . $from->format('Y-m-d H:i:s')
120     . ' to ' . $to->format('Y-m-d H:i:s') . "\n";
121 echo str_pad('Fetched: ', 25) . $stats['total_fetched'] . "\n";
122 echo str_pad('Processed: ', 25) . $stats['total_processed'] . "\n";
123 echo str_pad('Batches: ', 25) . $stats['batches_processed'] . "\n";
124
125 $ms = $stats['processing_time_ms'];
126 $seconds = round($ms / 1000, 3);
127 echo str_pad('Duration of Filtering Process: ', 25) . $ms . "
128     ms ({ $seconds } s)\n";
129 echo str_pad('Filtering Logic Time: ', 25) . $stats['
130     filtering_only_ms'] . " ms\n";
131
132 if (!empty($result['statistics']['errors']))
133 {
134     echo "\nErrors:\n";
135     foreach ($stats['errors'] as $error)

```

```
131     {
132         echo " - $error\n";
133     }
134 }
```

A.19 Config

config.local

```
1 <?php
2
3 return
4 [
5     'opensearch_url' => '',
6     'opensearch_user' => '',
7     'opensearch_password' => '',
8     'opensearch_index' => '',
9     'batch_size' => 2500,
10    'scroll_window' => '5m',
11 ];
```