

*Berner Fachhochschule
BTI3031 Project 1*

TraceSentry

AI-Aided Caches-n-Logs Monitoring-n-Wiping Daemon

Luca Scherer, Janic Scherer, Luca Ammann

Betreuer: Dr. Simon Kramer

17. Januar 2025

Zusammenfassung

TraceSentry ist eine plattformunabhängige und KI-gestützte Anwendung zur Suche, Überwachung, Bewertung und Bereinigung von Cache- und Logdateien. In einer praxisorientierten Fallstudie wurde das KI-Modell *GPT-4o Mini* von OpenAI ausgewählt und evaluiert, um Dateiinhalte zu bewerten und potenzielle Risiken zu erkennen. Die Anwendung umfasst eine Befehlszeilenschnittstelle (CLI) und einen Hintergrundprozess (Daemon), der in regelmäßigen Abständen Snapshots der überwachten Verzeichnisse erstellt. Diese Snapshots dienen als Momentaufnahmen der Verzeichnisse, werden als Merkle-Bäume konstruiert und miteinander verglichen. Eine SQLite-Datenbank wird verwendet, um die Snapshots sowie die erkannten Änderungen zu persistieren. Die modulare Architektur der Anwendung, die in Java unter Verwendung des Spring-Frameworks implementiert wurde, ermöglicht die Erweiterung um zusätzliche Funktionen oder Benutzeroberflächen.

Dieses Dokument beschreibt die technische Spezifikation und Umsetzung, die verwendeten Prozesse, eine umfangreiche Evaluation der Anwendung, die KI-Fallstudie sowie einen Ausblick auf zukünftige Arbeiten.

Inhaltsverzeichnis

Inhaltsverzeichnis	1
Tabellenverzeichnis	4
Abbildungsverzeichnis	5
Code-Ausschnitte	6
1 Einleitung	7
1.1 Ausgangssituation	7
1.1.1 Problem	7
1.1.2 Chance	7
1.1.3 Antrag	7
1.2 Projektziel	8
1.3 Prioritäten	8
2 Spezifikation	9
2.1 Systemabgrenzung	9
2.1.1 Systemumgebung (statisch)	9
2.1.2 Prozessumgebung (dynamisch)	11
2.2 Anforderungen	13
2.2.1 Funktionale Anforderungen	13
2.2.2 Grenz- und Vorbedingungen	14
2.2.3 Testkonzept	16
2.3 Usability	17
2.3.1 Personas	17
2.3.2 Storyboard	18
2.3.3 UX-Prototyping	19
3 Implementierung	20
3.1 Architektur	20
3.1.1 Technische Variantenentscheide	20
3.1.2 Lösungsarchitektur	28
3.1.3 Realisierung	30
3.2 Prozesse	42
3.2.1 Rollen	42
3.2.2 Sprintziele	42
3.2.3 Anwendung	42

3.2.4	Adaptionen	45
4	Evaluation und Tests	46
4.1	Manuelle End-2-End Tests	46
4.1.1	Testfälle	46
4.1.2	Testergebnisse	54
4.2	Messungen (Performanztests)	55
4.2.1	Suchfunktion	55
4.2.2	Monitoring	58
4.3	KI Fallstudie	60
4.3.1	Wahl des KI-Modells	60
4.3.2	Aktuelle Einschränkungen	61
4.3.3	Vorgehensweise	61
4.3.4	Testdateien	61
4.3.5	Ergebnisse	64
4.3.6	Kostenanalyse	65
4.3.7	Fazit	65
5	Bereitstellung/Integration	66
5.1	Lizenzierung	67
5.1.1	Open-Source Abhängigkeiten	67
5.1.2	Kommerzielle Abhängigkeiten	67
5.2	Installationshandbuch & Skript	68
5.2.1	OpenAI API Key	68
5.2.2	Skripte	68
5.2.3	Manuelle Installation	71
5.3	Benutzerhandbuch	74
6	Fazit	75
6.1	Diskussion	75
6.2	Endergebnis	76
6.2.1	Berechnung der Produktivitätssteigerung	76
6.3	Zukünftige Arbeiten	80
6.3.1	Funktionale Erweiterungen	80
6.3.2	User Experience	80
6.3.3	Sicherheit	81
6.3.4	Erweiterung der KI-Modelle bzw. Anbindung	81
6.3.5	Dateikompatibilität	81
	Glossar	81
	Index	84
	Literaturverzeichnis	85
A	Anhang	86
A.1	Originale Projektbeschreibung	87
A.2	CLI Dokumentation	90
A.3	API-Dokumentation	100

A.4	Klassendiagramm CLI	106
A.5	Klassendiagramm Daemon	108
A.6	Klassendiagramm Lib	110
A.7	Eigenständigkeitserklärung	112

Tabellenverzeichnis

2.1	Nichtfunktionale Anforderung NFR-001	14
2.2	Nichtfunktionale Anforderung NFR-002	14
2.3	Nichtfunktionale Anforderung NFR-003	14
2.4	Nichtfunktionale Anforderung NFR-004	15
3.1	Technologien der Komponenten	29
3.2	Scrum Rollen	42
3.3	Weitere Rollen	42
4.1	Manueller End-2-End Test E2E-001	46
4.2	Manueller End-2-End Test E2E-002	47
4.3	Manueller End-2-End Test E2E-003	47
4.4	Manueller End-2-End Test E2E-004	48
4.5	Manueller End-2-End Test E2E-005	49
4.6	Manueller End-2-End Test E2E-006	50
4.7	Manueller End-2-End Test E2E-007	52
4.8	Manueller End-2-End Test E2E-008	52
4.9	Manueller End-2-End Test E2E-009	53
4.10	Manuelle End-2-End Testergebnisse	54
4.11	Umgebungsangaben und Metriken Suchfunktion-Performanztest	55
4.12	Performanztest Suche WIN-1	56
4.13	Performanztest Suche WIN-2	56
4.14	Performanztest Suche MAC-1	56
4.15	Performanztest Suche MAC-2	57
4.16	Performanztest Suche LIN-1	57
4.17	Performanztest Suche LIN-2	57
4.18	Umgebungsangaben und Metriken Monitoring-Performanztest	58
4.19	Performanztest Monitoring WIN	58
4.20	Performanztest Monitoring MAC	59
4.21	Performanztest Monitoring LIN	59
4.22	Generierte Log-Typen	63

Abbildungsverzeichnis

2.1	Softwarekomponenten (High-Level)	9
2.2	Prozess KI-Bewertung (UML Communication Diagram)	11
2.3	Prozess periodische Überwachung (UML Communication Diagram) . . .	12
2.4	Anwendungsfälle (UML Use Case Diagram)	13
3.1	UML Komponentendiagramm	28
3.2	Sequenzdiagramm Inspect Command	30
3.3	Sequenzdiagramm Monitoring Controller	31
3.4	Sequenzdiagramm Monitoring Scheduler	33
3.5	Entity Relationship Diagram	37
3.6	Sequenzdiagramm InspectController	39
3.7	Sequenzdiagramm GPTInspectionService	39
3.8	Epic Beispiel	43
3.9	User-Story Beispiel	44
3.10	Taskboard Beispiel	45
5.1	UML Deploymentdiagramm	66

Code-Ausschnitte

3.1	OpenAI Anfrage	40
3.2	OpenAI Antwort	40
5.1	Linux Berechtigungen	69
5.2	Linux Installation	69
5.3	Windows Berechtigungen	69
5.4	Windows Installation	69
5.5	MacOS Berechtigungen	70
5.6	MacOS Quarantäne-Tag	70
5.7	MacOS Installation	70

1 Einleitung

1.1 Ausgangssituation

1.1.1 Problem

Cache- und Logdateien sind (temporäre) Dateien, die von Betriebssystemen und Anwendungen zur Speicherung von Zwischenständen, Nutzeraktivitäten und Systemereignissen erstellt werden. Viele Computerbenutzer sind sich nicht bewusst, welche Arten dieser Dateien auf ihrem Computer existieren und insbesondere, welche Applikationen diese lesen, schreiben, verändern oder löschen. Dazu sind sich Anwender nicht bewusst, welchen Zwecken diese Dateien dienen und welche möglicherweise vertrauliche Informationen enthalten, die für Dritte von Interesse sein könnten. Dieses Unkenntnis kann zu Datenschutzrisiken und fehlender Kontrolle führen.

1.1.2 Chance

Durch das Angebot an generativen KI-Modellen, die - meist frei oder relativ günstig - im Internet zugänglich sind, lässt sich schnell Fachkenntnis zu solchen Dateien einholen. Betrachtet werden hierbei vor allem die Struktur (Syntax) und Bedeutung (Semantik) von Dateiinhalten. Des Weiteren lassen sich Schlüsse aus Änderungen, Löschungen und Neuerstellungen von Dateien ziehen.¹ Durch das automatisierte Abfragen dieser KI-Modelle in einer eigenen Anwendung, welche unter anderem Momentaufnahmen (sogenannte Snapshots) relevanter Dateipfade und Metadaten speichert, sowie weitere Mehrwerte in Bezug auf die Nutzererfahrung realisiert, kann einem Benutzer auf einfache Art und Weise die Möglichkeit zur Kontrolle und Erleichterung dieses Umstandes geboten werden.

1.1.3 Antrag

Das Projekt-Proposal (siehe Originale Projektbeschreibung) umfasst einerseits die Entwicklung einer Software, welche die besagten Anforderungen erfüllt andererseits das Ermitteln des Potentials aktueller KI-Modelle zur Lösung des genannten Problems. Einziger Stakeholder ist Simon Kramer als Betreuer und Auftraggeber.

¹Evaluation durch KI-Modell in Bezug auf die Bedeutung von Änderungen, Löschungen und Neuerstellungen nicht weiter realisiert, siehe Abschnitt Zukünftige Arbeiten bzw. Funktionale Erweiterungen.

1.2 Projektziel

Es soll eine Anwendung entwickelt werden, welche die **Suche, Überwachung und Löschung von Cache- und Logdateien** ermöglicht. Dazu gehört das **Einholen von Einschätzungen eines KI-Modells**, welches die Dateien analysiert und bewertet. Die Applikation soll **plattformunabhängig, benutzerfreundlich** und mit der nötigen **Dokumentation** für den Benutzer (User-Manual u.a.) ausgestattet sein.

Das KI-Modell wird nur abgefragt bzw. eingebunden und dessen Entwicklung und Betrieb wird vom Projekt abgegrenzt. Passend zu dieser Abgrenzung soll als Komponente des Berichts das Potential von KI-Modellen bzw. des eingesetzten KI-Modells zur Lösung des Problems evaluiert werden.

1.3 Prioritäten

Grundsätzlich soll ein fertiges Produkt, welches dem Benutzer einen Mehrwert bietet, im Vordergrund stehen. Gemäss Checkliste²:

Resultat- statt Prozess-Orientierung priorisieren:

Fertiges Projekt anstreben (Prozesse, die unfertige Resultate produzieren, vermeiden)

Programme sind bessere Modelle (Spezifikations- und Design-Endlosschlaufen vermeiden)

Priorisiert sollen dabei in erster Linie Funktionalitäten eines MVP (Minimum Viable Product) sein, welche die Kernfunktionalität des Produkts sicherstellen. Dazu gehören vor allem die Suche und Überwachung von Dateien. Zur Überwachung gehört das Erstellen von Snapshots und das Vergleichen dieser, sowie das Einholen von Einschätzungen eines KI-Modells. Zudem werden alle Tätigkeiten, die dabei mitschwingen, wie Dokumentation, Tests und Codequalität, als wichtig erachtet. Zurück priorisierte Funktionalitäten, sowie Erweiterungen werden im Kapitel Zukünftige Arbeiten beschrieben.

²via Mail von Simon Kramer am 26.09.2024

2 Spezifikation

2.1 Systemabgrenzung

2.1.1 Systemumgebung (statisch)

2.1.1.1 Systemübersicht

Der Hauptverwendungszweck des TraceSentry ist das Überwachen von Dateien in bestimmten Verzeichnissen. Dabei liegt der Fokus auf Log- und Cache-Dateien, welche periodisch auf Änderungen in Form von Erstellung, Löschung oder Veränderung überprüft werden. Diese periodische Überwachung soll in einem Hintergrundprozess, einem so genannten Daemon, stattfinden. Notwendige Daten für die Überwachung sowie durch die Überwachung entstandene Daten, werden in einer Datenbank gespeichert. Verdächtige Dateien können durch eine KI-gestützte Analyse bewertet und anschliessend geleert oder gelöscht werden.

2.1.1.2 Softwarekomponenten

Die System- sowie Softwarearchitektur wurde im Rahmen dieser Arbeit entworfen und grundlegende Entscheide im Kapitel Technische Variantenentscheide festgehalten. Alle Softwarekomponenten wurden in Java unter der Verwendung des Spring-Frameworks entwickelt. Die endgültig erarbeitete Architektur wird im Kapitel Lösungsarchitektur beschrieben. Nachfolgend eine vereinfachte Übersicht der Softwarekomponenten:

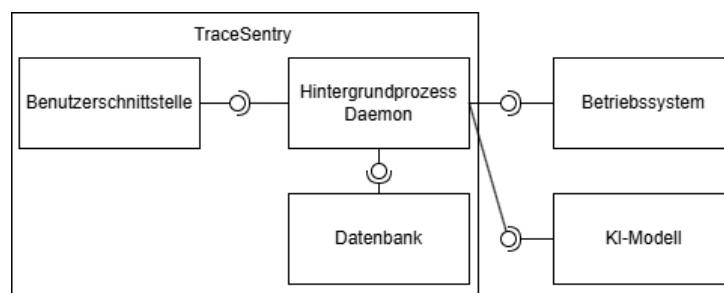


Abbildung 2.1: Softwarekomponenten (High-Level)

Die Bereitstellung und Laufzeitumgebung des Systems wird im Kapitel Bereitstellung/Integration beschrieben.

2.1.1.3 Hardwarekompatibilität

Der TraceSentry ist ein eigenständiges, plattformunabhängiges System, welches auf den gängigen Betriebssystemen (Windows, Linux, MacOS) lauffähig ist. Entwickelt und getestet wurde auf folgenden Betriebssystemen:

- Windows 11–10.0
- Linux 6.1.0-28
- macOS Sequoia 15.2

2.1.1.4 Benutzerschnittstelle

Im Rahmen der Anforderungsspezifikation wurde eine Benutzerschnittstelle in Form eines Command-Line-Interfaces (CLI) spezifiziert. Im Hinblick auf Erweiterung und Wartbarkeit ist die Benutzerschnittstelle logisch von der Geschäftslogik getrennt. Aufgrund dessen wurde die Benutzerschnittstelle als eigenständige Laufzeiteinheit definiert und entwickelt. Die Kommunikation zwischen CLI und Daemon erfolgt über eine REST-API. Diese Wahl wurde im Kapitel Technische Variantenentscheide begründet.

2.1.1.5 Datenbank

Die Datenbank wird für die Persistierung von Überwachungs-Metadaten sowie -Ergebnissen verwendet. Sie befindet sich lokal auf dem System in Form einer SQLite-Datenbank. Die Wahl des DBMS sowie dessen genaue Implementierung wird in den Kapiteln Technologieentscheid und Datenbank beschrieben.

2.1.1.6 KI-Anbindung

Es wurde das KI-Modell *GPT-4o mini* von OpenAI verwendet. Die Anbindung an das KI-Modell wird im Kapitel KI-Anbindung beschrieben. Zudem wurde eine Fallstudie zur Evaluation des Potentials eines aktuellen KI-Modells im Kontext des TraceSentry durchgeführt. Die Vorgehensweise sowie die Resultate dieser Fallstudie sind im Kapitel KI Fallstudie dokumentiert.

2.1.2 Prozessumgebung (dynamisch)

2.1.2.1 Daemon Lifecycle

Der Daemon-Prozess kann on-demand oder als Autostart-Service gestartet werden. Er läuft autonom im Hintergrund und kann über die CLI angesteuert werden.

2.1.2.2 CLI Lifecycle

Die CLI wird on-demand pro Benutzerinteraktion gestartet und beendet. Das bedeutet, dass die Lebensdauer der CLI auf die Dauer einer Benutzerinteraktion beschränkt ist. Zusätzlich hat der Benutzer die Möglichkeit, die CLI im sogenannten *interactive mode* zu starten. In diesem Modus steht dem Benutzer eine interaktive Shell zur Verfügung, in welcher er mehrere Befehle hintereinander ausführen kann. Im *interactive mode* wird die CLI nicht beendet, bis der Benutzer dies explizit veranlasst. Dies ist ein gängiges Verhalten von Command-Line-Interfaces und ermöglicht eine effiziente Interaktion mit dem System, ohne dass für jede Benutzerinteraktion ein neuer Prozess gestartet werden muss.

2.1.2.3 Prozessübersicht

Nachfolgend wird davon ausgegangen, dass der Daemon-Prozess beim Systemstart gestartet wurde und sich der Benutzer im interaktiven Modus der CLI befindet.

Prozess - KI-Bewertung In diesem Communication Diagram wird der Prozess der durch den Benutzer angeforderten KI-Bewertung dargestellt. Die Interaktion mit dem Dateisystem ist nicht modelliert.

Der Prozess ist für andere Funktionen des Systems identisch, weswegen auf weitere Modellierung verzichtet wird. Relevant für alle Prozesse ist die Kommunikation zwischen Benutzer und Daemon via CLI. Je nach Funktion interagiert der Daemon-Prozess mit dem Dateisystem, der Datenbank und/oder dem KI-Modell.

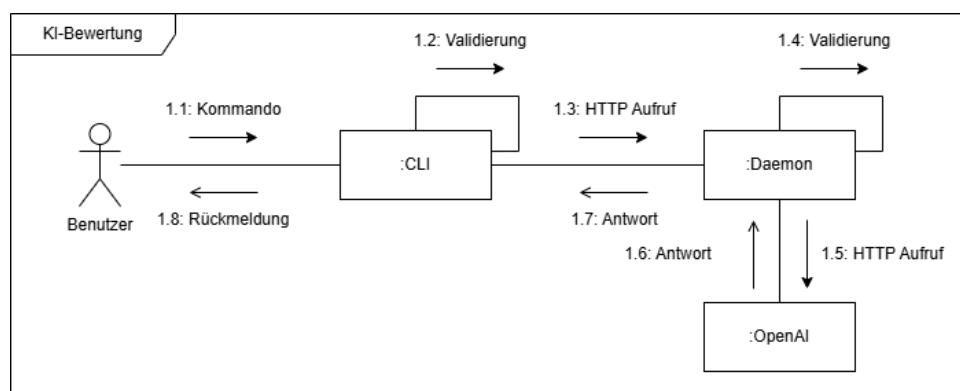


Abbildung 2.2: Prozess KI-Bewertung (UML Communication Diagram)

Prozess - periodische Überwachung Ein spezieller Prozess im System ist die periodische Überwachung von Dateien, da dieser autonom und ohne Benutzerinteraktion abläuft. Im folgenden Communication Diagram wird dieser Prozess modelliert. Der Daemon-Prozess ist selbstständig für die Auslösung der Überwachung verantwortlich und in dieser Hinsicht nicht vom Betriebssystem abhängig. Die Interaktion mit dem Dateisystem ist nicht modelliert.

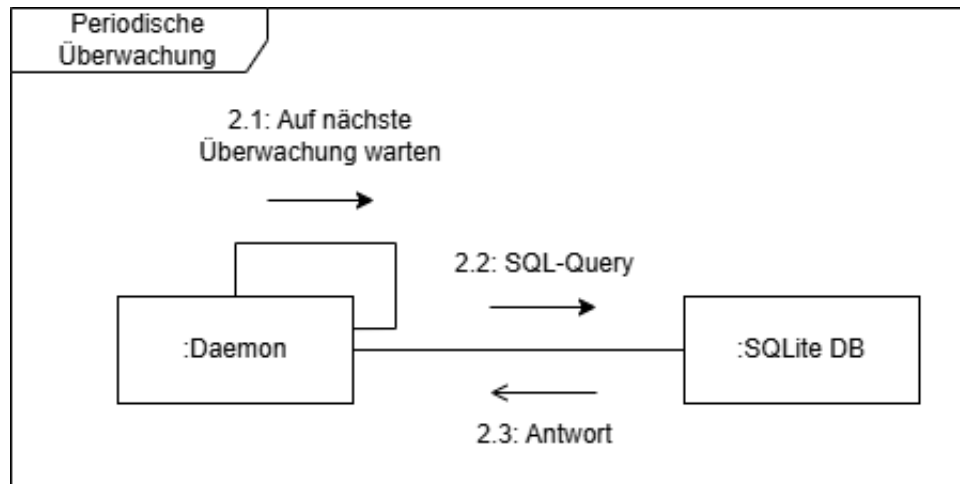


Abbildung 2.3: Prozess periodische Überwachung (UML Communication Diagram)

2.1.2.4 Fehlerbehandlung

Wie in der Abbildung 2.2 ersichtlich, werden Fehler grundsätzlich auf zwei Ebenen behandelt.

CLI Syntaktische oder semantische Fehler eines Kommandos werden durch die CLI behandelt. Die CLI gibt dem Benutzer, ohne Kommunikation mit dem Daemon, eine entsprechende Fehlermeldung zurück.

Daemon Fehler, die erst während der Ausführung eines Befehls entstehen, werden durch den Daemon behandelt. Ein Beispiel dafür wäre eine nicht vorhandene bzw. lesbare Datei. In diesem Fall wird der Fehler via REST-API an die CLI zurückgegeben und dem Benutzer angezeigt. Fehler, die während einer periodischen Überwachung entstehen, werden geloggt und falls notwendig wird die Überwachung verworfen.

2.2 Anforderungen

Die Anforderungen für den AI-Aided Caches-n-Logs Monitoring-n-Wiping Demon (TraceSentry) ergaben sich aus dem Projekt Proposal (siehe Anhang A.1). sowie einem initialen Termin mit Herr Kramer.

2.2.1 Funktionale Anforderungen

Alle funktionalen Anforderungen werden in Form von User Stories im Scrum Product Backlog geführt. Dieser wird laufend erweitert beziehungsweise konkretisiert. Nachfolgend werden die grundlegenden Anwendungsfälle aus Benutzersicht aufgezeigt. Die An-

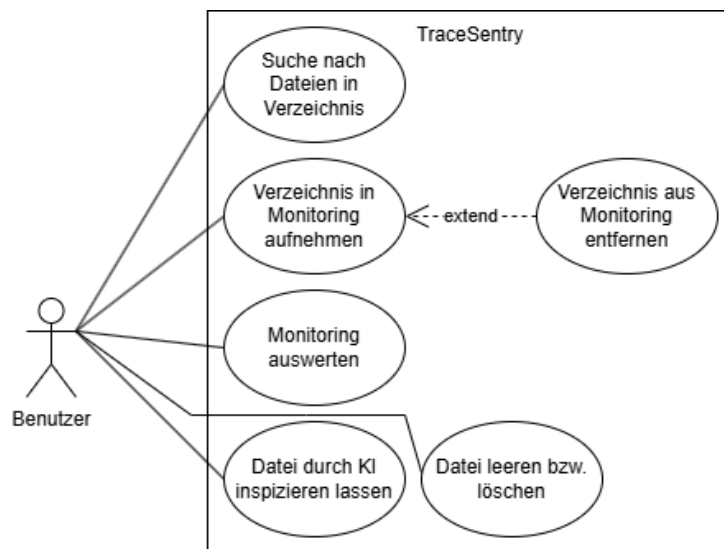


Abbildung 2.4: Anwendungsfälle (UML Use Case Diagram)

wendungsfälle wurden in folgende Hauptanforderungen an das System übersetzt, welche laufend konkretisiert, aufgebrochen und in User Stories inkl. Akzeptanzkriterien abgebildet wurden.

- Das System findet jegliche Dateien (insbesondere Cache- und Logdateien), welche sich in einem angegebenen Verzeichnis (inkl. Unterverzeichnisse) befinden.
- Das System durchsucht periodisch angegebene Verzeichnisse (inkl. Unterverzeichnisse) nach Log- oder Cache-Dateien.
- Das System erstellt periodisch sogenannte Snapshots der gefundenen Dateien, um diese später miteinander zu vergleichen.
- Das System identifiziert veränderte Dateien anhand der periodisch erstellten Snapshots.
- Das System bewertet Dateien nach deren Schädlichkeit und gibt eine Empfehlung ab, ob die Datei gelöscht oder geleert werden soll.
- Das System kann angegebene Dateien leeren oder löschen.

Auf die detaillierte Darstellung aller User Stories wird bewusst verzichtet. Der Einsatz von Scrum als agile Arbeitsmethode wird im Kapitel Prozesse beschrieben.

2.2.2 Grenz- und Vorbedingungen

2.2.2.1 Nichtfunktionale Anforderungen

Im Scrum Product Backlog werden nichtfunktionale Anforderungen bzw. Grenz- oder Vorbedingungen nicht explizit als User Story geführt. Aus dem Projekt-Proposal und dem initialen Termin zu Projektstart ergaben sich nachfolgende nichtfunktionale Anforderungen. Diese beeinflussen massgeblich die Architektur des Systems. Keine User Story darf eine oder mehrere nichtfunktionale Anforderungen verletzen, was durch die Definition of Done sichergestellt und laufend geprüft wird.

ID	NFR-001
Anforderung	Hintergrundprozess mit periodischem Task (Daemon)
Beschreibung	Das System muss einen Hintergrundprozess implementieren, der periodisch Aktionen durchführen kann.
Akzeptanzkriterien	<ul style="list-style-type: none">• Nach dem Start des Prozesses, ist dieser für den Benutzer nicht mehr ersichtlich.• Der Hintergrundprozess ist erweiterbar, sodass dieser periodisch und autonom Aktionen durchführen kann.

Tabelle 2.1: Nichtfunktionale Anforderung NFR-001

ID	NFR-002
Anforderung	Benutzerschnittstelle via Konsole (CLI)
Beschreibung	Mit dem laufenden Daemon soll via Konsole interagiert werden können.
Akzeptanzkriterien	<ul style="list-style-type: none">• Der Daemon kann via CLI gestartet werden.• Jegliche Interaktionen mit dem Daemon sind via CLI verfügbar.

Tabelle 2.2: Nichtfunktionale Anforderung NFR-002

ID	NFR-003
Anforderung	Systemunabhängigkeit
Beschreibung	Lauffähigkeit mit vollem Funktionsumfang auf gängigen Betriebssystemen.
Akzeptanzkriterien	<ul style="list-style-type: none">• Hintergrundprozess sowie Konsolenschnittstelle sind auf den gängigen Betriebssystemen lauffähig und mit vollem Funktionsumfang verwendbar.• Alle Funktionen werden auf den Betriebssystemen Windows 11–10.0, Linux 6.1.0-28 und macOS Sequoia 15.2 getestet.

Tabelle 2.3: Nichtfunktionale Anforderung NFR-003

ID	NFR-004
Anforderung	Codequalität & erweiterbare Architektur
Beschreibung	Minimaler, modularer und selbsterklärender Code
Akzeptanzkriterien	<ul style="list-style-type: none"> • Die Architektur wird so aufgebaut, dass neue Features problemlos ergänzt werden können (z.B. GUI). • Tests gemäss Testkonzept. • Statische Analysen zeigen keine schwerwiegenden Verstösse gegen Linting-Regeln. • Anwendungen von gängigen Design-Prinzipien wie SOLID, DRY, KISS etc. • Für jede Codeerweiterung wird ein Codereview durch einen zweiten Entwickler vorgenommen.

Tabelle 2.4: Nichtfunktionale Anforderung NFR-004

2.2.2.2 Aktuelle Limitationen

Die Funktionalität des Systems ist wie folgt eingeschränkt:

- **KI-Modell:** Im Rahmen dieser Arbeit wurde nur das KI-Modell *GPT-4o mini* von OpenAI integriert. Mit diesem lassen sich Dateien von knapp 128'000 Token bzw. rund 320'000 Zeichen analysieren. (siehe Erweiterung der KI-Modelle bzw. Anbindung)
- **Dateigrösse:** Die maximale unterstützte Dateigrösse für die Analyse und periodische Überwachung beträgt 2GB. Diese Limitation hängt mit der Java Files API zusammen. (siehe Dateikompatibilität)

2.2.2.3 Voraussetzungen

Das System wurde unter den folgenden Voraussetzungen entwickelt und getestet:

- **Java:** Es wird eine Installation der Java Runtime Environment (JRE) in der Version 21 vorausgesetzt.
- **Dateitypen:** Das System wurde hauptsächlich mit Dateitypen getestet, die als Plaintext vorliegen, wie beispielsweise Log- oder Textdateien.
- **KI-Modell:** Für die Nutzung des OpenAI-Modells wird eine Internetverbindung sowie ein gültiger API-Key benötigt.

2.2.3 Testkonzept

2.2.3.1 Ziel des Testkonzepts

- Sicherstellen, dass das gesamte System stabil betrieben werden kann und seine Hauptfunktionen zuverlässig verfügbar sind.
- Minimales Testing-Setup, um den Entwicklungsaufwand gering zu halten.
- Fokus auf Integrationstests und funktionale Tests zur Überprüfung der gesamten Systemfunktionalität.
- Klare Definition, welche Tests während der Entwicklung (Teil einer User Story) umgesetzt werden sollen.

2.2.3.2 Testarten und Testabdeckung

- **Unit-Tests:** Auf Unit-Tests wird verzichtet und eine möglichst hohe Code-Coverage durch Integrationstests angestrebt. In konkreten Fällen wie Funktionen für die Such- und Überwachungsfunktionalität, können Unit-Tests für isolierte Logik implementiert werden.
- **Integrationstests:** Testen der Zusammenarbeit mehrerer Komponenten.
 - **CLI-Befehle:** Überprüfen, dass die CLI-Kommandos (z.B. `run` oder `search`) korrekt ausgeführt werden und die erwarteten Parameter validieren.
 - **REST-HTTP-Anfragen:** Testen, ob alle HTTP-Schnittstellen korrekt reagieren. (Happy- sowie Exception-Paths)
- **End-to-End Tests:** Fokus auf End-to-End-Szenarien zur Validierung der zentralen Funktionalität. (Siehe Manuelle End-2-End Tests)

2.2.3.3 Test-Setup und -Konfiguration

- **Mocking von Ressourcen:** Verwenden von Mocks für Datenbank- und Dateisystemzugriffe.
- **Testumgebung:** Sofern sinnvoll, werden z.B. In-Memory-Datenbanken verwendet. Für Tests mit Dateisystem-Abhängigkeit können temporäre Verzeichnisse erstellt werden.

2.3 Usability

2.3.1 Personas

Nachfolgend handelt es sich um eine fiktive Person, die als Repräsentation eines typischen Benutzers des Systems dient.

2.3.1.1 Thomas

Thomas ist ein 35-Jähriger Berufsschullehrer, der Informatiker und Elektroniker unterrichtet. Für seine Schüler unterrichtet er unter anderem auch Systemadministration und IT-Sicherheit.

Thomas ist sehr interessiert an neuen Technologien und ist was Sicherheitsaspekte betrifft sehr sensibilisiert. Daher stellt er sich die Frage, warum so viele Log- und Cache-Dateien auf seinem Rechner sind, ob diese alle ihre Daseinsberechtigung haben und auch, ob das Lesen und Schreiben dieser Dateien durch Drittsoftware nicht ein Sicherheitsrisiko darstellt.

Thomas ist ein sehr erfahrener Benutzer und hat keine Probleme mit der Kommandozeile.

Aufgaben	KnowHow stärken (in Bezug auf seine Lehrtätigkeiten)
Ziele	Veränderungen beobachten können und diese besser verstehen.
Wünsche	Intuitive Benutzerschnittstelle und Automatisierung.
Hindernisse	Trotz Erfahrung in der Informatik, sind teilweise Wissenslücken vorhanden.

2.3.2 Storyboard

In diesem Abschnitt wird beschrieben, wie ein Nutzer mit den verschiedenen Funktionen der Software interagieren kann. Dafür ist folgend eine sinnvolle Abfolge zusammengestellt, wie ein Nutzer die Software in einem realen Anwendungsfall verwenden könnte.

1. Dateien suchen

Der Nutzer hat kürzlich in seinem Task-Manager ein Programm entdeckt, das komischerweise ziemlich viel Speicher braucht. Also verwendet er den `search`-Befehl mit dem Pfad auf die Installation von diesem Programm und ihm werden, die in diesem Verzeichnis gefundenen, Cache- sowie Log-Dateien aufgelistet:

```
internal/test.log  
internal/cache.jsonx
```

2. Dateien überwachen

Nun möchte der Nutzer dieses Verzeichnis `/internal` innerhalb des Programmverzeichnisses genauer überwachen. Also fügt er diesen Pfad mit dem `monitor add`-Befehl hinzu, wonach dieses Verzeichnis vom TraceSentry stündlich überwacht wird.

3. Snapshots vergleichen

Nach einigen Stunden möchte der Nutzer wissen was in diesem überwachten Verzeichnis passiert ist. Dafür nutzt er den `snapshots compare`-Befehl, um alle Zustände des Verzeichnisses vom aktuellen Zeitpunkt bis zum Start der Überwachung vergleichen zu können. Das System gibt ihm eine entsprechende Ausgabe:

```
Vergleich vom Verzeichnis /internal  
vom Zeitpunkt 01.12.2024 15:00:00 bis zum Zeitpunkt 01.12.2024 22:00:00:  
  
test.log CHANGED 7-times
```

4. KI-Abfrage für eine Datei

Als nächstes möchte der Nutzer wissen, wofür diese `test.log` Datei ist, angesichts der Tatsache, dass sich diese jede Stunde ändert. Also nutzt er den `inspect`-Befehl, um die angebundene KI abzufragen, wofür diese Datei genau ist, ob sie ein Sicherheitsrisiko darstellt und ob er die Datei evtl. leeren oder sogar löschen sollte. Es resultiert eine standardisierte Ausgabe, welche den Verwendungszweck, eine Schädlichkeitsbewertung und eine Empfehlung enthält.

5. Datei leeren oder löschen

Der Nutzer kann nun selbst entscheiden, was er mit dieser Information anfangen möchte. In diesem Fall möchte er dem Ratschlag der KI folgen und die Datei leeren. Er nutzt also den `wipe`-Befehl, gibt dabei den Pfad der `test.log` Datei an und das System leert die Datei.

2.3.3 UX-Prototyping

Da das System ausschliesslich über ein Command-Line-Interface (CLI) verfügt, wird die Dokumentation dieser Schnittstelle im Kapitel Benutzerhandbuch umfassend behandelt. Diese Dokumentation wurde im Rahmen eines iterativen und agilen Entwicklungsprozesses kontinuierlich, während der Implementierung der einzelnen Befehle, erstellt. Sie enthält eine detaillierte Beschreibung aller verfügbaren Befehle, einschliesslich ihrer Syntax, Parameter und Anwendungsbeispiele. Darüber hinaus werden mögliche Systemantworten im Kontext verschiedener Szenarien erläutert, um eine praxisnahe Nutzung zu gewährleisten.

3 Implementierung

3.1 Architektur

3.1.1 Technische Variantenentscheide

3.1.1.1 Architekturentscheid

In diesem Abschnitt werden die Überlegungen zur Architektur des Systems dargestellt. Es wird erläutert, welche Varianten in Betracht gezogen wurden, welche Entscheidung getroffen wurde und welche Gründe dazu geführt haben.

1. Datenbank

Eine zentrale Anforderung besteht darin, dass Pfad-Analysen zu einem späteren Zeitpunkt miteinander verglichen werden können. Daraus ergibt sich die Notwendigkeit, eine Datenbank (DBMS) zu integrieren. Im Folgenden werden die in Betracht gezogenen Varianten analysiert.

1.1 In-Memory

Bei dieser Variante wird die Datenbank während der Laufzeit des Programms im Arbeitsspeicher initialisiert und beim Beenden des Programms wieder gelöscht. Dies bedeutet, dass die Datenbank nur temporär verfügbar ist und den Hauptspeicher des Systems nutzt.

Vorteile

- Sehr hohe Zugriffsgeschwindigkeit, da die Daten im Arbeitsspeicher gehalten werden.
- Einfache Integration

Nachteile

- Keine Persistenz der Daten über die Laufzeit hinaus.
- Erhöhter Speicherbedarf während der Laufzeit des Programms.

Fazit Diese Variante wurde ausgeschlossen, da die Persistenz der Daten eine essentielle Anforderung darstellt. Eine In-Memory-Datenbank würde nicht ermöglichen, dass Analysen auch nach einem Neustart des Systems verfügbar bleiben.

1.2 Remote

Bei dieser Variante wird die Datenbank auf einem externen System betrieben und über das Netzwerk angesprochen. Die Daten bleiben persistent, solange sie auf dem externen System nicht gelöscht werden und dieses ordnungsgemäss funktioniert.

Vorteile

- Höhere Datensicherheit, da die Daten auf einem separaten System gespeichert werden können, welches durch Backup-Strategien abgesichert ist.
- Grössere Speicherkapazität, da die Skalierung auf leistungsfähige Server oder Cloud-Dienste möglich ist.

Nachteile

- Abhängigkeit von einer stabilen Netzwerkverbindung, wodurch das System ohne Internet nicht verfügbar ist.
- Zusätzlicher Aufwand und Kosten durch die Einrichtung und Wartung der externen Infrastruktur.

Fazit Die Remote-Datenbank wurde als ungeeignet für die vorhandenen Anwendungsfälle bewertet. Die Vorteile wie erhöhte Datensicherheit und Skalierbarkeit sind nicht essentiell. Die zusätzlichen Anforderungen an Infrastruktur und die Abhängigkeit von einer Netzwerkverbindung stellen einen unverhältnismässigen Mehraufwand dar, der den Nutzen nicht rechtfertigt.

1.3 File-basiert (serverlos)

Eine Datenbank wird lokal als Ressource in Form einer Datei dem Programm bereitgestellt und ist ohne äusserliche Einwirkungen persistent. Sie bietet eine leichtgewichtige Lösung für kleinere Anwendungen.

Vorteile

- Keine Notwendigkeit für externe Infrastruktur, wodurch die Einrichtung vereinfacht wird.
- Volle Offline-Funktionalität, da keine Verbindung zu externen Diensten erforderlich ist.
- Hohe Performanz durch direkte Anbindung an das Programm.

Nachteile

- Risiko eines Datenverlusts bei Fehlern oder Ausfällen des Anwendersystems.
- Eingeschränkte Skalierbarkeit bei wachsenden Datenmengen.

Fazit Die file-basierte und serverlose Datenbank wurde als die am besten geeignetste Option identifiziert. Sie erfüllt die Anforderungen eines kleinen Systems ohne die Notwendigkeit zusätzlicher Infrastruktur und ermöglicht eine einfache und schnelle Einrichtung und Verwendung. Die Verantwortung für die Datenbank wird auf den Anwender übertragen, was den Verwaltungsaufwand reduziert. Zudem ist so generell ein Offline-Betrieb möglich (im Kontext des Monitorings).

2. Schnittstelle CLI ↔ Daemon

Für die Kommunikation zwischen der CLI (Command Line Interface) und dem Daemon ist eine geeignete Schnittstelle erforderlich, um Anweisungen des Anwenders effizient und zuverlässig zu übertragen. Im Folgenden werden die in Betracht gezogenen Varianten analysiert.

2.1 Inter-Prozess-Kommunikation (IPC)

Diese Variante nutzt einen gemeinsamen Speicherbereich, beispielsweise ein geteiltes Verzeichnis, um Dateien für die Kommunikation zwischen CLI und Daemon auszutauschen. Die CLI schreibt Anweisungen in Dateien, die der Daemon anschliessend ausliest und verarbeitet.

Vorteile

- Einfach zu implementieren bei lokalem Einsatz auf demselben Rechner.
- Keine Netzwerkkonfiguration erforderlich.

Nachteile

- Erfordert die Definition eines eigenen Dateiformats und entsprechender Parser.
- CLI und Daemon müssen auf demselben System laufen, was die Flexibilität einschränkt.
- Es besteht das Risiko von Konflikten beim gleichzeitigen Schreiben und Lesen von Dateien, was zusätzliche Synchronisationsmechanismen erfordert.

Fazit Diese Option wurde aufgrund ihrer mangelnden Flexibilität und Erweiterbarkeit ausgeschlossen. Insbesondere die lokale Bindung zwischen CLI und Daemon sowie die fehlende Standardisierung und Robustheit der Kommunikation sprechen gegen diese Variante. Für zukünftige Anforderungen, wie die Kommunikation über ein Netzwerk, wäre sie ungeeignet.

2.2 Netzwerk-Sockets

Netzwerk-Sockets ermöglichen die Kommunikation zwischen Prozessen durch das Senden und Empfangen binärer Daten über spezifische Ports. Diese Methode kann sowohl lokal als auch über das Netzwerk verwendet werden.

Vorteile

- Einfach zu verwenden und flexibel steuerbar.

- Unterstützt die Kommunikation zwischen CLI und Daemon auf getrennten Systemen über das Netzwerk.
- In Java durch das `java.net`-Package nativ verfügbar.

Nachteile

- Nachrichtenformat muss individuell definiert und implementiert werden, was zusätzlichen Entwicklungsaufwand verursacht.

Fazit Obwohl Netzwerk-Sockets eine flexible und leistungsfähige Lösung darstellen, wurde diese Variante zugunsten der REST-Schnittstelle verworfen. Die REST-Implementierung erscheint aufgrund ihrer Standardisierung und einfacher Handhabung im Kontext des Projekts effizienter. Zudem entfällt der Aufwand für die Definition und das Parsen eines eigenen Nachrichtenformats.

2.3 REST-Schnittstelle

Die REST-Schnittstelle ermöglicht den Austausch von Ressourcen zwischen Client und Server über das HTTP-Protokoll. Der Austausch erfolgt im Rahmen definierter Formate (Content-Types) und kann über das Netzwerk abgewickelt werden.

Vorteile

- Standardisierte Schnittstelle, die durch zahlreiche Frameworks unterstützt wird.
- Hohe Robustheit, da REST auf etablierten HTTP-Standards basiert.
- Ermöglicht die Ausführung von CLI und Daemon auf getrennten Systemen, was eine flexible Architektur fördert.

Nachteile

- Erfordert die Implementierung eines Webservices, was zusätzlichen Entwicklungsaufwand bedeutet.

Fazit Die REST-Schnittstelle wurde als die bevorzugte Option gewählt, da sie eine klare und standardisierte technische Kommunikation ermöglicht. Sie ist stabil, flexibel und erweiterbar, was besonders wichtig für den langfristigen Betrieb und mögliche zukünftige Anforderungen, wie die Systemtrennung von CLI und Daemon, ist. Zudem ist die Implementierung dank vorhandener Frameworks verhältnismässig einfach und effizient.

3.1.1.2 Technologieentscheid

In diesem Abschnitt werden die Überlegungen zu den Technologien für die verschiedenen Systemkomponenten dargestellt. Basierend auf einer Analyse der Vor- und Nachteile wurde eine Entscheidung für die geeignetsten Technologien getroffen.

1. Datenbank

Die Auswahl einer passenden Datenbank ist entscheidend für die Persistenz und Verarbeitung der Anwendungsdaten. Dabei wurde zunächst die grundsätzliche Entscheidung zwischen **NoSQL** und **SQL** analysiert.

1.1 NoSQL

NoSQL-Datenbanken verwenden flexible Datenmodelle wie Dokumente, Key-Value oder Graphen und sind für hohe Skalierbarkeit sowie grosse Datenmengen ausgelegt.

Vorteile

- Hohe Flexibilität: Anpassbare Datenstrukturen ohne festes Schema.
- Einfache horizontale Skalierbarkeit: Gut geeignet für grosse Datenmengen.
- Spezialisierung: Verschiedene Modelle für spezifische Anwendungsfälle.

Nachteile

- Fehlende Standardisierung: Unterschiedliche Abfragesprachen und Protokolle je nach Implementierung.
- Eingeschränkte Transaktionssicherheit: ACID-Eigenschaften werden häufig zugunsten der Performance vernachlässigt.
- Höhere Komplexität: Abfragen sind weniger intuitiv als in SQL-basierten Systemen.

Fazit Die Vorteile von NoSQL-Datenbanken bringen für dieses Projekt keinen nennenswerten Mehrwert, da dieser Anwendungsfall keine besonderen Anforderungen an Skalierbarkeit oder unstrukturierte Daten stellt. Auch aufgrund mangelnder Erfahrung im Team mit NoSQL-Systemen wurde diese Option verworfen.

1.2 SQL

SQL-Datenbanken basieren auf einem relationalen Modell mit festen Schemata. Sie verwenden die standardisierte Abfragesprache SQL und sind besonders geeignet für datenintensive Anwendungen mit hohen Anforderungen an Konsistenz.

Vorteile

- Standardisierung: Einheitliche Abfragesprache und breite Unterstützung.
- Datenkonsistenz: Verlässliche Transaktionen dank ACID-Eigenschaften.
- Leistungsstarke Abfragemöglichkeiten: Geeignet für komplexe relationale Analysen.

Nachteile

- Feste Schemata: Weniger flexibel bei Änderungen in der Datenstruktur.
- Begrenzte horizontale Skalierbarkeit: Schwieriger als bei NoSQL umzusetzen.
- Suboptimale Performance bei grossen unstrukturierten Datenmengen.

Fazit Die Wahl fiel auf SQL-Datenbanken, da diese den Anforderungen an Konsistenz und relationale Abfragemöglichkeiten optimal entsprechen. Alle Teammitglieder verfügen über umfassende Erfahrung mit SQL, was die Implementierung erleichtert. Die Nachteile, wie begrenzte Skalierbarkeit, sind für dieses Projekt vernachlässigbar.

1.2.1 H2

H2 ist eine leichtgewichtige, Java-basierte relationale Datenbank, die sowohl eingebettet als auch im Servermodus betrieben werden kann.

Vorteile

- Integration in Java: Optimiert für Java-Anwendungen und unterstützt JDBC.
- Vielseitigkeit: Kann als eingebettete (bzw. file-basierte) oder serverbasierte Datenbank eingesetzt werden.
- Erweiterte Funktionen: Unterstützung für in-memory-Datenbanken und Multi-Threaded-Betrieb.

Nachteile

- Eingeschränkte Eignung für grosse Workloads: Nicht für datenintensive Anwendungen ausgelegt.
- Java-Abhängigkeit: Begrenzte Unterstützung in nicht-Java-Umgebungen.
- Persistenzprobleme: Daten können im Speicher verloren gehen, wenn nicht korrekt gesichert.

Fazit H2 wurde ausgeschlossen, da es nicht für grosse Datenmengen ausgelegt ist und eine zu starke Abhängigkeit von Java besteht. Zudem erfüllt SQLite unsere Anforderungen besser.

1.2.2 SQLite

SQLite ist eine serverlose, eingebettete relationale Datenbank, die Daten in einer einzigen Datei speichert und keine separate Konfiguration erfordert.

Vorteile

- Einfachheit: Keine zusätzliche Konfiguration oder separate Server erforderlich.
- Portabilität: Daten sind in einer Datei gespeichert, was die Migration erleichtert.
- Ressourcenschonend: Geringe Anforderungen an Speicher und Rechenleistung.

Nachteile

- Begrenzte Skalierbarkeit: Nicht geeignet für Anwendungen mit hohem gleichzeitigen Datenzugriff. (Single-Writer-Prinzip)
- Eingeschränkte Funktionalität: Unterstützt nicht alle erweiterten SQL-Funktionen.
- Performance-Einbussen bei grossen Datenmengen.

Fazit SQLite wurde ausgewählt, da es unseren Anforderungen an Einfachheit und lokale Persistenz optimal entspricht. Die Nachteile sind für dieses Projekt vernachlässigbar.

2. CLI/Daemon

Für die Entwicklung von CLI und Daemon wurde eine einheitliche Technologie gewählt, um den Entwicklungsaufwand zu reduzieren und die Wiederverwendbarkeit von Code zu fördern.

2.1 Python

Python ist eine einfach zu erlernende, vielseitige Programmiersprache, die insbesondere für datengetriebene Anwendungen und schnelle Entwicklung geeignet ist.

Vorteile

- Einfache Syntax: Kürzere Entwicklungszeit und schnelle Einarbeitung.
- Starke KI-Unterstützung: Umfangreiche Bibliotheken wie **transformers** und **torch**.
- Flexibilität: Besonders geeignet für schnelle Iterationen und Skripte.

Nachteile

- Geringere Performance: Nicht ideal für rechenintensive Anwendungen.
- Dynamische Typisierung: Kann bei grossen Projekten zu Wartungsproblemen führen.
- Distributionsprobleme: Erfordert oft spezielle Umgebungen oder Pakete.

Fazit Obwohl Python starke Vorteile in Bezug auf KI-Bibliotheken bietet, wurde diese Option aufgrund der geringeren Performanz und fehlenden Erfahrung im Team verworfen. Java erwies sich als besser geeignet für unsere Anforderungen.

2.2 Java

Java ist eine objektorientierte, plattformunabhängige Sprache, die sich durch Stabilität, Performance und breite Unterstützung auszeichnet.

Vorteile

- Hohe Performance: Geeignet für rechenintensive Anwendungen.
- Strukturierte Entwicklung: Klare Organisation und starke Typisierung fördern die Wartbarkeit.
- Integration: Gute Unterstützung für APIs und Frameworks wie Spring.

Nachteile

- Höhere Komplexität: Erfordert mehr Boilerplate-Code und strikte Typisierung.
- Eingeschränkte Bibliotheken für KI: Weniger umfangreiche KI- und ML-Bibliotheken im Vergleich zu Python.
- Langsamere Prototypenentwicklung: Weniger geeignet für schnelle Iterationen.

Fazit Java wurde gewählt, da alle Teammitglieder bereits über umfassende Erfahrung verfügen und die Sprache eine hohe Performanz und Stabilität bietet. Die Einschränkungen hinsichtlich KI-Integration waren für dieses Projekt zu wenig gewichtig. Frameworks wie Spring erleichtern zudem die Implementierung der CLI/Daemon-Architektur.

2.2.1 Spring

Spring ist ein umfangreiches Framework für die Entwicklung von Java-Anwendungen, das eine Vielzahl von Funktionen und Bibliotheken bereitstellt. Das Framework wurde bereits von allen Teammitgliedern in früheren Projekten erfolgreich eingesetzt. Deswegen wurde es auch für dieses Projekt als geeignet und grosse Unterstützung angesehen. Im Rahmen dieses Systems bietet Spring insbesondere Unterstützung für die Implementierung der REST-Schnittstelle und der Datenbankbindung. Ausserdem bietet es durch Spring Shell ein gutes Gerüst für die Implementierung der CLI.

3. KI-Modell

Die Auswahl des KI-Modells wird ausführlich im Abschnitt KI Fallstudie behandelt.

3.1.2 Lösungsarchitektur

Das System besteht aus zwei Hauptkomponenten, welche als eigene Laufzeiteinheiten ausgeführt werden. Die CLI wird ad hoc gestartet und dient als Benutzerschnittstelle. Der Daemon läuft nach dem Start im Hintergrund und führt periodisch Operationen aus. Ausserdem stellt er der CLI eine REST-Schnittstelle zur Verfügung. Ein Datenbanksystem speichert die Applikationsdaten. Diese Architektur ermöglicht eine klare Trennung von Benutzerschnittstelle und Geschäftslogik. Jegliche Interaktionen mit Dateisystem, Datenbank und KI-Modell werden über den Daemon abgewickelt.

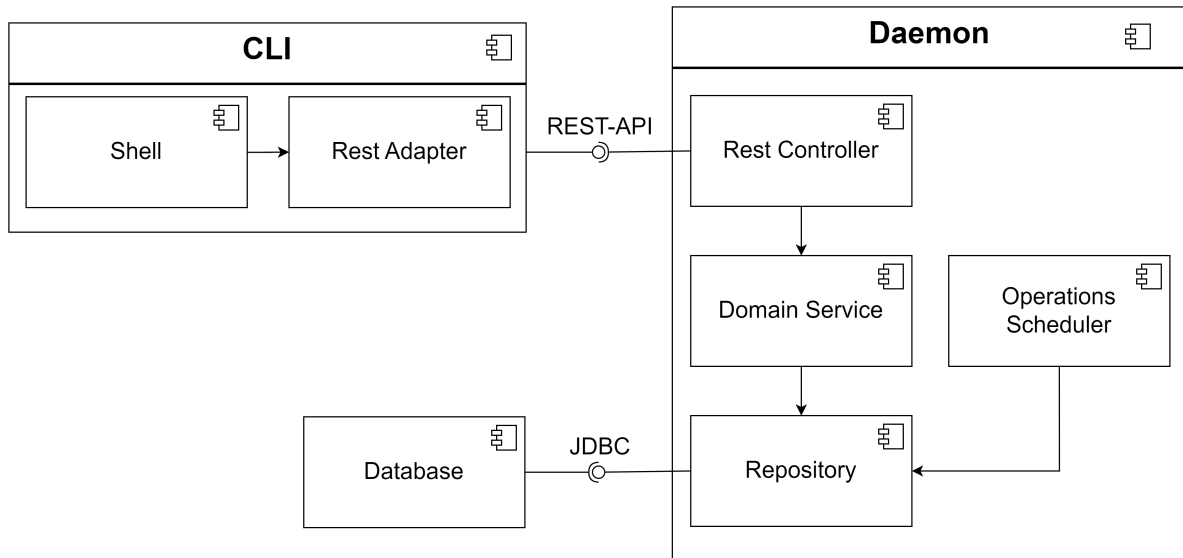


Abbildung 3.1: UML Komponentendiagramm

3.1.2.1 Domänen

Die im Diagramm dargestellten Subkomponenten kapseln Funktionalitäten und sind in der Regel in eigenen Klassen implementiert. Es können mehrere Instanzen dieser Subkomponenten implementiert werden und nach fachlichen Aspekten, sogenannten Domänen, aufgeteilt werden.

Am Beispiel einer Snapshot-Domäne würden dann im Backend ein Snapshot-(Rest)-Controller, ein Snapshot-(Domain)-Service und ein Snapshot-Repository existieren. D.h. eine Kapselung findet in technischer und fachlicher Hinsicht statt. Eine komplette Abkapselung ist jedoch in der Praxis schwierig und soll lediglich angestrebt werden.

3.1.2.2 Shell

Die Shell stellt eine Kommandozeilen-Benutzerschnittstelle zur Verfügung. Sie kapselt die Definition von Kommandos und deren Parameter. Ausserdem übernimmt sie das Parsen der Benutzereingaben, clientseitige Validierung sowie Exception-Handling.

3.1.2.3 Rest Adapter

Der Rest Adapter kapselt Funktionen zur Interaktion mit der REST-Schnittstelle.

3.1.2.4 Rest Controller

Der Rest Controller implementiert die Endpunkte der REST-Schnittstelle gemäss API-Spezifikation.

3.1.2.5 Domain Service

Der Domain Service implementiert die Applikations- bzw. Geschäftslogik. Dazu gehört serverseitige Validierung, Datenverarbeitung und -Persistierung.

3.1.2.6 Operations Scheduler

Der Operations Scheduler führt periodisch Operationen aus.

3.1.2.7 Repository

Das Repository stellt den Zugang zur Persistenzschicht zur Verfügung. Dabei handelt es sich um Create, Read, Update, Delete (CRUD) Operationen in Form von Datenbankqueries.

3.1.2.8 Technologien

	Technologien (primär)
Shell	Java, Spring (Shell)
Rest Adapter	Java, Spring (Web)
Rest Controller	Java, Spring (Web)
Domain Service	Java, Spring
Operations Scheduler	Java, Spring
Repository	Java, Spring (Data)
Database	SQLite

Tabelle 3.1: Technologien der Komponenten

3.1.3 Realisierung

Die Realisierung ist die Umsetzung der Anforderungen mittels der Lösungsarchitektur. Hauptbaublöcke sind Java-Klassen. Geteilte Klassen sind in einer eigenen Bibliothek (**lib**) ausgelagert. Die Klassendiagramme für die CLI, den Daemon und die Lib sind im Anhang unter Klassendiagramm CLI, Klassendiagramm Daemon und Klassendiagramm Lib zu finden.

3.1.3.1 Frontend (CLI)

Beispiel Komponenteninteraktion Das folgende Sequenzdiagramm zeigt eine typische Interaktion zwischen Implementationen von Subkomponenten gemäss der Lösungsarchitektur. In diesem Fall handelt es sich um das **inspect** Kommando (siehe Benutzerhandbuch). Die Klasse **InspectCommand** ist Teil der Shell (siehe Shell). Der **DaemonAdapter** ist die Implementation des **RestAdapters** (siehe Rest Adapter).

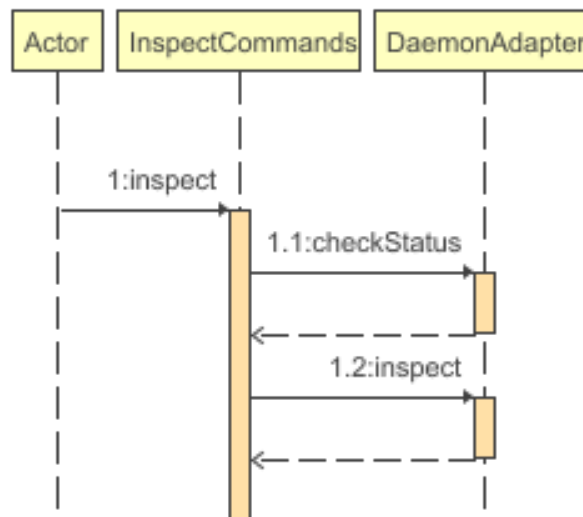


Abbildung 3.2: Sequenzdiagramm Inspect Command

Dieses Muster ist für alle Kommandos in der Shell identisch, daher wird dieser Abschnitt in prägnanter Form gehalten.

3.1.3.2 Backend (Daemon)

Beispiel Komponenteninteraktion Das folgende Sequenzdiagramm zeigt ebenfalls eine typische Interaktion zwischen Implementationen von Subkomponenten gemäss der Lösungsarchitektur. Man sieht innerhalb der Monitoring-Domäne den Controller (**MonitoringController**), den Service (**MonitoringDomainService**) und das Repository (**MonitoredPathRepository**). Des Weiteren sieht man eine Utility-Klasse (**ControllerUtils**) und eine Entitätsklasse (**MonitoredPath**) und zur Veranschaulichung (in rot) die Superklasse vom Spring-Framework (**CrudRepository**).

In diesem Beispiel wird ein überwachter Pfad (**MonitoredPath**) erstellt:

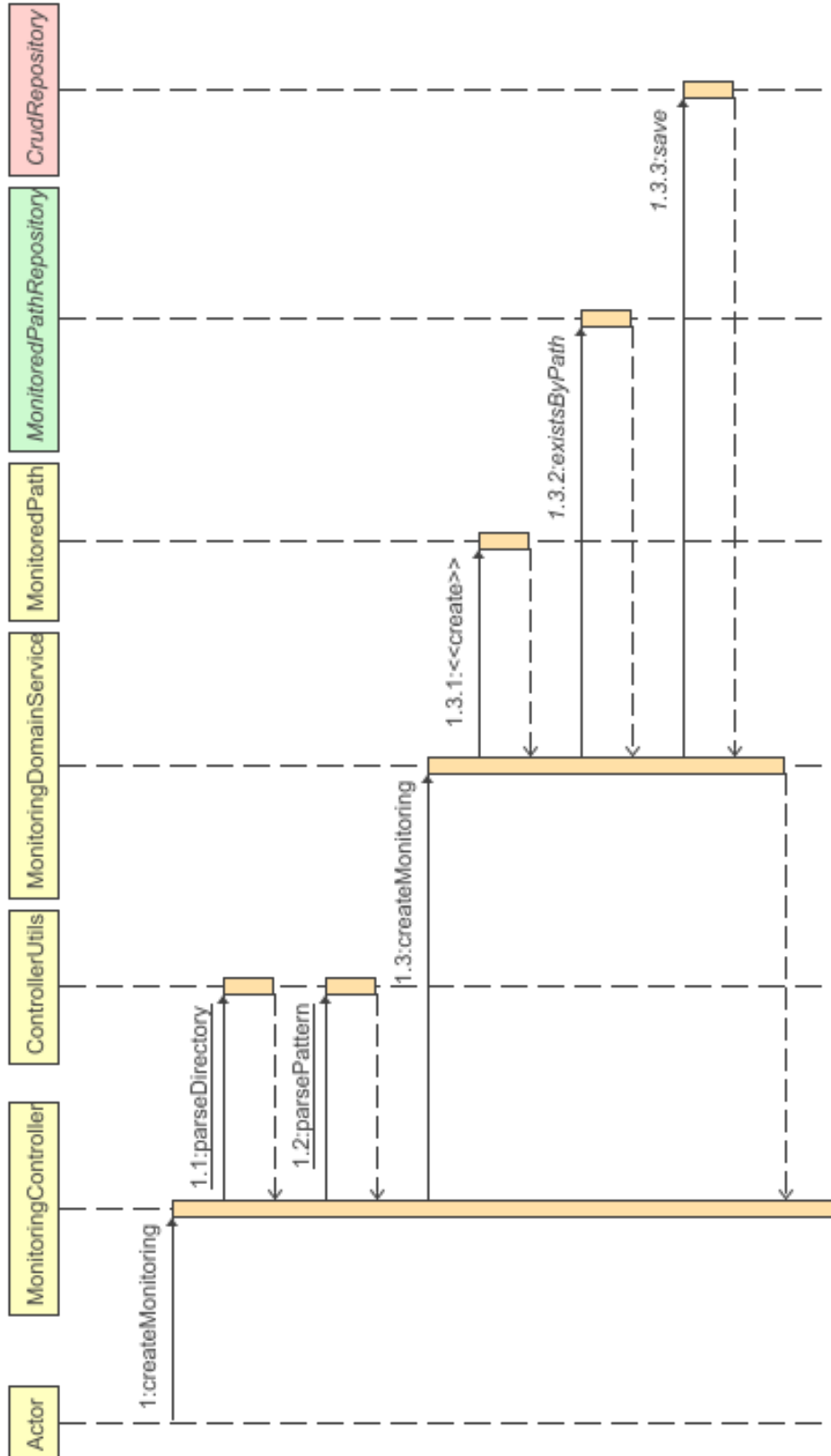


Abbildung 3.3: Sequenzdiagramm Monitoring Controller

Monitoring

Der Monitoring Scheduler ist die Implementation des Operations Schedulers (siehe Lösungsarchitektur). Er speichert periodisch Snapshots von überwachten Pfaden in der Datenbank (1.1)¹. Für jeden überwachten Pfad wird ein Snapshot (1.1.3), d.h. ein Hashbaum, erstellt (1.1.4).

Für jeden Knoten des Baumes werden die Hashes mit dem letzten Snapshot verglichen und ein Flag gesetzt, falls sich der Knoten geändert hat. Falls ein Knoten nicht mehr existiert, wird dafür ein Flag im alten Snapshot gesetzt. Diese zwei Flags sind die einzigen Felder, die auf Informationen vom Vorgänger (1.1.1, 1.1.2) bzw. Nachfolger zurückgreifen, alle anderen Felder stellen Informationen des aktuellen Snapshots im Sinne einer Momentaufnahme dar.

Der Intervall für die Periodizität der Snapshot-Erstellung ist konfigurierbar und kann z.B. via Programmargument oder Konfigurationsdatei gesetzt (mit Key `monitoring-scheduler.snapshot.interval`) werden. Standardmässig ist der Wert auf 1 Stunde gesetzt.

¹Diese Nummerierung bezieht sich auf das nachfolgende Sequenzdiagramm

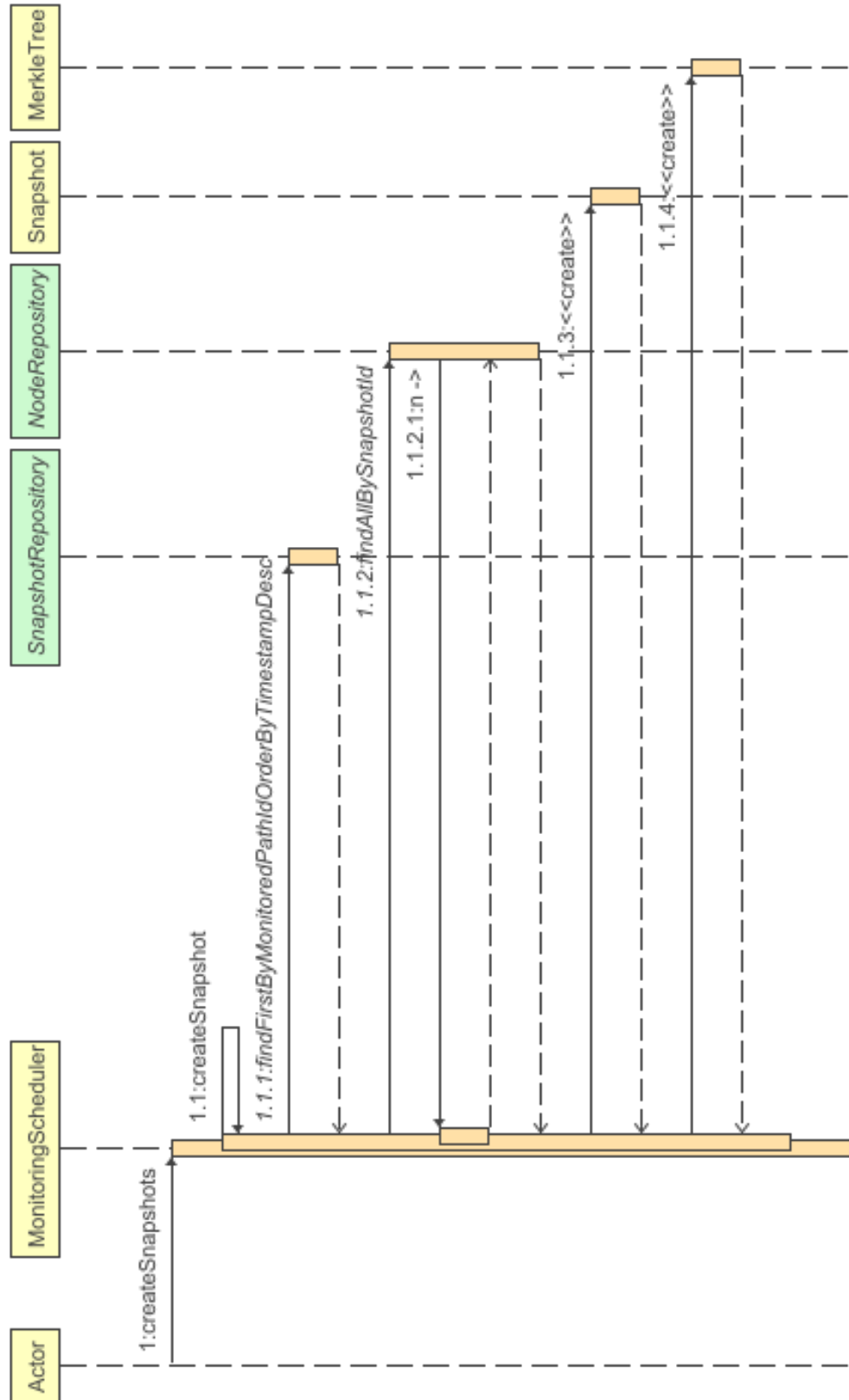


Abbildung 3.4: Sequenzdiagramm Monitoring Scheduler

Optimierungen

In den folgenden zwei Szenarien wurde geprüft, ob die Implementierung eines **Caching-Mechanismus** sinnvoll ist. Basierend auf den analysierten Aspekten wurde jedoch entschieden, darauf zu verzichten. Die Entscheidungsgrundlage wird im Folgenden erläutert:

1. Für jeden **MonitoredPath** könnte der zuletzt erzeugte Merkle-Tree zwischengespeichert werden, um diesen beim nächsten Snapshot direkt mit dem neuen Baum vergleichen zu können, ohne ihn erneut aus der Datenbank laden zu müssen.
 - Ein Caching-Ansatz ist nur dann sinnvoll, wenn dieselben Daten mehrfach genutzt werden. Da jedoch der Merkle-Tree des letzten Snapshots lediglich einmal (beim Vergleich mit dem neuen Baum) verwendet wird, ergibt sich hieraus kein signifikanter Nutzen.
 - Der Leistungsgewinn durch Caching wäre im Vergleich zum Laden aus der Datenbank marginal. Die Datenbank ist lokal angebunden, und die Abfrage weist eine geringe Komplexität auf, sodass ein effizienter Zugriff bereits gewährleistet ist.
2. Für jeden **MonitoredPath** könnte eine geordnete Liste der Änderungen jedes Snapshots im Speicher gehalten werden, um diese direkt zurückgeben zu können, ohne auf die Datenbank zugreifen zu müssen.
 - Auch in diesem Szenario wird keine nennenswerte Steigerung der Performance erwartet. Selbst wenn die Daten bereits im Speicher verfügbar wären, erfordert die Erstellung der zusammengefassten Antwort zusätzliche Logik. Diese kann ebenso effizient durch eine Abfrage auf die lokale Datenbank umgesetzt werden, wodurch sich kein signifikanter Vorteil ergibt.

3.1.3.3 Merkle-Tree (Hashbaum)

Ein Merkle-Tree ist eine binäre Baumstruktur, die zur effizienten Überprüfung von Datenintegrität verwendet wird. Er eignet sich für das Vergleichen von Baumstrukturen, wie sie in Dateisystemen vorkommen.

Bei der Erstellung eines Snapshots eines überwachten Pfades wird ein eben solcher Baum erstellt. Er besteht aus Knoten, die Hashes von Dateien oder Unterverzeichnissen enthalten.

Folgender Pseudocode (stark vereinfacht) zeigt das Erstellen eines Merkle-Trees für einen Snapshot eines überwachten Pfades.

```
buildMerkleTree(path, snapshot):
    1. ueberpruefe, ob der Pfad gueltig ist. Falls nicht:
       Abbrechen.
    2. Erstelle Wurzel-Knoten fuer das Verzeichnis.
    3. Rufe BuildTreeRecursively(Wurzel, snapshot,
       Verzeichnisinhalte) auf.

buildTreeRecursively(parent, snapshot, files):
    1. Initialisiere eine Liste fuer Kind-Knoten.
    2. Fuer jede Datei im Verzeichnis:
       a. Wenn Datei ein Unterverzeichnis ist:
          - Erstelle Knoten fuer das Verzeichnis.
          - Rufe BuildTreeRecursively fuer das
            Unterverzeichnis auf.
       b. Wenn Datei gueltig ist (entspricht Filter):
          - Erstelle Knoten fuer die Datei.
       c. Fuege den neuen Knoten zur Kinderliste hinzu.
    3. Kombiniere Hashes aller Kinder und speichere im
       Elternknoten.
```

3.1.3.4 Komplexität

Unser System ist von Grund auf auf Effizienz und Performanz ausgelegt, der Hashbaum als zentrale Datenstruktur ermöglicht dies. Die Datenmenge mit der ein Benutzer arbeitet, kann sehr gross werden, daher ist es wichtig, dass die Skalierbarkeit des Systems gewährleistet ist. Eine Analyse der Komplexität der verschiedenen Operationen des Merkle-Trees ist daher angebracht.

Jeder Ordner und jede Datei wird als Knoten im Hashbaum repräsentiert. Die Wurzel des Hashbaums repräsentiert den zu überwachenden Pfad. Dateien sind stets Blätter.

Folgend ist n die Anzahl der Knoten im Verzeichnis des zu überwachenden Pfades.

Konstruktion Der Baualgorithmus besucht rekursiv alle Dateien und Ordner im zu überwachenden Pfad, was einer Komplexität von $O(n)$ für das Traversieren entspricht. Das Hashen eines Ordners mit k Kindern setzt eine Iteration über alle Kinder voraus, was einer Komplexität von $O(k)$ entspricht. Das Hashen eines Blattes (Datei) ist abhängig von der Grösse s der Datei und hat eine Komplexität von $O(s)$.

Die Gesamtkomplexität des Bauens des Hashbaums beträgt also $O(n \cdot s)$. Um die Berechnung einfach zu halten, wird hierbei s als konstant angenommen. In der Praxis variiert

die Grösse der Dateien jedoch stark, weshalb die Komplexität in der Realität ebenfalls variiert. k ist implizit in n enthalten.

Linearisierung Es gibt Fälle, in denen der Hashbaum linearisiert wird, um z.B. die Java Stream API zu verwenden oder direkt alle Knoten miteinander zu persistieren. Es werden für alle Knoten die Referenzen in einer Liste gespeichert, was einer Komplexität von $\mathcal{O}(n)$ entspricht.

3.1.3.5 Datenbank

Produktiv wird eine SQLite-Datenbank verwendet, welche lokal auf dem System des Benutzers in Form einer einzigen *.db-Datei gespeichert wird. Für die automatisierten Tests wurde eine H2-In-Memory-Datenbank verwendet, welche für jeden Test mit dem gleichen Schema, das auch für die produktive Datenbank verwendet wird, initialisiert wird.

Modell

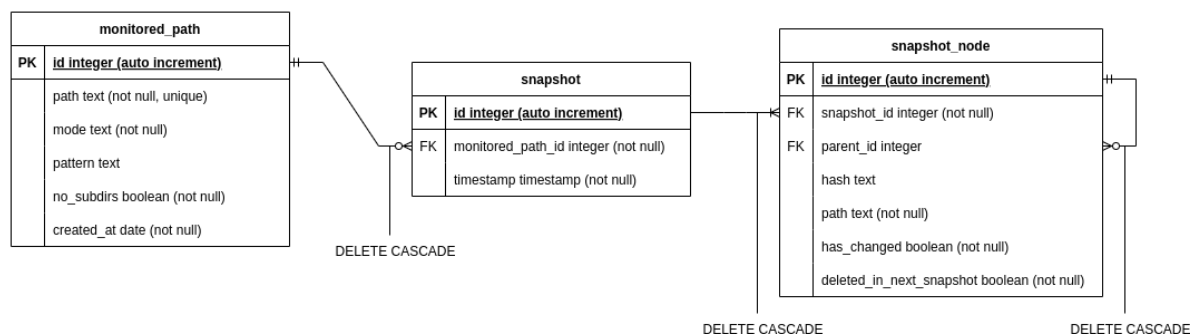


Abbildung 3.5: Entity Relationship Diagram

monitored_path

In dieser Tabelle werden die zu überwachenden Pfade mit den dazugehörigen Such-Optionen gespeichert.

Attribute:

- **id** (INTEGER): Eindeutige ID des überwachten Pfades.
- **path** (TEXT): Absoluter Pfad des überwachten Verzeichnisses.
- **mode** (TEXT): Gibt an, mit welchem Modus der Pfad durchsucht werden soll.
- **pattern** (TEXT): Regex-Pattern für Dateien, nach denen im Verzeichnis des Pfads gesucht werden soll.
- **no_subdirs** (BOOLEAN): Gibt an, ob Unterverzeichnisse durchsucht werden sollen.
- **created_at** (DATE): Zeitpunkt der Erstellung des überwachten Pfades.

snapshot

In dieser Tabelle werden die Snapshots der überwachten Pfade gespeichert, welche periodisch mit einem definierten Intervall erstellt werden.

Attribute:

- **id** (INTEGER): Eindeutige ID des Snapshots.
- **monitored_path_id** (INTEGER): Fremdschlüssel auf den überwachten Pfad. Falls der überwachte Pfad gelöscht wird, werden auch alle zugehörigen Snapshots gelöscht.

- **timestamp** (TIMESTAMP): Zeitpunkt der Erstellung des Snapshots.

snapshot_node

In dieser Tabelle werden die Knoten des Merkle-Trees gespeichert, welche die Dateien und Verzeichnisse des überwachten Pfades repräsentieren. Durch die Natur von relationalen Datenbanken wird der Baum linearisiert gespeichert und die Beziehungen durch Fremdschlüssel auf die gleiche Entität (Parent-Knoten) rekursiv abgebildet.

Attribute:

- **id** (INTEGER): Eindeutige ID des Knotens.
- **snapshot_id** (INTEGER): Fremdschlüssel auf den Snapshot, zu dem der Knoten gehört. Falls der Snapshot gelöscht wird, werden auch alle zugehörigen Knoten gelöscht.
- **parent_id** (INTEGER): Fremdschlüssel auf den Parent-Knoten. Falls der Parent-Knoten gelöscht wird, wird auch der Kind-Knoten gelöscht.
- **hash** (TEXT): Hash des Elements, welches von diesem Knoten abgebildet wird.
- **path** (TEXT): Absoluter Pfad des Elements, welches von diesem Knoten abgebildet wird.
- **has_changed** (BOOLEAN): Gibt an, ob sich das Element, welches von diesem Knoten abgebildet wird, seit dem letzten Snapshot geändert hat oder neu hinzugefügt wurde.
- **deleted_in_next_snapshot** (BOOLEAN): Gibt an, ob das Element, welches von diesem Knoten abgebildet wird, im nächsten Snapshot nicht mehr existiert.

Indexing

Die Entscheidung über die Sinnhaftigkeit eines SQL-Indexes für bestimmte Attribute der Datenbank basiert auf den folgenden Kriterien:

Index sinnvoll:

- Das Attribut weist eine hohe Kardinalität (viele unterschiedliche Werte) auf.
- Das Attribut enthält wenige oder keine Null-Werte.
- Das Attribut wird häufig in SQL WHERE/JOIN-Klauseln verwendet.

Index nicht sinnvoll:

- Die Tabelle ist von geringer Grösse.
- Das Attribut wird häufig manipuliert.

Auf Basis dieser Überlegungen wurden die folgenden Attribute zur Indexierung ausgewählt:

- `snapshot_node.snapshot_id`
- `snapshot_node.has_changed`
- `snapshot_node.deleted_in_next_snapshot`

3.1.3.6 REST-Schnittstelle

Siehe API-Dokumentation im Anhang.

3.1.3.7 KI-Anbindung

Architektur und Funktionsweise Die Anbindung an KI-Modelle erfolgt über den *InspectController*, der die HTTP-Anfragen der CLI entgegennimmt. Dieser liest die zu analysierende Datei ein und ruft eine Implementierung des *InspectionService* Interfaces auf, welche für die Kommunikation mit einem spezifischen KI-Modell zuständig ist.

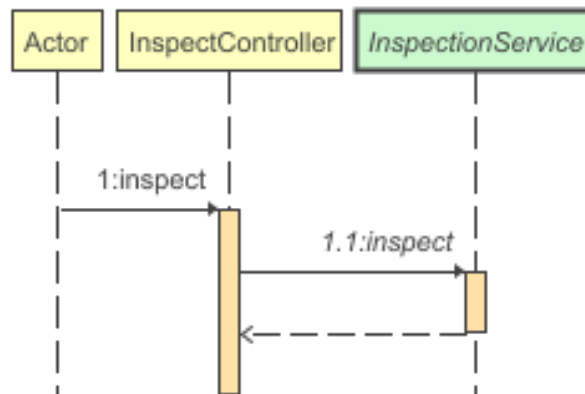


Abbildung 3.6: Sequenzdiagramm InspectController

Im Rahmen dieses Projekts wurde GPT-4o mini als KI-Modell verwendet (Siehe KI Fallstudie). Für dieses Modell wurde der *GPTInspectionService* implementiert, welcher die Anfrage an die OpenAI-API sendet und die Antwort verarbeitet.

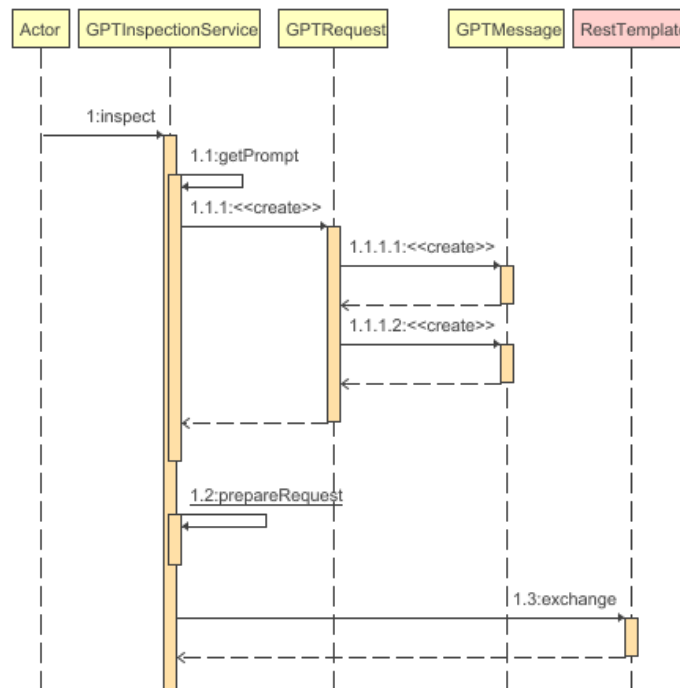


Abbildung 3.7: Sequenzdiagramm GPTInspectionService

OpenAI Schnittstelle Die Kommunikation mit dem KI-Modell basiert auf REST-API-Calls. Um eine Anfrage an die OpenAI-API zu senden, wird ein HTTP-POST-Request an <https://api.openai.com/v1/chat/completions> gesendet. Die Authentifizierung erfolgt über einen API-Key, der im Authorization-Header mitgegeben wird. Dieser wird im Vorfeld aus der Umgebungsvariable `OPENAI_API_KEY` ausgelesen. Die Anfrage ist wie folgt strukturiert:

Listing 3.1: OpenAI Anfrage

```
1  {
2      "model": "gpt-4o-mini",
3      "messages": [
4          {
5              "role": "system",
6              "content": "<SYSTEM_PROMPT>"
7          },
8          {
9              "role": "user",
10             "content": "<userPrompt>"
11         }
12     ]
13 }
```

Diese Struktur wird durch die Klassen `GPTRquest` inkl. `GPTMessage` abgebildet (siehe Sequenzdiagramm `GPTInspectionService`).

Die Antwort der OpenAI Schnittstelle ist deutlich umfangreicher und enthält neben der eigentlichen Antwort auch Metadaten. Für die Analyse wird lediglich der Antworttext benötigt, welcher aus der JSON-Response extrahiert wird.

Listing 3.2: OpenAI Antwort

```
1  {
2      "choices": [
3          {
4              "message": {
5                  "content": "<Antwort>"
6              }
7          },
8          {}
9      ]
10 }
```

Prompt-Engineering Die Qualität der Antwort des KI-Modells hängt stark von der Qualität des System- und Benutzer-Prompt ab. Beide Prompts werden in derselben Anfrage an die OpenAI-API übergeben. Das System-Prompt ist fix und definiert den Kontext, in dem die Antwort generiert wird. Das Benutzer-Prompt beinhaltet den absoluten Pfad im Dateisystem sowie den gesamten Inhalt der zu analysierenden Datei.

You will receive the entire file content, including its name and path. These are typically log or cache files. Your task is to evaluate the file for harmful or problematic features. Pay close attention to the file's name and path as they may provide important context.

Respond concisely in ****a maximum of 10 sentences****, addressing the following:

1. ****Intended use****: Briefly describe the file's purpose based on its content, name, and path. If its content contains suspicious or harmful elements, describe them.
2. ****Assessment****: Classify the file as harmful, potentially harmful, or harmless.
3. ****Recommended to Wipe****: Suggest one of the following actions : No, Clear file, Delete file.
4. ****Additional recommendations****: If necessary, provide additional recommendations or actions.

Example:

Intended use:

[Describe the file's intended use in up to 10 sentences. Use up to 10 sentences to describe harmful or suspicious elements.]

Assessment:

[Harmful / Potentially harmful / Harmless]

Recommended to Wipe:

[No / Clear file / Delete file]

Additional recommendations:

[Provide additional recommendations or actions only if necessary.]

Diese System-Prompt beschreibt bereits die zu erwartende Struktur der Benutzer-Prompt, wie auch die zu erwartende Antwortstruktur. Dank dieser ausführlichen System-Prompt erfolgen die Antworten des Modells zuverlässig und konsistent (siehe KI Fallstudie).

Benutzer-Prompt:

```
file path: <absoluter Pfad>
file content:
<Inhalt der Datei>
```

Die Benutzer-Prompt ist bewusst kurz gehalten, damit sich das Modell auf die relevanten Informationen konzentrieren kann.

3.2 Prozesse

In diesem Kapitel wird auf die Verwendung der Projektmethode Scrum während des Projekts eingegangen. Dabei ist wichtig wie Scrum adaptiert wurde, um so gut wie möglich im vorgegebenen Projektrahmen zu funktionieren.

3.2.1 Rollen

Developer	Ganzes Team
Scrum Master	Luca Ammann
Product Owner	Luca Scherer

Tabelle 3.2: Scrum Rollen

Wir arbeiten alle als Entwickler und versuchen dabei die Arbeitslast mit Rücksicht auf die zusätzlichen Aufgaben der SM- und PO-Rollen so gleichmässig wie möglich zu verteilen.

Weitere Rollen

Betreuer	Dr. Simon Kramer
PM-Betreuer	Frank Helbling

Tabelle 3.3: Weitere Rollen

3.2.2 Sprintziele

Die Sprintziele wurden jeweils nach dem SMART-Prinzip definiert. Dazu folgende zwei Beispiele:

- Aufsetzen von lauffähiger CLI- und Daemon-Komponente und vollständige, sowie getestete Implementation der Search-Funktion (gemäss Testkonzept).
- Monitoring-Kommandos (gemäss Spezifikation) für das Persistieren und Auflisten der Dateipfade sind vollständig implementiert und getestet (gemäss Testkonzept).

3.2.3 Anwendung

3.2.3.1 Sprints

Die Sprints leben auf Stufe einer Group in Gitlab. Sie werden als Iterations bezeichnet. Issues können explizit zu einer Iteration zugewiesen werden.

3.2.3.2 Epics

Epics werden in Gitlab auf Stufe einer Group verwaltet. User Stories können zu einem Epic promoted werden. Epics sind grob beschrieben und kapseln mehrere kohärente Funktionen in Form von User Stories. Dazu folgendes Beispiel:

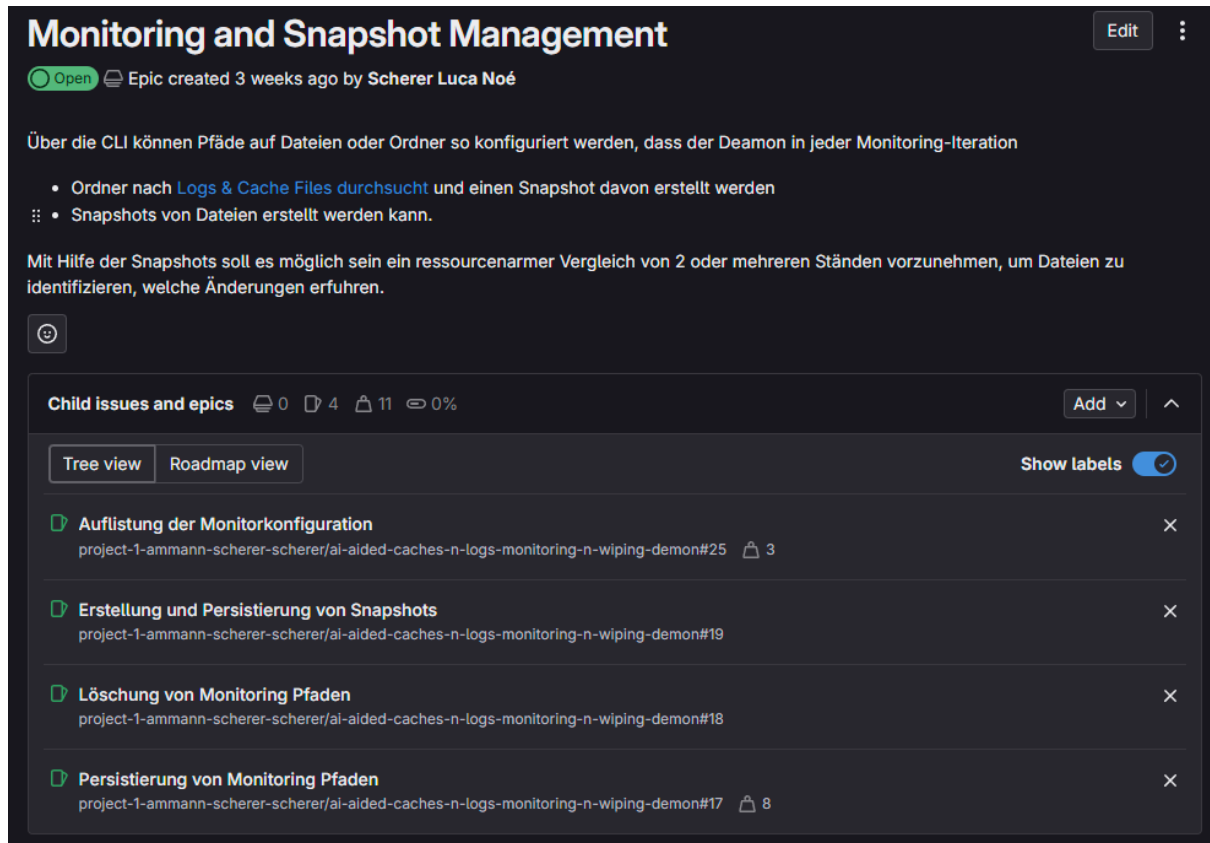


Abbildung 3.8: Epic Beispiel

3.2.3.3 User Stories

Eine User Story wird in Gitlab als Issue bezeichnet. Sie lebt auf der Stufe eines Projektes (Repository) und kann einem Epic zugeordnet sein. Sie kann ebenfalls sogenannte Tasks enthalten, welche kleinere Arbeiten der User Stories (meistens aus Entwickler-Sicht) repräsentieren. Jede User Story ist gemäss folgendem Schema dokumentiert:

Als <Benutzerrolle> möchte ich <Funktion>, sodass <Begründung/Benefit>. Dazu folgendes Beispiel:

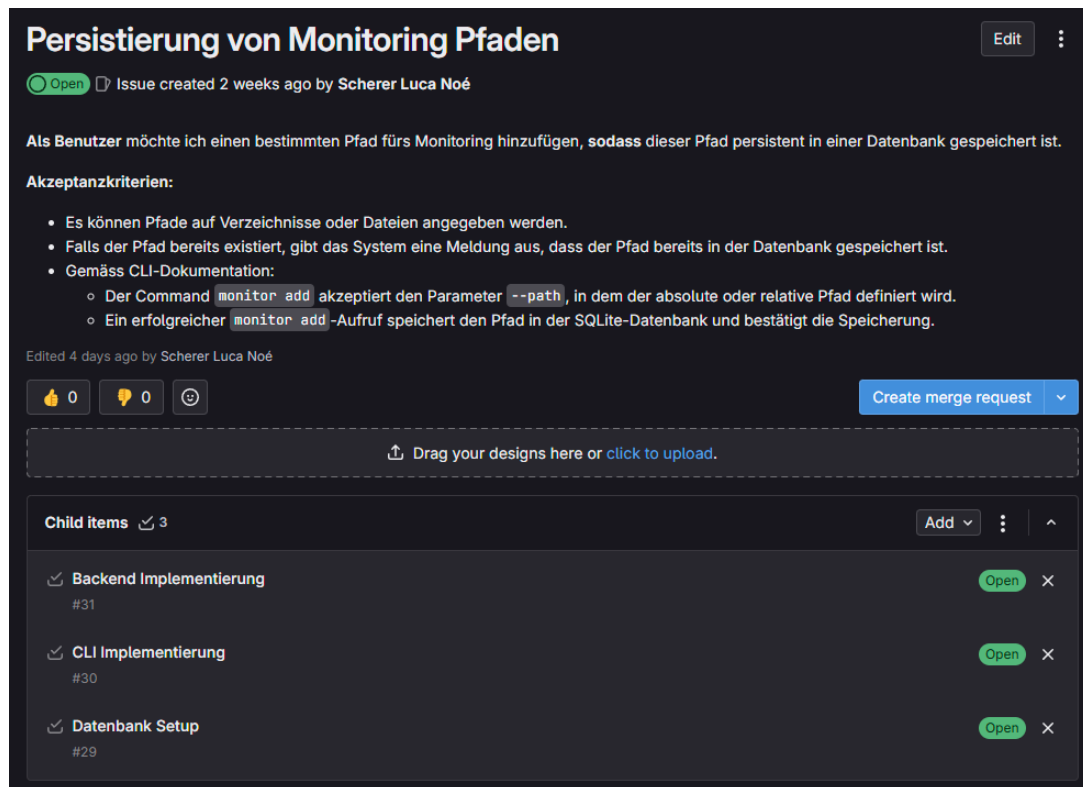


Abbildung 3.9: User-Story Beispiel

3.2.3.4 Definition of Ready

Damit eine User Story in ein Sprint Backlog gezogen werden darf, muss sie die Definition of Ready erfüllen. Wir haben diese, anlehnend an die INVEST-Kriterien, wie folgt definiert:

- **Independent:** if possible, a user story should not depend on other user stories.
- **Estimable:** The story must be possible to estimate, and an estimate in Story Points is provided.
- **Small:** The effort for the implementation should be manageable. A few hours to a max of a few days.
- **Testable:** Their successful implementation should be verifiable with objective criteria. Every story must have acceptance criteria.
- Does not violate a non-functional requirement (provided in the project report).

3.2.3.5 Definition of Done

Damit eine User Story in die Closed-Spalte gezogen werden darf, muss nachfolgende Definition of Done erreicht sein:

- Pull (Merge) Request is approved by another developer. Code Review is done.
- Acceptance criteria met.
- Functionality tested by someone who was not involved in the story as a developer.
- Documentation is done.

3.2.3.6 Taskboard

Das Taskboard lebt ebenfalls in unserem Gitlab Project. Es beinhaltet folgende Spalten: Open (Product Backlog), Sprint Backlog, Spalte pro Entwickler sowie die Spalte Closed. Dies ist die empfohlene Vorgehensweise, da Gitlab so automatisch versteht, welche Issues in welchen Sprint gezogen und auch darin erledigt wurden. Gitlab kann so beispielsweise, ohne jegliche Konfigurationen, Burndowncharts pro Iteration erstellen. Dazu folgendes Beispiel:

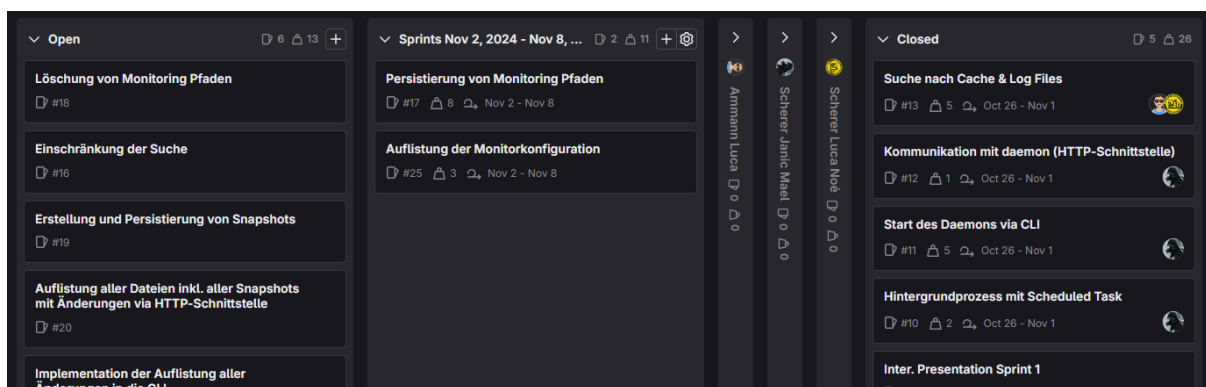


Abbildung 3.10: Taskboard Beispiel

3.2.4 Adaptionen

Die **Sprint-Dauer** haben wir auf **1 Woche** gesetzt, um so in Bezug auf Änderungen schneller reagieren zu können, da unsere Projektdauer eher kurz ist im Vergleich zu vielen anderen Scrum-Projekten. Ausserdem können wir so jede Woche den aktuellen Stand und vor allem die Neuerungen besprechen. Das hilft uns, unseren Fokus auf das Ziel und die Deadlines zu halten.

Das **Daily** können wir aufgrund unserer beschränkten Zeit, die wir am Projekt arbeiten können, nicht täglich durchführen. Dafür sind wir fast täglich per Chat im Austausch und können so kurzfristig, je nach Gebrauch, Besprechungen per Video-Call planen.

Die **Scrum-Zeremonien** (Review, Retro, Planning) führen wir en bloc **jeden Freitag** zum Sprint-Abschluss durch und starten sogleich den nächsten geplanten Sprint.

4 Evaluation und Tests

4.1 Manuelle End-2-End Tests

Wie durch das Testkonzept definiert, wurden die End-2-End Tests auf mehreren Endgeräten manuell durchgeführt.

4.1.1 Testfälle

ID	E2E-001
Beschreibung	Daemon starten und stoppen
Vorbedingungen	<ul style="list-style-type: none">• Daemon läuft nicht
Ablauf	<ol style="list-style-type: none">1. <code>ts status</code>2. Erwartet: <code>daemon is not running</code>3. <code>ts run</code>4. <code>ts status</code>5. Erwartet: <code>daemon is running</code>6. <code>ts kill</code>7. <code>ts status</code>8. Erwartet: <code>daemon is not running</code>
Erwartetes Ergebnis	Daemon kann gestartet und gestoppt werden.

Tabelle 4.1: Manueller End-2-End Test E2E-001

ID	E2E-002
Beschreibung	Autostart
Vorbedingungen	<ul style="list-style-type: none"> • Autostart Konfiguration gem. Installationshandbuch
Ablauf	<ol style="list-style-type: none"> 1. System neustarten 2. <code>ts status</code> 3. Erwartet: <code>daemon is running</code>
Erwartetes Ergebnis	Daemon wird via Autostart gestartet.

Tabelle 4.2: Manueller End-2-End Test E2E-002

ID	E2E-003
Beschreibung	Inspect
Vorbedingungen	<ul style="list-style-type: none"> • <code>OPENAI_API_KEY</code> Umgebungsvariable gesetzt gem. Installationshandbuch • Daemon läuft • Beispiel-Datei unter [PFAD] vorhanden
Ablauf	<ol style="list-style-type: none"> 1. <code>ts inspect [PFAD]</code> 2. Erwartet: Intended use: The file ... Assessment: ... Recommended to Wipe: ... Additional recommendations: ...
Erwartetes Ergebnis	Einschätzung und Empfehlung von der KI wird angezeigt.

Tabelle 4.3: Manueller End-2-End Test E2E-003

ID	E2E-004
Beschreibung	Pfad zum Überwachen hinzufügen und entfernen
Vorbedingungen	<ul style="list-style-type: none"> • Daemon läuft • Der [PFAD] wurde noch nicht hinzugefügt
Ablauf	<ol style="list-style-type: none"> 1. <code>ts monitor add [PFAD]</code> 2. Erwartet: Successfully added [PFAD] to the monitoring database. 3. <code>ts monitor list</code> 4. Erwartet: Der hinzugefügte Pfad wird aufgelistet. [ID] des Pfades notieren. 5. <code>ts monitor remove [ID]</code> 6. Erwartet: Successfully removed path with ID [ID] from the monitoring database. 7. <code>ts monitor list</code> 8. Erwartet: Der gelöschte Pfad wird nicht mehr aufgelistet.
Erwartetes Ergebnis	Ein Pfad kann ohne Fehler hinzugefügt und anschließend wieder entfernt werden.

Tabelle 4.4: Manueller End-2-End Test E2E-004

ID	E2E-005
Beschreibung	Pfad zum Überwachen mit Mode und ohne Unterordner hinzufügen.
Vorbedingungen	<ul style="list-style-type: none"> • Daemon läuft • Der [PFAD] wurde noch nicht hinzugefügt
Ablauf	<ol style="list-style-type: none"> 1. <code>ts monitor add [PFAD]</code> <code>--mode log --no-subdirs</code> 2. Erwartet: Successfully added [PFAD] to the monitoring database. 3. <code>ts monitor list</code> 4. Erwartet: Pfad wird mit mode=LOG und no- subdirs=true aufgelistet.
Erwartetes Ergebnis	Ein Pfad kann mit den gewünschten Optionen hinzugefügt werden.

Tabelle 4.5: Manueller End-2-End Test E2E-005

ID	E2E-006
Beschreibung	Pfad zum Überwachen mit Regex-Pattern hinzufügen.
Vorbedingungen	<ul style="list-style-type: none"> • Daemon läuft • Der [PFAD] wurde noch nicht hinzugefügt
Ablauf	<ol style="list-style-type: none"> 1. <code>ts monitor add [PFAD] --mode pattern --pattern ^ts_.*_ts\$</code> 2. Erwartet: Successfully added [PFAD] to the monitoring database. 3. <code>ts monitor list</code> 4. Erwartet: Pfad wird mit <code>pattern=^ts_.*_ts\$</code> aufgelistet.
Erwartetes Ergebnis	Ein Pfad kann mit einem gewünschten Regex-Pattern hinzugefügt werden.

Tabelle 4.6: Manueller End-2-End Test E2E-006

ID	E2E-007
Beschreibung	Snapshots generieren lassen und vergleichen.
Vorbedingungen	<ul style="list-style-type: none"> • Daemon läuft • Der [PFAD] wurde noch nicht hinzugefügt. (Empfohlen: ein nicht zu tiefer Pfad, damit die Snapshots schneller generiert werden)
Ablauf	<ol style="list-style-type: none"> 1. <code>ts monitor add [PFAD] && ts monitor list</code> 2. Erwartet: Auflistung des zu überwachenden Pfads, sowie dessen [ID]. 3. <code>ts kill && ts run</code> 4. Warten bis Snapshot in Datenbank vorhanden ist. Mit nachfolgendem Schritt überprüfen. 5. <code>ts snapshots list [ID]</code> 6. Erwartet: Liste mit genau 1 Snapshot. 7. Eine Datei test.log in [PFAD] erstellen und mit Inhalt füllen. 8. <code>ts kill && ts run</code> 9. Warten bis zweiter Snapshot in Datenbank vorhanden ist. Mit nachfolgendem Schritt überprüfen. 10. <code>ts snapshots list [ID]</code> 11. Erwartet: Liste mit genau 2 Snapshots. 12. <code>ts snapshots compare [ID]</code> 13. Erwartet: Unterschied in der Datei test.log wird aufgelistet. 14. <code>ts kill && ts run</code> 15. Warten bis in Datenbank genau 3 Snapshots vorhanden sind. (siehe oben) 16. <code>ts kill && ts run</code> 17. Warten bis in Datenbank genau 4 Snapshots vorhanden sind. (siehe oben) 18. Den Inhalt der Datei test.log ändern. 19. <code>ts kill && ts run</code> 20. Warten bis in Datenbank genau 5 Snapshots vorhanden sind. (siehe oben)

Ablauf	21. <code>ts snapshots compare [ID]</code> 22. Erwartet: Unterschied in der Datei test.log wird aufgelistet. 23. test.log löschen. 24. <code>ts kill && ts run</code> 25. Warten bis in Datenbank genau 6 Snapshots vorhanden sind. 26. <code>ts snapshots compare [ID]</code> 27. Erwartet: Löschung der Datei test.log wird aufgelistet.
Erwartetes Ergebnis	Snapshots werden generiert und drei verschiedenen Zeitintervalle verglichen. Erstellung, Änderung und Löschung einer Datei werden erkannt.

Tabelle 4.7: Manueller End-2-End Test E2E-007

ID	E2E-008
Beschreibung	Search (Cache- und Log-Dateien)
Vorbedingungen	<ul style="list-style-type: none"> • Daemon läuft • Beispiel-Verzeichnis unter [PFAD] vorhanden
Ablauf	1. <code>ts search [PFAD]</code> 2. Erwartet: Auflistung der Inhalte unter [PFAD] welche “cache“ oder “log“ enthalten.
Erwartetes Ergebnis	Auflistung von Dateiinhalten in einem Verzeichnis.

Tabelle 4.8: Manueller End-2-End Test E2E-008

ID	E2E-009
Beschreibung	Leeren und Löschen einer Datei
Vorbedingungen	<ul style="list-style-type: none"> • Datei unter [PFAD] ist nicht geschützt. (z.B. durch Berechtigungen) • Daemon läuft
Ablauf	<ol style="list-style-type: none"> 1. <code>ts wipe [PFAD]</code> 2. Erwartet: <code>Successfully cleared file.</code> 3. Erwartet: Datei wurde geleert 4. <code>ts wipe [PFAD] --remove</code> 5. Erwartet: <code>Successfully removed file.</code> 6. Erwartet: Datei wurde gelöscht
Erwartetes Ergebnis	Datei wird geleert und gelöscht.

Tabelle 4.9: Manueller End-2-End Test E2E-009

4.1.2 Testergebnisse

Datum	Betriebssystem	Testfall	Ergebnis
26.12.2024	Windows 11–10.0	E2E-001	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-001	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-001	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-002	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-002	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-002	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-003	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-003	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-003	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-004	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-004	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-004	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-005	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-005	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-005	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-006	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-006	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-006	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-007	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-007	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-007	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-008	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-008	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-008	Erfolgreich
26.12.2024	Windows 11–10.0	E2E-009	Erfolgreich
26.12.2024	Linux 6.1.0-28	E2E-009	Erfolgreich
26.12.2024	macOS Sequoia 15.2	E2E-009	Erfolgreich

Tabelle 4.10: Manuelle End-2-End Testergebnisse

4.2 Messungen (Performanztests)

Um eine Aussage über die Performanz unseres Systems zu machen, haben wir einige Performanztests durchgeführt. Diese dienen lediglich als Indikator und sind nicht repräsentativ für alle möglichen Szenarien. Es werden nur Kernfunktionen getestet, die performanzkritisch sein könnten.

4.2.1 Suchfunktion

Es wird im Modus FULL (Cache- und Logdateien) im spezifizierten Pfad nach Dateien gesucht.

4.2.1.1 Verwendete Umgebungsangaben und Metriken

System	Betriebssystem und Architektur
Pfad	Verzeichnis, in dem gesucht wird
Anz. Dateien im Pfad	Anzahl Dateien im Pfad (mit allen Unterverzeichnissen)
Anz. Verzeichnisse im Pfad	Anzahl Unterverzeichnisse
Anz. Ebenen des erstellten Baumes	Anzahl Ebenen im konstruierten Baum aller gefundenen Dateien und Verzeichnisse
Anz. Knoten des erstellten Baumes	Anzahl Knoten im konstruierten Baum aller gefundenen Dateien und Verzeichnisse
Gefundene Dateien	Anzahl Dateien, die gefunden wurden
Dauer (Stichprobenmittel)	Mittelwert der Dauer in Millisekunden aus mehreren Stichproben

Tabelle 4.11: Umgebungsangaben und Metriken Suchfunktion-Performanztest

4.2.1.2 Ergebnisse

System	Windows 11–10.0 / amd64
Pfad	C:\Windows
Anz. Dateien im Pfad	188'428
Anz. Verzeichnisse im Pfad	51'184
Anz. Ebenen des erstellten Baumes	14
Anz. Knoten des erstellten Baumes	54'682
Gefundene Dateien	3'455
Dauer (Stichprobenmittel)	8'800ms

Tabelle 4.12: Performanztest Suche WIN-1

System	Windows 11–10.0 / amd64
Pfad	C:\Program Files
Anz. Dateien im Pfad	66'386
Anz. Verzeichnisse im Pfad	8'925
Anz. Ebenen des erstellten Baumes	20
Anz. Knoten des erstellten Baumes	10'109
Gefundene Dateien	1'183
Dauer (Stichprobenmittel)	800ms

Tabelle 4.13: Performanztest Suche WIN-2

System	macOS Sequoia 15.2 / aarch64
Pfad	/Library/Logs/
Anz. Dateien im Pfad	229
Anz. Verzeichnisse im Pfad	89
Anz. Ebenen des erstellten Baumes	9
Anz. Knoten des erstellten Baumes	120
Gefundene Dateien	31
Dauer (Stichprobenmittel)	6.6ms

Tabelle 4.14: Performanztest Suche MAC-1

System	macOS Sequoia 15.2 / aarch64
Pfad	/Library/Caches/
Anz. Dateien im Pfad	677
Anz. Verzeichnisse im Pfad	8
Anz. Ebenen des erstellten Baumes	3
Anz. Knoten des erstellten Baumes	6
Gefundene Dateien	0
Dauer (Stichprobenmittel)	0.6ms

Tabelle 4.15: Performanztest Suche MAC-2

System	Linux 6.1.0-28 / amd64
Pfad	/opt/idea-IU-242.22855.74
Anz. Dateien im Pfad	2248
Anz. Verzeichnisse im Pfad	793
Anz. Ebenen des erstellten Baumes	10
Anz. Knoten des erstellten Baumes	806
Gefundene Dateien	13
Dauer (Stichprobenmittel)	21ms

Tabelle 4.16: Performanztest Suche LIN-1

System	Linux 6.1.0-28 / amd64
Pfad	/home/luca
Anz. Dateien im Pfad	47'3915
Anz. Verzeichnisse im Pfad	9'8549
Anz. Ebenen des erstellten Baumes	29
Anz. Knoten des erstellten Baumes	104'448
Gefundene Dateien	5'429
Dauer (Stichprobenmittel)	2'663ms

Tabelle 4.17: Performanztest Suche LIN-2

4.2.2 Monitoring

Es werden mehrere Pfade über einen längeren Zeitraum (simuliert) überwacht bzw. Snapshots erstellt.

4.2.2.1 Verwendete Umgebungsangaben und Metriken

System	Betriebssystem und Architektur
Pfade	Verzeichnisse, die überwacht werden
Anz. Erstellte Snapshots	Anzahl erstellter Snapshots
Anz. Erstellte Knoten	Anzahl erstellter Knoten in der Datenbank
Benötigte Zeit	Gesamte Zeit, die benötigt wurde, um alle Snapshots zu erstellen
Durchschnittliche Dauer zum Erstellen eines Snapshots	Durchschnittliche Dauer, die benötigt wurde, um einen Snapshot zu erstellen

Tabelle 4.18: Umgebungsangaben und Metriken Monitoring-Performanztest

4.2.2.2 Ergebnisse

System	Windows 11-10.0 / amd64
Pfade	<ul style="list-style-type: none">• C:\Users\Luca\AppData\Roaming\discord\Cache• C:\Users\Luca\AppData\Roaming\discord\logs• C:\Users\Luca\AppData\Roaming\Docker Desktop\Cache• C:\Users\Luca\AppData\Roaming\Microsoft• C:\Users\Luca\AppData\Roaming\NVIDIA• C:\ProgramData\ASUS• C:\ProgramData\Package Cache• C:\ProgramData\Corsair• C:\ProgramData\NVIDIA Corporation• C:\ProgramData\LGHUB
Anz. Erstellte Snapshots	7'200
Anz. Erstellte Knoten	583'920
Benötigte Zeit	8min
Durchschnittliche Dauer zum Erstellen eines Snapshots	670ms

Tabelle 4.19: Performanztest Monitoring WIN

System	macOS Sequoia 15.2 / aarch64
Pfade	<ul style="list-style-type: none"> • /Library/Logs/ • /Library/Caches/ • /Library/Google • /Library/Google • /Library/Apple • /Library/OSAnalytics • /Library/Filesystems • /Library/Bluetooth • /Users/janicscherer/IdeaProjects • /Users/janicscherer/bin • /tmp
Anz. Erstellte Snapshots	7'200
Anz. Erstellte Knoten	9'079'920
Benötigte Zeit	10min
Durchschnittliche Dauer zum Erstellen eines Snapshots	914ms

Tabelle 4.20: Performanztest Monitoring MAC

System	Linux 6.1.0-28 / amd64
Pfade	<ul style="list-style-type: none"> • /home/luca/tracesentry-1.0.2 • /opt/idea-IU-242.22855.74 • /opt/vivaldi • /tmp • /etc • /usr/local • /usr/share/vim • /dev • /opt/az/lib/python3.12/email • /opt/android-studio
Anz. Erstellte Snapshots	7'200
Anz. Erstellte Knoten	1'717'920
Benötigte Zeit	2min
Durchschnittliche Dauer zum Erstellen eines Snapshots	173ms

Tabelle 4.21: Performanztest Monitoring LIN

4.3 KI Fallstudie

In diesem Abschnitt wird das Potenzial des, in TraceSentry integrierten, KI-Modells für die Erkennung von Anomalien in Logdateien untersucht. Es wurden insgesamt 46 Logdateien generiert und per Inspektionsfunktion an das KI-Modell übergeben.

4.3.1 Wahl des KI-Modells

4.3.1.1 Generative KI

Generative KI-Modelle, wie die GPT-Modelle von OpenAI sind in der Lage unstrukturierte Daten zu analysieren und Einschätzungen in Form von Text auszugeben. Diese Technologie hat in den letzten Jahren erheblich an Popularität gewonnen und wird sehr breit als Lern- oder Arbeitswerkzeug eingesetzt. Für den Anwendungsfall von TraceSentry wurde sich für generative KI entschieden, da:

- sie in der Lage ist, Informationen aus unstrukturierten Daten zu extrahieren und zu bewerten. Obwohl einzelne Logdateien strukturiert sind, ist die Gesamtheit verschiedener Logdateien unstrukturiert und schwer zu interpretieren bzw. übergreifende Muster zu erkennen.
- sie in der Lage ist, grundlegende semantische Zusammenhänge zu erkennen und dadurch sensible oder schädliche Informationen zu identifizieren.
- die Popularität und der breite Einsatz von Modellen wie ChatGPT darauf hinweisen, dass generative KI eine vielversprechende Technologie ist und breite Anwendungsfälle hat.
- ein praxisnaher Anwendungsfall evaluiert werden sollte, um die Grenzen und Möglichkeiten von aktuellen KI-Technologien zu untersuchen.

Darüber hinaus ermöglicht generative KI eine effiziente Verarbeitung und Interpretation von Textdaten, was sie zu einer idealen Wahl für den Anwendungsfall von TraceSentry macht.

4.3.1.2 OpenAI GPT-4o mini

Die Wahl des KI-Modells fiel auf das GPT-4o mini Modell von OpenAI. Dieses Modell befindet sich zusammen mit den Modellen GPT-4o, o1 und o1-mini im Flagship-Portfolio von OpenAI (Stand September 2024). Es ist zudem das aktuelle Standardmodell auf <https://chatgpt.com/> und ohne Einschränkung im Gratis-Tarif verfügbar. Die genaue Modellbezeichnung lautet `gpt-4o-mini-2024-07-18`, wobei o für *omni* steht, was den breiten Anwendungsbereich des Modells beschreibt. *mini* steht für die leichtgewichtige Variante des grossen 4o Modells. Es ist speziell für schnelle und ressourcenschonende, fokussierte Anwendungen entwickelt worden. Es kann durch sogenannte *fine-tuning*- oder *distillation*-Techniken annähernd an die Leistung des grossen 4o Modells herankommen, dies sogar oft mit weniger Kosten und Latenz. Der sogenannte *knowledge-cutoff* des Modells liegt im Oktober 2023. Dies bedeutet, dass das Modell mit Daten bis zu diesem Zeitpunkt trainiert wurde. Für den Anwendungsfall des TraceSentry wurden die Modelle 4o und o1-mini initial evaluiert. Die Resultate wiesen jedoch keine signifikanten Unterschiede auf. Die deutlich höheren Kosten und die längere Latenz beider Modelle waren

deren ausschlaggebende Ausschlusskriterien.

4.3.2 Aktuelle Einschränkungen

Das sogenannte Context Window des GPT-4o mini Modells beträgt, wie auch dasjenige des GPT-4o Modells, 128'000 Token. Das bedeutet, dass die Eingabe (Prompt) und die Ausgabe (Completion) in der Summe maximal 128'000 Token lang sein dürfen. Grundsätzlich lässt sich 1 Token zu rund $\frac{3}{4}$ Wörtern umrechnen. Da jedoch in Systemdateien wie Log- oder Cache-Dateien viele Sonderzeichen und Leerzeichen oder Klammern vorkommen, ist eher mit einem Verhältnis von 1 Token zu $\frac{1}{2}$ Wörtern zu rechnen. Eine genaue Umrechnung kann mit dem Online-Werkzeug <https://platform.openai.com/tokenizer> durchgeführt werden.

4.3.2.1 Praxisbeispiel

Nachfolgend eine Zeile aus einer Webserver-Logdatei:

```
[Thu Jul 07--07:54:05 2005] [error] [client 221.203.160.2] Directory  
index forbidden by rule: /var/www/html/
```

Diese 107 Zeichen lange Zeile entspricht 40 Tokens. Angenommen die System-Prompt und Completion soll 2'000 Tokens lang sein, so könnte eine Logdatei mit maximal $\frac{128'000 - 2'000}{40} = 3'150$ Zeilen analysiert werden.

4.3.3 Vorgehensweise

Ziel war es die Testdateien per HTTP-Schnittstelle an den Daemon zu senden, welcher diese mit der System-Prompt an das KI-Modell weiterleitet. Alle Testdateien befanden sich auf dem Testsystem. Der Aufruf der HTTP-Schnittstelle erfolgte über einen Spring-BootTest im Daemon-Projekt. Die Antworten des KI-Modells wurden in einem Ausgabe-Verzeichnis als Textdateien abgelegt. Diese wurden schliesslich manuell bewertet und mit der zuvor definierten Erwartungshaltung verglichen.

4.3.4 Testdateien

Im Internet existieren zwar zahlreiche reale, vorgelabelte und gesäuberte Datasets, welche für das Erkennen von Anomalien in Logdateien verwendet werden können. Diese befinden sich jedoch vorwiegend in einer Form, welche sich für Machine Learning eignet. Sprich, in CSV-Dateien oder als riesige, zusammengeführte Textdateien, welche sich oft über mehrere Gigabyte erstrecken.

4.3.4.1 Beispiele:

- <https://github.com/logpai/loghub>
- <https://www.kaggle.com/datasets/ispangler/csic-2010-web-application-attacks>

4.3.4.2 Sammlung eigener Testdateien

Da kein Dataset in der passenden Form gefunden werden konnte, wurden eigene Testdateien generiert sowie Logs aus der Gitlab CI/CD Pipeline verwendet. Zur Generierung der Logdateien für unsere Fallstudie wurde ein Python-Skript erstellt. Dieses Skript enthält sogenannte Template-Funktionen für die Log-Typen HTTP, System (Linux), MySQL, Router, Authentifizierung, SMTP und Docker der Form:

```
def logtyp_template(is_anomalous=False)
```

Die Funktion generiert genau eine Zeile, wobei der Parameter `is_anomalous` bestimmt, ob die Zeile anomalous sein soll. Jede Funktion enthielt eine Auswahl an realistischen Anomalien sowie eine Auswahl an normalen Zeilen. Dynamische Inhalte wie IP-Adressen, URLs, Dateipfade, etc. wurden zufällig generiert. Nachfolgend alle generierten Log-Typen:

Log-Typ	Normale Zeilen	Anomalien
HTTP	<ul style="list-style-type: none"> • Einfacher GET-Request 	<ul style="list-style-type: none"> • SQL-Injektion • Ungültiger Statuscode • Unauthorisierter Zugriff
System	<ul style="list-style-type: none"> • USB-Gerät angeschlossen • CRON-Job ausgeführt • CPU-Temperatur über Schwellwert • systemd-Service gestartet • Netzwerkkonfiguration aktualisiert • systemd-Service gestoppt • Speicherzuweisungsfehler 	<ul style="list-style-type: none"> • Kernel Panic • Unauthorisierter Zugriff
MySQL	<ul style="list-style-type: none"> • SELECT-Queries • UPDATE-Queries • DELETE-Queries 	<ul style="list-style-type: none"> • SELECT-Query mit SQL-Injektion • DROP TABLE-Query • INSERT-Query mit Angreiferdaten
Router	<ul style="list-style-type: none"> • Packet akzeptiert • Packet abgelehnt 	<ul style="list-style-type: none"> • Unauthorisierter Zugriff • Port-Scanning • DoS-Attacke
Auth	<ul style="list-style-type: none"> • Login akzeptiert • Login abgelehnt 	<ul style="list-style-type: none"> • Brute Force • Unauthorisierter Zugriff
SMTP	<ul style="list-style-type: none"> • E-Mail versendet 	<ul style="list-style-type: none"> • Spam • Relay abgelehnt • Invalide Domain
Docker	<ul style="list-style-type: none"> • Container gestartet • Container gestoppt • Container neugestartet 	<ul style="list-style-type: none"> • Container abgestürzt • Image-Pull fehlgeschlagen • Speicherlimit erreicht • Speicherplatz niedrig • Unauthorisierter Zugriff

Tabelle 4.22: Generierte Log-Typen

Insgesamt wurde für alle 7 Log-Typen folgende Anzahl an Testdateien generiert:

- 2 normale Testdateien ohne Anomalien
- 2 Testdateien mit seltenen Anomalien (alle 40–60 Zeilen)
- 2 Testdateien mit häufigeren Anomalien (alle 18–25 Zeilen)

Dies führte zu insgesamt 42 generierten Testdateien. Dazu wurden 2 normale sowie 2 anomale (häufig) Logdateien aus der Gitlab CI/CD Pipeline verwendet. Somit standen insgesamt 46 Testdateien zur Verfügung.

4.3.5 Ergebnisse

4.3.5.1 Normale Testdateien

Aus den 16 normalen Testdateien wurden 9 als unschädlich und 7 als potenziell schädlich eingestuft. Die empfohlene Aktion für alle 7 potenziell schädlichen Dateien war diese zu leeren. Die Gründe für die potenzielle Schädlichkeit waren:

- Auth: viele fehlgeschlagene Login-Versuche
- MySQL: viele manipulative SQL-Queries
- Router: viele abgelehnte Pakete
- SMTP: Senderadressen Spam-Verdächtig

Die Klassifizierung als potenziell schädlich erfolgte in jedem Fall gut begründet und nachvollziehbar. Das Ergebnis ist vor allem auf die Beschaffenheit der Testdateien zurückzuführen.

4.3.5.2 Anomale Testdateien (seltene Anomalien)

Aus den 14 Testdateien mit seltenen Anomalien wurden 11 als potenziell schädlich, 2 als schädlich und 1 als unschädlich eingestuft. Die empfohlenen Aktionen korrelierten mit der Einstufung. Alle potenziell schädlichen Dateien sollten geleert, alle schädlichen Dateien gelöscht und die unschädliche Datei ignoriert werden.

Nennenswerte Ergebnisse:

- MySQL: Beide Dateien wurden als schädlich eingestuft, dies ist auf die Anomalien zurückzuführen, welche klar als Angriffe identifiziert wurden.
- Router: Eine Datei wurde als unschädlich eingestuft, dies da die Anomalien als zu harmlos eingestuft wurden.

Auch hier lassen sich die Ergebnisse auf die Beschaffenheit der Testdateien zurückführen.

4.3.5.3 Anomale Testdateien (häufige Anomalien)

Aus den 16 Testdateien mit häufigen Anomalien wurden 11 als potenziell schädlich, 2 als schädlich und 3 als unschädlich eingestuft. Die empfohlene Aktionen korrelierten, wie bereits bei den seltenen Anomalien, mit der Einstufung. Unerwartete Ergebnisse:

- Docker: Eine Datei wurde als unschädlich eingestuft, dies da die Anomalien als zu harmlos eingestuft wurden. Fehler beim Image-Pull oder Speicherplatzmangel wurden als zu harmlos interpretiert.
- Gitlab: Beide Dateien wurden als unschädlich eingestuft, dies da fehlgeschlagene Pipelines als zu harmlos eingestuft wurden.

4.3.5.4 Zusammenfassung

Aus den 46 Testdateien wurden insgesamt 7 schädlicher und 4 weniger schädlich eingestuft als erwartet. Alle Klassifizierungen erfolgten, unter Berücksichtigung der Beschaffenheit der Testdateien, nachvollziehbar und gut begründet. Ebenso wurden alle empfohlenen Aktionen nachvollziehbar begründet und korrelierten mit der Einstufung. Die Verwendungszwecke der Testdateien wurden richtig hergeleitet und beschrieben.

4.3.6 Kostenanalyse

Die genauen Kosten sind auf <https://platform.openai.com/> ersichtlich und werden pro Tag und Modell dargestellt. Die Kosten für die Analyse der 46 Testdateien beliefen sich auf insgesamt 0.58 USD. Diese lassen sich in nachfolgende Kategorien unterteilen:

- 0.561 USD für das Annehmen des Inputs (System-Prompt + Testdatei)
- 0.004 USD für das Cachen einiger Teile des Inputs
- 0.016 USD für das Erstellen des Outputs (Completion)

4.3.7 Fazit

Die Ergebnisse der KI Fallstudie sind als erfolgreich zu bewerten. Das Herleiten der Verwendungszwecke sowie die Klassifizierung der Testdateien erfolgten nachvollziehbar und gut begründet. Die empfohlenen Aktionen korrelierten mit der Einstufung und waren nachvollziehbar begründet. Die Kosten für die Analyse der Testdateien beliefen sich auf insgesamt 0.58 USD, was rund 0.011 USD pro Testdatei entspricht.

Der Verwendungszweck sowie Empfehlungen für die Testdateien wurden grundsätzlich technisch beschrieben und waren für technisch versierte Personen verständlich. In Anbetracht der Tatsache, dass Logdateien wie MySQL-Logs oder Router-Logs eher von technisch versierten Personen analysiert werden, ist dies als positiv zu bewerten.

5 Bereitstellung/Integration

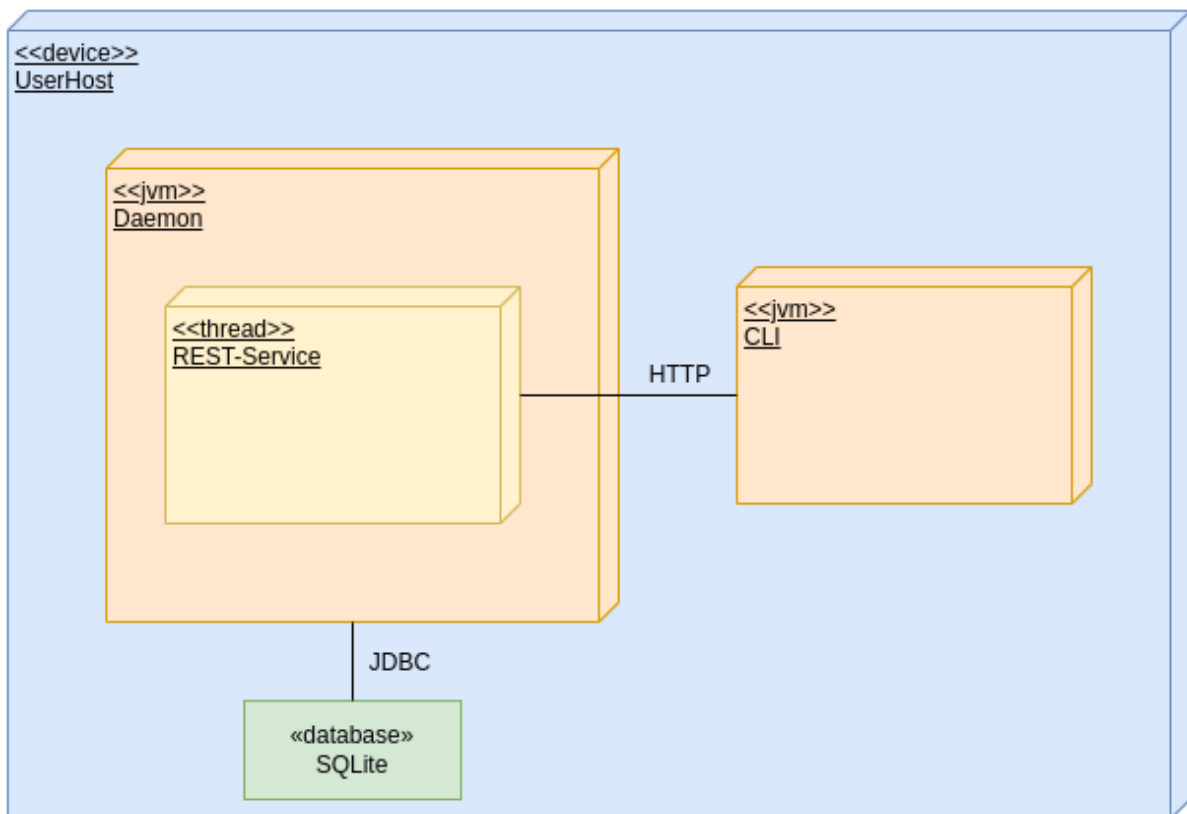


Abbildung 5.1: UML Deploymentdiagramm

Das ganze System befindet sich während der Laufzeit auf dem Host-Gerät des Anwenders. Dazu werden zwei separate Jar-Dateien ausgeliefert. Die Erste sollte bereits beim Start des Geräts ausgeführt werden (Autostart) und die JVM für den Daemon starten. Dieser startet einen Webservice in Form einer REST-Schnittstelle für dessen Steuerung. Der Daemon kommuniziert ausserdem via JDBC mit der SQLite-Datenbank, die als Datei auf dem Host-Gerät gespeichert wird.

Das zweite Jar wird manuell vom Anwender gestartet werden, sobald er Funktionen des Daemons ausführen möchte. Dieses Jar startet wiederum eine JVM, welche ein CLI für den Anwender mit bestimmten Kommandos zur Verfügung stellt. Dieses CLI konsumiert die REST-Schnittstelle des Daemons zur Ausführung der Kommandos.

5.1 Lizenzierung

Unser Projekt basiert auf den Prinzipien von **Free/Libre and Open Source Software (FLOSS)** und ist unter der **MIT-Lizenz** veröffentlicht. Diese Lizenz ermöglicht es, die Software frei zu nutzen, zu modifizieren und weiterzuverbreiten, sowohl für private als auch kommerzielle Zwecke, solange der ursprüngliche Lizenztext beibehalten wird.

5.1.1 Open-Source Abhängigkeiten

Im Projekt werden folgende Technologien und Bibliotheken verwendet, die ebenfalls unter Open-Source-Lizenzen stehen:

- **Java:** Entwickelt unter der **GPL v2 mit Classpath Exception**, was die Nutzung in geschlossenen Projekten ermöglicht.
- **Maven:** Unter der **Apache License 2.0**, die eine flexible Nutzung und Modifikation erlaubt.
- **Spring:** Ebenfalls lizenziert unter der **Apache License 2.0**, was eine Integration und Anpassung in eigenen Projekten ohne Einschränkungen unterstützt.
- **SQLite:** Steht unter der **Public Domain** bzw. der **blessing license**, wodurch es uneingeschränkt genutzt und verteilt werden kann.
- **H2:** Unter der **Mozilla Public License 2.0 (MPL 2.0)**, die flexible Nutzung, Modifikation und Distribution erlaubt, solange Änderungen am Quellcode unter derselben Lizenz veröffentlicht werden.

Durch die sorgfältige Auswahl dieser FLOSS-Lizenzen bleibt das Projekt vollständig mit den Open-Source-Grundsätzen kompatibel und bietet maximale Flexibilität für zukünftige Erweiterungen.

5.1.2 Kommerzielle Abhängigkeiten

Zusätzlich wird im Projekt ein kostenpflichtiger **GPT-Service** von *OpenAI* eingebunden, der über eine API-Schnittstelle angebunden ist. Die Nutzung dieses Dienstes unterliegt den **kommerziellen Nutzungsbedingungen von OpenAI**. Es wird sichergestellt, dass der Zugriff auf diesen Dienst in unserem Projekt optional ist und die Kernfunktionalität auch ohne dessen Nutzung gewährleistet bleibt. Somit bleibt das Projekt als FLOSS klassifiziert, während die GPT-Integration als optionaler, externer Dienst gehandhabt wird.

5.2 Installationshandbuch & Skript

In diesem Kapitel sind die Installationsanleitungen für alle unterstützten Plattformen beschrieben. Grundsätzlich ist der TraceSentry ein plattformunabhängiges System, welches auf den gängigen Betriebssystemen (Windows, Linux, MacOS) lauffähig ist. Entwickelt und getestet wurde allerdings nur auf den folgenden Betriebssystemversionen, weshalb die Kompatibilität mit anderen Versionen nicht garantiert werden kann:

- Windows 11–10.0
- Linux 6.1.0-28
- macOS Sequoia 15.2

Für alle nachfolgenden Schritte wird folgendes vorausgesetzt:

- Eine Installation von Java 21+ (JRE).
- Eine aktuelle Maven-Installation (insofern manuell kompiliert wird).
- Das zip-Archiv wurde am gewünschten Installationsort entpackt.

5.2.1 OpenAI API Key

Wenn Sie die Inspektionsfunktion nutzen möchten, erstellen Sie einen OpenAI-API-Schlüssel, indem Sie den Anweisungen unter <https://platform.openai.com/settings/organization/api-keys> folgen. Fügen Sie Guthaben zu Ihrem Konto hinzu, um die OpenAI-API nutzen zu können: <https://platform.openai.com/settings/organization/billing/overview>. Dieser API Key wird in nachfolgenden Schritten benötigt. Sollten Sie diesen Teil überspringen, werden Sie keinen Zugriff auf die Inspektionsfunktion haben.

5.2.2 Skripte

Für eine einfache Installation sind für alle unterstützten Plattformen Skripte im Installationsverzeichnis bereitgestellt.

- **build** Kompiliert das Projekt, erstellt die ausführbaren JAR-Dateien und ersetzt diese im bin-Ordner.
- **setup** Erstellt notwendige Umgebungsvariablen und richtet den Autostart des Daemons ein.
- **build-setup** Sequenzielle Ausführung von **build** und **setup**.

Da die JAR-Dateien bereits im Installationsverzeichnis enthalten sind, ist ein **manuelles Kompilieren nicht notwendig**. Es wird im Folgenden empfohlen, direkt **setup** auszuführen. Damit der OpenAI API Key von den Skripten erkannt wird, muss er sich in einer Datei namens **apikey.txt** im Installationsverzeichnis befinden.

5.2.2.1 Linux

Folgende Skripte sind für Linux verfügbar:

- `build`
- `setup-lin`
- `build-setup-lin`

Das `setup` Skript sollte nur einmal ausgeführt werden, um doppelte Einträge in den System-Dateien zu vermeiden.

Im Vorfeld muss sichergestellt werden, dass die CLI und die Skripte ausgeführt werden können. Dies wird mit folgendem Befehl erreicht:

Listing 5.1: Linux Berechtigungen

```
chmod +x ts
chmod +x <verwendete Skripte>
```

Nachfolgend die empfohlene Ausführung von `setup-lin`:

Listing 5.2: Linux Installation

```
./setup-lin
```

5.2.2.2 Windows

Folgende Skripte sind für Windows verfügbar:

- `build.ps1`
- `setup-win.ps1`
- `build-setup-win.ps1`

Im Vorfeld muss sichergestellt werden, dass die Skripte ausgeführt werden können. **Achtung:** hierbei wird die Ausführung von allen Skripten erlaubt. Diese Aktion kann nur von Administratoren durchgeführt werden und wird mit folgendem Befehl in PowerShell erreicht:

Listing 5.3: Windows Berechtigungen

```
set-executionpolicy unrestricted
```

Nachfolgend die empfohlene Ausführung von `setup-win.ps1`:

Listing 5.4: Windows Installation

```
.\setup-win.ps1
```

5.2.2.3 MacOS

Folgende Skripte sind für MacOS verfügbar:

- `build`
- `setup-mac`
- `build-setup-mac`

Achtung: Der Einfachheit halber werden ausschliesslich Konfigurationen im `~/.zprofile` File vorgenommen. Diese können bei Bedarf manuell entfernt werden. Das `setup` Skript sollte nur einmal ausgeführt werden, um doppelte Einträge im `~/.zprofile` zu vermeiden.

Im Vorfeld muss sichergestellt werden, dass die CLI und die Skripte ausgeführt werden können. Dies wird mit folgendem Befehl erreicht:

Listing 5.5: MacOS Berechtigungen

```
chmod +x ts
chmod +x <verwendete Skripte>
```

Sollte zusätzlich ein Quarantine-Tag gesetzt sein, kann dieser mit folgendem Befehl entfernt werden:

Listing 5.6: MacOS Quarantäne-Tag

```
xattr -d com.apple.quarantine ts
xattr -d com.apple.quarantine <skript>
```

Nachfolgend die empfohlene Ausführung von `setup-mac`:

Listing 5.7: MacOS Installation

```
./setup-mac
```


5.2.3 Manuelle Installation

Die manuelle Installation wird nur empfohlen, wenn die Skripte nicht funktionieren oder die Installation individuell angepasst werden soll.

5.2.3.1 Linux

1. Definieren Sie die folgenden Umgebungsvariablen in der Datei `/etc/environment`:

```
JAVA_HOME=<absoluter Pfad zum JDK-Ordner>
TRACE_SENTRY_DIR=<absoluter Pfad zum
    Installationsverzeichnis>
```

Und fügen Sie die Variable `TRACE_SENTRY_DIR` am Ende der Datei `/etc/profile` zum `PATH` hinzu:

```
PATH=$PATH:$TRACE_SENTRY_DIR
```

2. Setzen Sie anschliessend den generierten API-Schlüssel als Umgebungsvariable am Ende der Datei `/etc/profile`:

```
export OPENAI_API_KEY=<generierter API-Schlüssel>
```

3. Wenn der Daemon bei jedem Systemstart automatisch im Hintergrund gestartet werden soll, können Sie Folgendes in einem Terminal ausführen:

```
crontab -e
// Fügen Sie die folgende Zeile am Ende der geöffneten
    Datei hinzu
@reboot PATH=$JAVA_HOME/bin:$PATH $TRACE_SENTRY_DIR/ts run
```

Speichern Sie dann die Datei.

4. Nach einem Systemneustart sind Sie bereit, TraceSentry zu verwenden.

5.2.3.2 Windows

1. Setzen Sie die Umgebungsvariable TRACE_SENTRY_DIR via Systemsteuerung oder via PowerShell:

```
[System.Environment]::SetEnvironmentVariable("TRACE_SENTRY_DIR", "<absoluter Pfad zum Installationsverzeichnis>", "User")
```

2. Damit die CLI korrekt funktioniert, fügen Sie das Installationsverzeichnis zum PATH hinzu:

```
$currentPath = [System.Environment]::GetEnvironmentVariable("Path", "User")  
[System.Environment]::SetEnvironmentVariable("Path", "$currentPath;<absoluter Pfad zum Installationsverzeichnis>", "User")
```

3. Setzen Sie anschliessend den generierten API-Schlüssel als Umgebungsvariable via PowerShell:

```
[System.Environment]::SetEnvironmentVariable("OPENAI_API_KEY", "<generierter API-Schlüssel>", "User")
```

4. Wenn der Daemon bei jedem Systemstart automatisch im Hintergrund gestartet werden soll, muss die Datei ts-daemon.bat vom Installationsverzeichnis in den Autostart-Ordner kopiert werden:

```
Copy-Item "$env:TRACE_SENTRY_DIR\ts-daemon.bat" "$env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup"
```

Beim nächsten Systemstart wird der Daemon automatisch gestartet. Beim ersten Start kann eine Sicherheitswarnung erscheinen, in der Sie eine Checkbox deaktivieren müssen, damit diese in Zukunft nicht mehr erscheint.

5. Starten Sie das Terminal neu, und Sie sind bereit, TraceSentry zu verwenden.

5.2.3.3 macOS

Der Einfachheit halber werden hier ausschliesslich Konfigurationen im `.zprofile` File benützt. Die nachfolgenden Befehle schreiben die Konfigurationen direkt in die Datei.

1. Setzen Sie die Umgebungsvariable `TRACE_SENTRY_DIR` im `.zprofile` File (befindet sich in Ihrem Home-Verzeichnis):

```
echo '\n# Added for TraceSentry' >> ~/.zprofile
echo export TRACE_SENTRY_DIR="<absoluter Pfad zum  
Installationsverzeichnis>" >> ~/.zprofile
```

2. Damit die CLI korrekt funktioniert, fügen Sie im `.zprofile` File das Installationsverzeichnis zum `PATH` hinzu:

```
echo export PATH="\$TRACE_SENTRY_DIR:\$PATH" >> ~/.zprofile
```

3. Setzen Sie anschliessend den generierten API-Schlüssel als Umgebungsvariable im `.zprofile` File:

```
echo export OPENAI_API_KEY=<generierter API-Schlüssel> >>
~/.zprofile
```

4. Wenn der Daemon bei jedem Systemstart automatisch im Hintergrund gestartet werden soll, passen Sie das `.zprofile` File in ihrem Home-Verzeichnis wie folgt an:

```
echo ts run >> ~/.zprofile
```

Dies wird als pragmatische Lösung empfohlen. Alternativ können Sie auch *launchd* verwenden. [2]

5. Starten Sie das Terminal neu, dadurch wird der Daemon automatisch gestartet (sofern Sie Schritt 6 befolgt haben), und Sie sind bereit, TraceSentry zu verwenden.

5.3 Benutzerhandbuch

In diesem Abschnitt soll der Nutzer eine Anleitung für die Verwendung des TraceSentry-Systems erhalten. Die CLI-Dokumentation (siehe Anhang A.2) ist dafür zentral.

Vorab noch einige Hinweise zur Verwendung des Systems:

- Der TraceSentry kann sowohl im interaktiven Modus, als auch im nicht-interaktiven Modus verwendet werden. Der Unterschied besteht darin, dass beim interaktiven Modus zu Beginn ein einziges Mal die CLI in einer neuen Session gestartet wird und dann alle folgenden Befehle in dieser Session/Laufzeit ausgeführt werden. Dieser interaktive Modus wird gestartet, indem in einem Host-Terminal nur der Befehl `ts` ausgeführt wird. Hingegen wird beim nicht-interaktiven Modus die gesamte CLI-Applikation bei jedem Befehl neu gestartet und nach dessen beendeter Ausführung auch direkt wieder beendet. Der nicht-interaktive Modus wird gestartet, indem in einem Host-Terminal der Befehl `ts <Befehl>` ausgeführt wird.
- Im TraceSentry ist ein `help`-Befehl eingebaut, welcher eine Übersicht über alle verfügbaren Befehle und deren Verwendung anzeigt. Dieser Befehl wird genau gleich wie alle anderen Befehle ausgeführt, indem im Host-Terminal `ts help` eingegeben wird. Um genauere Informationen zu einem bestimmten Befehl zu erhalten, kann nach dem `help`-Befehl auch der Name des gewünschten Befehls angegeben werden (z.B. `ts help search`).
- Für die Verwendung mit Windows ist zu beachten, dass alle Pfade in den Befehlen mit **zwei** Backslashes (`\\`) geschrieben werden müssen.
- Pfade welche Leerzeichen enthalten, müssen in Anführungszeichen gesetzt werden.

6 Fazit

6.1 Diskussion

Das Projekt hat die Relevanz der Problematik im Umgang mit Cache- und Log-Dateien aufgezeigt und eine erste Lösung erarbeitet, mit der ein Anwender die Integrität dieser Dateien überwachen kann. Die Nutzung von generativen KI-Modellen zur Analyse von solchen Dateien hat sich dabei als effektives Mittel erwiesen, um Anomalien zu erkennen und einzuordnen.

Die Verwendung ebendieser KI-Modelle regt aber auch neue Diskussionen an, insbesondere im Hinblick auf den Datenschutz und die Privatsphäre. Denkt man an das Kernproblem - dem Schutz von sensiblen Daten - so ist die Verwendung eines kommerziellen und proprietären KI-Modells wie es die GPT-Modelle sind, durchaus kritisch zu betrachten und regt im Grunde das selbe Problem an, das sie eigentlich lösen sollte.

Des Weiteren ist die umgesetzte Lösung mit der Kommandozeilenschnittstelle vor allem für technisch versierte Anwender geeignet. Diese haben zwar auch Bedarf an einer solchen Lösung, jedoch sind sie eher in der Lage, auch ohne ein solches Tool die Risiken einzuschätzen und zu minimieren. Gerade für weniger technisch versierte Anwender, die oft nicht in der Lage sind, die Risiken zu erkennen und zu minimieren, wäre eine einfache und intuitive Lösung umso wichtiger.

Für eine zukünftige Weiterentwicklung könnten neben der besagten UX-Verbesserung etliche Funktionen und technische Verbesserungen vorgenommen werden. Das Problem bietet sehr viel Raum dafür.

Nichtsdestotrotz ist die Lösung in ihrer jetzigen Form ein Anfang, löst das ursprüngliche Problem und bietet eine Basis, auf der weiter aufgebaut werden kann. Die Arbeit bietet zudem die Möglichkeit, das Potential von generativen KI-Modellen für spezifische Anwendungsfälle zu erkennen und ein Verständnis zu schaffen, wie diese in Zukunft eingesetzt werden können.

6.2 Endergebnis

Im Verlauf des Projekts wurde eine Software entwickelt, die es ermöglicht, jegliche Art von Dateien zu suchen, zu überwachen und zu löschen. Dabei bietet das System direkt die Möglichkeit dies für Cache- und Log-Dateien zu tun. Mittels angebundenem KI-Modell, können schnell und effizient Einschätzungen zu solchen Dateien eingeholt werden und so potentielle Gefahren erkannt und eine Löschung veranlasst werden.

Es wurde ein FLOSS lizenziertes und plattformunabhängiges System entwickelt, das auf eine CLI- und Daemon-Architektur setzt und auf dem Host des Anwenders installiert und ausgeführt wird. In der jetzigen Form richtet es sich an eine Zielgruppe, zwischen Laien und Experten, vor allem in der Hinsicht der Kommandozeilenschnittstelle.

Während die praktische Umsetzung der Software die grundlegenden Anforderungen erfüllt, hat das Projekt zudem die Chancen und Grenzen von generativen KI-Modellen in einer Fallstudie aufgezeigt.

Die primäre Zielsetzung wurde erreicht und das Erweiterungspotential wird im folgenden Kapitel diskutiert.

6.2.1 Berechnung der Produktivitätssteigerung

In diesem Abschnitt wird die geschätzte Produktivitätssteigerung für Benutzer von TraceSentry berechnet. Die Berechnungsfaktoren umfassen die **Aufgabendauer ohne und mit der Software**, die resultierende **Zeitersparnis**, die **Kosten pro Zeiteinheit** und die **Gesamtkostenersparnis**.

Die folgenden Befehle sind in der CLI Dokumentation spezifiziert.

Es wird das `inspect` Kommando und die Monitoring-Funktionen betrachtet. Für Funktionen wie `search` und `wipe` wird keine Produktivitätssteigerung angenommen, da sie nur Shell-Befehle umhüllen. Administrationsbefehle wie `kill`, `run` etc. werden ebenfalls nicht betrachtet, diese sind “Mittel zum Zweck“ und es gibt logischerweise keine äquivalenten Vorgänge ohne die Software.

6.2.1.1 Berechnung

Annahmen Für die Berechnung der Produktivitätssteigerung werden folgende Annahmen getroffen:

- TraceSentry ist gemäss Installationshandbuch installiert (inkl. Autostart).
- Das Hochladen einer Datei in die GPT-Prompt dauert (inkl. dem Vorschlagen der Output-Formulierung bzw. -Formatierung) ca. 20 Sekunden.
- Die Analyse einer Datei in der GPT-Prompt dauert ca. 4 Sekunden. (Hierbei spielt vor allem die Qualität der Prompt-Formulierung eine Rolle.)
- Das Analysieren einer Datei mit TraceSentry (**inspect**) dauert samt Befehlseingabe ca. 10 Sekunden.
- Das Analysieren einer Datei mit GPT-4o mini kostet gemäss KI Fallstudie 0.011 USD \approx 0.01 CHF.
- Der durchschnittliche Stundenlohn beträgt 38 CHF.
- Eine Datei wird im Schnitt für eine Woche überwacht.
- Ein Computer läuft im Schnitt 6 Stunden pro Tag.
- Das Hinzufügen eines zu überwachenden Verzeichnisses, damit TraceSentry die darin enthaltenen Dateien überwachen kann, dauert ca. 10 Sekunden.
- Das Erstellen einer Kopie einer Datei (manueller Snapshot) dauert ca. 10 Sekunden.
- Das Vergleichen zweier Dateien dauert ca. 20 Sekunden.

Analyse inspect Funktion

Ohne TraceSentry:

Input vorbereiten : Dateiinhalt in GPT-Prompt kopieren/hochladen

Output abwarten : GPT-Response

$$\begin{aligned}\text{Aufgabendauer ohne SW} &= (\text{Input vorbereiten} + \text{Output abwarten}) \\ &= 20s + 4s \\ &= 24s\end{aligned}$$

Mit TraceSentry:

$$\begin{aligned}\text{Aufgabendauer mit SW} &= \text{inspect} \\ &= 10s\end{aligned}$$

Produktivitätssteigerung:

$$\text{Zeitersparnis} = 24 - 10 = 14s = 0.23min$$

$$\text{GPT-4o mini Kosten} = 0.01 \text{ CHF (pro Datei)}$$

$$\begin{aligned}\text{Kosten pro Zeiteinheit} &= 38\text{CHF/Stunde} \\ &\approx 0.63\text{CHF/Minute}\end{aligned}$$

$$\begin{aligned}\text{Gesamtkostenersparnis} &= 0.23 * 0.63\text{CHF/Minute} - 0.01\text{CHF} \\ &= 0.137 \text{ CHF} \\ &= 13.70 \text{ Rappen (pro Datei)}\end{aligned}$$

$$\begin{aligned}\text{Produktivitätssteigerung} &= \frac{\text{Zeitersparnis}}{\text{Aufgabendauer ohne SW}} \\ &= \frac{14}{24} \\ &= 58\%\end{aligned}$$

Analyse Monitoring Funktion

Ohne TraceSentry:

Erläuterung: Der Benutzer speichert hierbei für eine Woche, 6 Mal am Tag manuell die Datei (als Snapshot) in einen dafür vorgesehenen Ordner. Mit TraceSentry wäre es möglich mehrere Dateien zu überwachen, was nochmals zu einer signifikanteren Steigerung führen würde. Nachfolgend wird die Berechnung für eine Datei durchgeführt.

Manueller Snapshot erstellen : Datei in einen Ordner speichern

Snapshots vergleichen : zwei aufeinander folgende Snapshots vergleichen

$$\begin{aligned}\text{Aufgabendauer ohne SW} &= 6 * 7 * (\text{Manueller Snapshot erstellen} + \text{Snapshots vergleichen}) \\ &= 6 * 7 * (10s + 20s) \\ &= 1260s \\ &= 21min\end{aligned}$$

Mit TraceSentry:

$$\begin{aligned}\text{Aufgabendauer mit SW} &= \text{Hinzufügen eines zu überwachenden Verzeichnisses} \\ &= 10s\end{aligned}$$

Produktivitätssteigerung:

$$\begin{aligned}\text{Zeitersparnis} &= 1260 - 10 = 1250s \\ &\approx 20.83min\end{aligned}$$

$$\begin{aligned}\text{Kosten pro Zeiteinheit} &= 38\text{CHF}/\text{Stunde} \\ &\approx 0.63\text{CHF}/\text{Minute}\end{aligned}$$

$$\begin{aligned}\text{Gesamtkostenersparnis} &= 20.83 * 0.63\text{CHF}/\text{Minute} \\ &= 13.12 \text{ CHF (pro überwachte Datei)}\end{aligned}$$

$$\begin{aligned}\text{Produktivitätssteigerung} &= \frac{\text{Zeitersparnis}}{\text{Aufgabendauer ohne SW}} \\ &= \frac{1250}{1260} \\ &\approx 99\%\end{aligned}$$

6.3 Zukünftige Arbeiten

6.3.1 Funktionale Erweiterungen

6.3.1.1 Ideen aus der Originalen Projektbeschreibung

Folgende Features können dem Proposal (Originale Projektbeschreibung) entnommen werden. Teilweise wurden diese nicht vollständig umgesetzt. Dies hängt mit der generellen Philosophie der realisierten Lösung, sowie Gründen der Zeit und Komplexität zusammen.

- **Schlüsse aus Änderungen ziehen**¹: Die Abfrage des KI-Modells könnte erweitert werden, um nicht nur Dateiinhalte zu analysieren, sondern auch Schlüsse aus Änderungen, Löschungen und Neuerstellungen von Dateimetadaten und/oder Dateiinhalten zu ziehen.
- **Detektion von unautorisiertem Schreibzugriff**¹: Das System könnte erweitert werden, um unautorisierte Schreibzugriffe auf Dateien zu erkennen, einzuordnen und zu verhindern.

6.3.1.2 Generelle Ideen für zukünftige Features

- **Individuelle Monitoring Intervalle**: Der Anwender soll die Möglichkeit haben, die Periodizität des Monitorings pro überwachter Datei individuell einzustellen.
- **Monitoring von geschützten Verzeichnissen/Dateien**: Der Anwender soll die Möglichkeit haben (eventuell mit Erfordernissen via Betriebssystem), geschützte Verzeichnisse/Dateien zu überwachen.
- **Doppel-Backslash in Windows**: Wegen des Escapings von Java sind momentan in Windows-Pfaden doppelte Backslashes notwendig. Dies könnte mit einer Erweiterung im `ts.bat` Skript für Windows behoben werden.

6.3.2 User Experience

Der generelle technische Aufbau des Systems und insbesondere dessen Architektur bieten die Möglichkeit zusätzliche Clients zu entwickeln. Insbesondere eine grafische Benutzeroberfläche (GUI) könnte die Benutzerfreundlichkeit des Systems erheblich verbessern und die Zielgruppe erweitern. Die REST-Schnittstelle des Daemons bietet die notwendige Flexibilität dafür.

In Belangen der Benutzerfreundlichkeit wäre ausserdem eine Verbesserung der Fehlermeldungen (mit besserem Exceptionhandling) in Betracht zu ziehen.

¹Gemäss Originale Projektbeschreibung:

any sense of it being wiped or even deleted (and its recreation being prohibited through file access rights, which - en passant - may enable the detection of any unauthorised file writer)

6.3.3 Sicherheit

In der jetzigen Lösung wurde wenig bis gar keine Rücksicht auf Sicherheitsaspekte genommen. Das deshalb, weil das System als Ganzes nur auf dem Host des Anwenders läuft und keine externen Schnittstellen bietet. Es sind also primär Angriffe “von innen” möglich. Dabei ist vor allem die ungesicherte REST-Schnittstelle des Daemons anfällig. Das Hinzufügen von Authentifizierung und Autorisierung (v.a. der REST-Schnittstelle), wäre also ein wichtiger Schritt in einer zukünftigen Version.

6.3.4 Erweiterung der KI-Modelle bzw. Anbindung

Im Rahmen dieser Arbeit wurde nur ein einziges KI-Modell (GPT-4o mini) eingebunden. Dieses ist, obwohl nicht primär dafür entwickelt, durchaus in der Lage, die Anforderungen des TraceSentry zu erfüllen. In einer zukünftigen Version könnte die Anbindung von weiteren Modellen, die spezifisch für Anomalieerkennung in Dateien entwickelt wurden, in Betracht gezogen werden. Falls notwendig, könnte auch ein eigenes Modell entwickelt werden.

6.3.5 Dateikompatibilität

Bei der Inspektion von Dateien durch KI sowie dem Erstellen eines Merkle-Trees für die periodische Überwachung wird jeweils der gesamte Dateiinhalt via Java Files-API eingelesen. Die Funktion `readAllBytes` hat dabei eine Limitierung von 2 GB pro Datei [1]. Hierbei könnte man in einer zukünftigen Version die Dateien für die Inspektion in kleinere Teile aufteilen und diese separat analysieren. Für die Erstellung des Merkle-Trees wäre ein möglicher Ansatz, die Dateien in Blöcke von 2 GB zu teilen und diese einzeln zu hashen. Die Hashes aller Blöcke könnten dann konkateniert und gehasht werden, um den Hash-Wert der gesamten Datei zu erhalten.

Glossar

ACID Ein Akronym für Atomicity, Consistency, Isolation und Durability, Eigenschaften von Transaktionen in Datenbanksystemen. 22

API Application Programming Interface, eine Schnittstelle zur Kommunikation zwischen Software-Komponenten. 8

Cache-Dateien Zwischengespeicherte Dateien, die häufig von Anwendungen erstellt werden, um spezifische Daten auf einem System zu speichern, damit sie schneller abgerufen werden können. 11

CLI Command Line Interface, eine textbasierte Benutzerschnittstelle zur Interaktion mit Software. 8

CRUD Ein Akronym für Create, Read, Update und Delete, grundlegende Operationen auf Datenbanken. 27

CSV Comma-Separated Values, ein Dateiformat für tabellarische Daten. 59

Daemon Ein Hintergrundprozess, der Dienste bereitstellt oder Überwachungsaufgaben ausführt. 7

DBMS Database Management System, Software zur Verwaltung von Datenbanken. 8

FLOSS Free/Libre and Open Source Software, Software, die frei verfügbar und quelloffen ist. 65

GPT Generative Pre-trained Transformer, namensgebung für eine Reihe von KI-Modellen, die von OpenAI entwickelt wurden. 8

H2 Eine leichtgewichtige, relationale Datenbank, die in-memory verwendet werden kann. 23

HTTP Hypertext Transfer Protocol, ein Protokoll zur Übertragung von Daten im Web. 14

Java Die objektorientierte Programmiersprache, die in diesem Projekt verwendet wurde. 7

JRE Java Runtime Environment, eine Laufzeitumgebung für Java-Anwendungen. 66

KI-Modell Ein Modell, das Aufgaben mithilfe von künstlicher Intelligenz löst. 6

Log-Dateien Dateien, die Ereignisse oder Nachrichten von Software dokumentieren. 16

Maven Ein Build-Management-Tool für Java-Projekte. 65

Merkle-Tree Eine Hash-basierte Baumstruktur zur effizienten Überprüfung der Integrität von Verzeichnissen oder Dateien. 32

MIT Eine Lizenz für freie Software, die weit verbreitet ist und die Verwendung in kommerziellen Projekten erlaubt. 65

Monitoring Die periodische Überwachung von Systemen oder Dateien. 11

MVP Minimum Viable Product, die minimal funktionsfähige Version eines Produkts. 6

OpenAI Ein Unternehmen, das KI-Modelle wie GPT entwickelt. 8

PowerShell Eine plattformübergreifende Shell und Skriptsprache, entwickelt von Microsoft. 70

Prompt Eine Eingabeaufforderung oder eine Frage, die einer KI gegeben wird. 38

Regex-Pattern Ein regulärer Ausdruck zur Mustererkennung in Text. 35

REST Representational State Transfer, ein Architekturstil für Webservices bzw. APIs. 8

Shell Eine Schnittstelle, die die Interaktion mit dem Betriebssystem ermöglicht, oft über eine Kommandozeile. 9

Snapshot Ein Zustand eines Verzeichnisses oder einer Datei zu einem bestimmten Zeitpunkt. 30

Spring Ein Framework zur Entwicklung von Java-Anwendungen. 7

SQL Structured Query Language, eine Sprache zur Abfrage und Manipulation von Daten in relationalen Datenbanken. 22

SQLite Eine serverlose, eingebettete relationale Datenbank. 8

Token Repräsentiert eine Einheit von Text, die von einem GPT-Modell verarbeitet wird. 13

TraceSentry Der Name des Projekts, eine KI-gestützte Anwendung zur Überwachung von Cache- und Logdateien. 7

UX User Experience, das Erlebnis eines Benutzers mit einem Produkt oder einer Dienstleistung. 73

Index

Cache- und Logdateien, 5, 6, 11
CLI, 8, 28, 85

Daemon, 7, 9, 28, 69–71

KI-Modell, 5, 8, 58

Merkle-Tree, 32, 33, 36

Produktivitätssteigerung, 5, 74
Projektziel, 6

REST-API, 8, 37, 95

Scrum, 11, 40

Snapshots, 5, 11, 30, 33, 35

SQLite, 8, 22, 23, 35

Testing, 14, 44

Literaturverzeichnis

- [1] Oracle. *Java Files-API Dokumentation*: <https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html#readAllBytes-java.nio.file.Path->. Zuletzt aufgerufen am 10. Januar 2025.
- [2] Developer Apple. *Creating Launchd Jobs*. <https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html>, zuletzt aufgerufen am 10. Januar 2025.
- [3] GeeksforGeeks. *Introduction to Merkle Tree*. <https://www.geeksforgeeks.org/introduction-to-merkle-tree/>, zuletzt aufgerufen am 10. Januar 2025.
- [4] Wikipedia. *Merkle Tree*. https://en.wikipedia.org/wiki/Merkle_tree, zuletzt aufgerufen am 10. Januar 2025.
- [5] Baeldung. *Merkle Trees*. <https://www.baeldung.com/cs/merkle-trees>, zuletzt aufgerufen am 10. Januar 2025.
- [6] Spring Framework. *Spring Shell - Getting Started*. <https://docs.spring.io/spring-shell/reference/3.3/getting-started.html>, zuletzt aufgerufen am 10. Januar 2025.
- [7] McKinsey & Company. *The Economic Potential of Generative AI: The Next Productivity Frontier*. <https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier>, zuletzt aufgerufen am 10. Januar 2025.
- [8] OpenAI. *Models Documentation*. <https://platform.openai.com/docs/models>, zuletzt aufgerufen am 10. Januar 2025.

A Anhang

A.1 Originale Projektbeschreibung

AI-Aided Caches-n-Logs Monitoring-n-Wiping Demon

Description

The **goal (what)** of this project is to deliver a FLOSS-licensed, platform-independent piece of software (computer program), called the *Cache-n-Log Wiper* (not *Cache-n-Log Viper* ;-), that rocks-n-rolls by

1. monitoring the log and cache files in the (log and cache directories of the) file sytem of a computer;
2. invoking an AI-oracle so as to learn the purpose of each monitored file (what it is good or bad for) as well as any sense of it being wiped or even deleted (and its recreation being prohibited through file access rights, which - *en passant* - may enable the detection of any unauthorised file writer); and
3. wiping (emptying the content of) or even deleting these files, if both the AI and you consider it reasonable.

The **purpose (why)** of this project is

1. to empower computer users to take control of their own computer (to defend themselves against any surveillance by remote aliens who could be siphoning off confidential information for evil purposes ;-) and thus to take control of their own digital lives by contributing to their knowledge of what the hell is really going on in the entrails of their own computer (security mindset: reasonable paranoia ;-)
2. to experience the current potential of AI as a learning tool in a practical case study.

The code should be minimal, modular, and self-explaining.

The project report should be concise (maximally informative, minimally long). It must contain this project description as a quotation.

Technologies

Artificial Intelligence APIs, File Systems, Java

Advisor

Dr. Simon KRAMER (<https://www.simon-kramer.ch/Simon-Kramer.vcf>)

A.2 CLI Dokumentation

CLI Documentation

`ts` shorthand for TraceSentry

Commands

1. [run](#)
2. [status](#)
3. [kill](#)
4. [search](#)
5. [monitor add](#)
6. [monitor list](#)
7. [monitor remove](#)
8. [snapshots list](#)
9. [snapshots compare](#)
10. [inspect](#)
11. [wipe](#)

Enter `help` to get a list of all available commands. Or `help <command>` to get more information about a specific command.

1. run

This command starts the daemon process by specifying the path to the JAR file if it is not already running. If the daemon is already running, it will not be restarted. If the path is not provided, the command will attempt to infer the path from the environment variable `TRACE_SENTRY_DIR`.

Usage:

```
ts run <path>
```

Parameters:

- **path**: The full path to the daemon JAR file. Either the relative path from the current directory or the absolute path can be used. (Optional: default value is the composed path from the installation directory, as described above)

Examples:

```
ts run ./path/to/daemon.jar # relative path unix
ts run C:\\Path\\To\\Daemon.jar # absolute path windows
```

Success message:

```
Daemon started successfully.
```

Already running message:

```
Daemon is already running.
```

Error message:

```
Failed to start daemon.
```

2. status

This command checks the status of the daemon process.

Usage:

```
ts status
```

Example:

```
ts status
```

Output:

```
daemon is running
```

If the daemon is not running:

```
daemon is not running
```

3. kill

This command stops the daemon process if it is running.

Usage:

```
ts kill
```

Example:

```
ts kill
```

Success message:

```
daemon killed
```

If the daemon is not running:

```
daemon is not running
```

Other errors:

```
error killing daemon
```

4. search

This command initiates a search operation in the specified directory. The search mode can be customized, and subdirectory scanning is enabled by default unless the `--no-subdirs` flag is set. Additionally, a regex pattern can be used to filter files when the `pattern` mode is selected.

Usage:

```
ts search <path> [--mode <log|cache|full|pattern>] [--pattern <regex>] [--no-subdirs]
```

Parameters:

- **path** : The full path to the directory you want to search. Either the relative path from the current directory or the absolute path can be used.
- **--mode** : Defines the search mode:
 - **log** : Search for log files.
 - **cache** : Search for cache files.
 - **full** : Search for both log and cache files (default).
 - **pattern** : Search for files that match a custom regex pattern (requires **--pattern** parameter).
- **--pattern** : If the mode is set to **pattern**, this parameter defines the regular expression to match files.
- **--no-subdirs** : If this flag is present, the search will not include subdirectories (default is to search subdirectories).

Examples:

- Search a directory in full mode (default):

```
ts search /etc/path/to/directory # absolute path unix
ts search Path\\To\\Directory # relative path windows
```

- Search only for log files without scanning subdirectories:

```
ts search /path/to/directory --mode log --no-subdirs
```

- Search using a custom regex pattern:

```
ts search /path/to/directory --mode pattern --pattern ".*\\.log$"
```

Successful search:

```
Listing 2 files in /path/to/directory...
relative/path/from/search/directory/file1.log
relative/path/from/search/directory/file2.log
```

Error message:

```
Failed to search directory.
```

Notes:

- Ensure that the path you provide is absolute or relative to the current working directory.
- If no mode is specified, the search defaults to **full** mode.
- The **pattern** mode requires the **--pattern** parameter to define the regex for matching files.

5. monitor add

This command adds a specified path for monitoring. Paths added will be tracked for changes or new log/cache files.

Usage:

```
ts monitor add --path <path/to/monitor> [--mode <log|cache|full|pattern>] [--pattern <regex>] [--no-subdirs]
```

Parameters:

- **--path** : The full path you want to add for monitoring. This can be a relative path from the current directory or an absolute path. It might be a directory or a specific file.
- **--mode** : Defines the monitoring mode:
 - **log** : Monitor log files.
 - **cache** : Monitor cache files.
 - **full** : Monitor both log and cache files (default).
 - **pattern** : Monitor files that match a custom regex pattern (requires **--pattern** parameter).
- **--pattern** : If the mode is set to **pattern**, this parameter defines the regular expression to match files.
- **--no-subdirs** : If this flag is present, the monitoring will not include subdirectories (default is to monitor subdirectories).

Example:

- Add an absolute path to the monitoring database:

```
ts monitor add --path /etc/path/to/monitor # Unix
```

- Add a relative path to the monitoring database:

```
ts monitor add --path File\\To\\Monitor\\cache.log # Windows
```

Successful addition:

```
Successfully added /path/to/monitor to the monitoring database.
```

Exact path already monitored:

```
Error: /path/to/monitor is already being monitored.
```

6. monitor list

This command retrieves and displays all paths currently being monitored. Each entry shows the ID, path, and date of addition.

Usage:

```
ts monitor list
```

Examples:

- List all monitored paths:


```
ts monitor list
```

Example output:

```
+-----+-----+-----+-----+-----+
|ID|path|mode|pattern|no-subdirs|created at|
+-----+-----+-----+-----+-----+
|0001|C:\Users|FULL||false|2024-11-26|
+-----+-----+-----+-----+-----+
|0002|C:\Users\CoolDude|PATTERN|.*\cookie|true|2024-11-26|
+-----+-----+-----+-----+-----+
|0003|C:\Users\CoolDude\IdeaProjects\_bfh|PATTERN|env|false|2024-11-27|
+-----+-----+-----+-----+-----+
```

If no paths are being monitored:

```
No paths are currently being monitored.
```

7. monitor remove

This command removes a specified path from monitoring by its unique ID. The specified path is removed from the monitoring database.

Usage:

```
ts monitor remove --id <ID>
```

Parameters:

- **--id**: The unique identifier for the path to remove from monitoring. This ID can be obtained from the `monitor list` command.

Example:

- Remove a monitored path by ID:

```
ts monitor remove --id 3210
```

Successful removal:

```
Successfully removed path with ID 3210 from the monitoring database.
```

Error message if the ID does not exist:

```
Error: No monitored path found with ID 3210.
```

8. snapshots list

This command lists all snapshots taken for a monitored path.

Usage:

```
ts snapshots list --id <ID>
```

Parameters:

- **--id** : The unique identifier for the monitored path to get the snapshots from.

Example:

- Get the snapshots of a monitored path by ID:

```
ts snapshots list --id 1
```

Example output of successful execution:

```
+-----+-----+---+
|Number|Timestamp      |ID|
+-----+-----+---+
|1      |18.12.2024 22:42:01|2 |
+-----+-----+---+
|2      |18.12.2024 22:41:15|1 |
+-----+-----+---+
```

Example outputs of unsuccessful execution:

If the path is not monitored:

```
No monitored path found with ID [id].
```

If no snapshots are available:

```
No snapshots found for monitored path with ID [id].
```

Other:

```
Error: could not list snapshots.
```

9. snapshots compare

This command outputs the comparison of a user-defined range of snapshots taken from a given monitored path. The parameters **start** and **end** refer to the **Number** column in the output of the **snapshots list** command.

Usage:

```
ts snapshots compare --id <ID> --start <startNumber> --end <endNumber>
```

Parameters:

- **--id** : The unique identifier for the monitored path to get the comparison from.
- **--start** : Snapshot-number (begin at 1 and snapshots are ordered from latest to oldest) to start the comparison from. (Optional: default value is 1)
- **--end** : Snapshot-number (begin at 2 and snapshots are ordered from latest to oldest) to end the comparison at. (Optional: default value is 2)

Example:

- Get the comparison of a monitored path by ID and of the last 5 taken snapshots:

```
ts snapshots compare --id 1 --start 1 --end 5
```

Example output of successful execution:

Listing comparison of /test from 01.12.2024 15:30:00 to 01.12.2024 17:30:00...

Path	Snapshot IDs	Comparison
cache.txt	4	CHANGED
	6	LAST TRACK
log/log.txt	2	CHANGED
	8	CHANGED

Example outputs of unsuccessful execution:

If the range is too big for the existing snapshots:

Error: Not enough snapshots existing at the moment for this range.

If the index range is not valid:

Error: Start index needs to be smaller than the end index and not negative.

If it is an unexpected error:

Error: could not compare snapshots.

10. inspect

This command sends the contents of the file to the OpenAI API for analysis. Ensure that no sensitive, confidential, or private data is included in the file before proceeding. By using this command, you agree to the terms and conditions of the OpenAI API.

This command allows you to inspect a specific file. The output comes directly from an OpenAI AI-model. The `OPENAI_API_KEY` environment variable containing the API key is required so that this can be queried.

Usage:

```
ts inspect <path>
```

Parameters:

- **path**: The full path to the file you want to inspect. Either the relative path from the current directory or the absolute path can be used.

Examples:

- Inspect a file:

```
ts inspect /path/to/apache.log
```

Successful inspection:

Intended use:

The file `apache.log` appears to be a web server access log, capturing various HTTP requests [...] not valid IP addresses and may indicate malicious activity or probing attempts. [...] the abnormal entries warrant further investigation.

Assessment:

Potentially harmful

Recommended to Wipe:

Clear file

Additional recommendations:

Perform a thorough analysis of the web server's security posture and monitor for unusual traffic patterns. Validate legitimate access attempts to identify any potential breaches or vulnerabilities.

File not found:

Error: File not found.

Error message:

Error: Failed to inspect file. Make sure the file is accessible, the connection to the internet is working and the environment variable OPENAI_API_KEY is set.

11. wipe

This command wipes a given file. This either means to clear the content of the file or to delete it. Which can be specified by the `remove` flag.

Usage:

```
ts wipe <path> [--remove]
```

Parameters:

- **path** : The full path to the file you want to wipe. Either the relative path from the current directory or the absolute path can be used.
- **--remove** : If this flag is present, the file will be deleted. Otherwise, the content will be cleared.

Examples:

- Wipe the content of a file:

```
ts wipe /path/to/cool.log
```

- Remove a file:

```
ts wipe /path/to/cool.log --remove
```

Successful clearing:

Successfully cleared file.

Successful removal:

Successfully removed file.

File not processable:

Error: File could not be processed.

Error message:

Error: Failed to wipe file.

A.3 API-Dokumentation

AI-Aided Caches n Logs Monitoring n Wiping Daemon API

Overview

A command-line tool for monitoring and wiping cache and log files.

Paths

GET /search Searches the specified directory for specific files

Parameters

Type	Name	Description	Schema
query	mode <i>optional</i>	Specifies the search mode (log, cache, full, pattern). Default is full.	enum (log,cache,full,pattern)
query	path <i>required</i>	The full path to the directory you want to search.	string
query	pattern <i>optional</i>	Specifies the regex pattern to match files (required if mode is pattern).	string
query	no-subdirs <i>optional</i>	Prevents searching subdirectories (default is false).	boolean

Responses

Code	Description	Links									
200	<p>Search completed successfully</p> <p><i>Content</i></p> <p>application/json</p> <p><i>Properties</i></p> <table><tr><th>Name</th><th>Description</th><th>Schema</th></tr><tr><td>numberOfFiles <i>optional</i></td><td></td><td>integer</td></tr><tr><td>files <i>optional</i></td><td></td><td>< string > array</td></tr></table>	Name	Description	Schema	numberOfFiles <i>optional</i>		integer	files <i>optional</i>		< string > array	No Links
Name	Description	Schema									
numberOfFiles <i>optional</i>		integer									
files <i>optional</i>		< string > array									
400	Invalid input	No Links									

Code	Description	Links
422	Not processable input	No Links
500	Internal server error	No Links

GET /monitored-path Returns all monitored paths.

Responses

Code	Description	Links
200	Returns all monitored paths. <i>Content</i> application/json	No Links

POST /monitored-path Adds a path to the monitoring database.

Responses

Code	Description	Links
201	Path successfully added to the monitoring database.	No Links
400	Invalid input.	No Links
422	Invalid or non-existent path.	No Links
409	Path already exists in the monitoring database.	No Links
500	Internal server error.	No Links

DELETE /monitored-path/{id} Removes a path from the monitoring database.

Parameters

Type	Name	Description	Schema
path	id <i>required</i>	The ID of the path you want to remove.	integer

Responses

Code	Description	Links
204	Path successfully removed from the monitoring database.	No Links
404	No monitored path with the specified ID found.	No Links

GET /monitored-path/{id}/snapshots/changes Returns a comparison of the monitored path.

Parameters

Type	Name	Description	Schema
query	start <i>optional</i>	The start index for the comparison range.	integer
query	end <i>optional</i>	The end index for the comparison range.	integer
path	id <i>required</i>	The ID of the monitored path you want the comparison from.	integer

Responses

Code	Description	Links
200	Returns the comparison. <i>Content</i> application/json	No Links
422	Not a valid index range or not found enough snapshots in the given index range.	No Links

GET /monitored-path/{id}/snapshots Returns all snapshots of the monitored path.

Parameters

Type	Name	Description	Schema
path	id <i>required</i>	The ID of the monitored path you want the snapshots from.	integer

Responses

Code	Description	Links
200	Returns all snapshots. <i>Content</i> application/json	No Links
404	The path with the specified ID is not being monitored.	No Links

GET /inspect Inspects a given file by querying a GPT service.

Parameters

Type	Name	Description	Schema
query	path <i>required</i>	The full path to the file you want to inspect.	string

Responses

Code	Description	Links
200	Inspection completed successfully <i>Content</i> application/json	No Links
400	Invalid input	No Links
422	Not processable input	No Links
500	Internal server error	No Links

POST /wipe Wipes the content of a file or deletes it.

Responses

Code	Description	Links
200	File successfully wiped or deleted. <i>Content</i> application/json	No Links
400	Invalid input.	No Links
422	File not found or not writable.	No Links
500	Internal server error.	No Links

Components

Schemas

MonitoredChangesDTO

Properties

Name	Description	Schema
monitoredPath <i>optional</i>		string
startSnapshotC reation <i>optional</i>		string
endSnapshotCr eation <i>optional</i>		string
comparison <i>optional</i>		< SnapshotComp arisonDTO > array

SnapshotComparisonDTO

Properties

Name	Description	Schema
snapshotIds <i>optional</i>		< integer > array
path <i>optional</i>		string
comparison <i>optional</i>		< string > array

MonitorPath

Properties

Name	Description	Schema
id <i>optional</i>		integer
path <i>optional</i>		string
createdAt <i>optional</i>		string (date)

SnapshotDTO

Properties

Name	Description	Schema
id <i>optional</i>		integer
timestamp <i>optional</i>		string (date- time)

A.4 Klassendiagramm CLI

-
- Falls das Diagramm nicht optimal dargestellt wird, kann es im Abgabeordner unter folgenden Pfaden gefunden werden: `docs/diagrams/intellij/cli.uml` (kann nur mit IntelliJ IDEA geöffnet werden) bzw. `docs/assets/cli_class_diag.png`
 - Wegen dem Spring IoC Container: nicht alle Beziehungen zwischen Klassen abgebildet.
 - Diagramm erstellt mit IntelliJ IDEA Ultimate.

A.5 Klassendiagramm Daemon

-
- Falls das Diagramm nicht optimal dargestellt wird, kann es im Abgabeordner unter folgenden Pfaden gefunden werden: `docs/diagrams/intellij/daemon.uml` (kann nur mit IntelliJ IDEA geöffnet werden) bzw. `docs/assets/daemon_class_diag.png`
 - Wegen dem Spring IoC Container: nicht alle Beziehungen zwischen Klassen abgebildet.
 - Diagramm erstellt mit IntelliJ IDEA Ultimate.

<div><div><div></div></div><div>SnapshotController</div></div>	<div><div><div></div></div><div>SnapshotDomainService</div></div>
<div><div><div></div></div><div><div><div>snapshotDomainService</div></div><div><div>getSnapshot(Integer)</div></div><div><div>getMonitoredChanges(Integer, Integer, Integer)</div></div><div><div>monitoredChangesDTO</div></div></div></div>	<div><div><div></div></div><div><div><div>SnapshotDomainService</div></div><div><div>snapshotComparatorRepository</div></div><div><div>monitoredPathRepository</div></div><div><div>snapshotRepository</div></div><div><div>getSnapshotDTO(Integer)</div></div><div><div>getSnapshotComparator(Integer, Integer, Integer)</div></div><div><div>listSnapshotComparatorDTOs</div></div><div><div>getMonitoredPath(Integer)</div></div><div><div>monitoredPath</div></div></div></div>
<div><div><div></div></div><div><div><div>modelMapper</div></div><div><div>snapshotComparatorRepository</div></div><div><div>monitoredPathRepository</div></div><div><div>snapshotRepository</div></div><div><div>getSnapshot(Integer)</div></div><div><div>getSnapshotComparator(Integer, Integer, Integer)</div></div><div><div>listSnapshotComparatorDTOs</div></div><div><div>getMonitoredPath(Integer)</div></div><div><div>monitoredPath</div></div></div></div>	<div><div><div></div></div><div><div><div>ModelMapper</div></div><div><div>SnapshotComparatorRepository</div></div><div><div>MonitoredPathRepository</div></div><div><div>SnapshotRepository</div></div><div><div>List<SnapshotDTO></div></div><div><div>List<SnapshotComparator(Integer, Integer, Integer)></div></div><div><div>List<SnapshotComparatorDTO></div></div><div><div>List<MonitoredPath></div></div></div></div>

<div><div><div></div></div><div>SnapshotRepository</div></div>
<div><div><div></div></div><div><div><div>findAllByMonitoredPathOrderByTimestampDesc(Integer)</div></div><div><div>findAllByMonitoredPathOrderByTimestampDesc(Integer)</div></div><div><div>Optional<Snapshot></div></div></div></div>

<div><div><div></div></div><div>SnapshotComparatorRepository</div></div>
<div><div><div></div></div><div><div><div>entityManager</div></div><div><div>getSnapshotComparator(Integer, Integer, Integer)</div></div><div><div>List<SnapshotComparatorDTO></div></div></div></div>

<div><div><div></div></div><div>Snapshot</div></div>
<div><div><div></div></div><div><div><div>id</div></div><div><div>timestamp</div></div><div><div>monitoredPath</div></div></div></div>
<div><div><div></div></div><div><div><div>getId()</div></div><div><div>setMonitoredPath(MonitoredPath) void</div></div><div><div>getTimestamp() Timestamp</div></div><div><div>set(Integer) void</div></div><div><div>getMonitoredPath() MonitoredPath</div></div><div><div>setTimestamp(Timestamp) void</div></div></div></div>

<div><div><div></div></div><div>NodeRepository</div></div>
<div><div><div></div></div><div><div><div>findAllBySnapshotId(Integer)</div></div><div><div>List<Node></div></div></div></div>
<div><div><div></div></div><div><div><div>Node</div></div></div></div>
<div><div><div></div></div><div><div><div>snapshot</div></div><div><div>path</div></div><div><div>id</div></div><div><div>children</div></div><div><div>hash</div></div><div><div>parent</div></div><div><div>deletedInNextSnapshot</div></div><div><div>hasChanged</div></div><div><div>setHash(String)</div></div><div><div>getParent()</div></div><div><div>getHash()</div></div><div><div>getChilder()</div></div><div><div>isHasChanged()</div></div><div><div>getSnapshot()</div></div><div><div>setSnapshot(Snapshot) void</div></div><div><div>setHasChanged(boolean) void</div></div><div><div>isParent()</div></div><div><div>setDeletedInNextSnapshot(boolean) void</div></div><div><div>setChilder(List<Node>) void</div></div><div><div>isDeletedInNextSnapshot()</div></div><div><div>getParent()</div></div><div><div>getHash()</div></div><div><div>set(Integer) void</div></div><div><div>getId()</div></div><div><div>setPath(String) void</div></div></div></div>

<div><div><div></div></div><div>MonitoringController</div></div>	<div><div><div></div></div><div>MonitoringDomainService</div></div>
<div><div><div></div></div><div><div><div>monitoringDomainService</div></div><div><div>getMonitoredPath()</div></div><div><div>dataMonitoring(Integer)</div></div><div><div>createMonitoring(CreateMonitorPathDTO)</div></div><div><div>monitoredPathDTO</div></div><div><div>void</div></div><div><div>void</div></div></div></div>	<div><div><div></div></div><div><div><div>Logger</div></div><div><div>ModelMapper</div></div><div><div>monitoredPathRepository</div></div><div><div>monitoredPathRepository</div></div><div><div>createMonitoring(String, SearchMode, Pattern, boolean) void</div></div><div><div>deleteMonitoring(Integer)</div></div><div><div>getMonitoredPath()</div></div><div><div>List<MonitoredPathDTO></div></div></div></div>

<div><div><div></div></div><div>MonitoredPathRepository</div></div>
<div><div><div></div></div><div><div><div>findByPath(String)</div></div><div><div>Optional<MonitoredPath></div></div><div><div>existsByPath(Integer)</div></div><div><div>existsByPath(String)</div></div><div><div>boolean</div></div><div><div>boolean</div></div></div></div>

<div><div><div></div></div><div>MonitoredPath</div></div>
<div><div><div></div></div><div><div><div>path</div></div><div><div>id</div></div><div><div>noSubdirs</div></div><div><div>createdAt</div></div><div><div>mode</div></div><div><div>getMode()</div></div><div><div>setPattern(String)</div></div><div><div>noSubdir(boolean)</div></div><div><div>isNoSubdir()</div></div><div><div>path(String)</div></div><div><div>model(SearchMode)</div></div><div><div>completePattern()</div></div><div><div>pattern(String?)</div></div><div><div>getCreatedAt()</div></div><div><div>getId()</div></div><div><div>getParent()</div></div><div><div>createdAt(LocalDate)</div></div><div><div>setNoSubdir(boolean)</div></div><div><div>set(Integer)</div></div><div><div>getPattern()</div></div><div><div>setModel(SearchMode)</div></div></div></div>
<div><div><div></div></div><div><div><div>String</div></div><div><div>Integer</div></div><div><div>String?</div></div><div><div>boolean</div></div><div><div>LocalDate</div></div><div><div>SearchMode</div></div><div><div>void</div></div><div><div>MonitoredPath</div></div><div><div>boolean</div></div><div><div>MonitoredPath</div></div><div><div>Pattern</div></div><div><div>MonitoredPath</div></div><div><div>LocalDate</div></div><div><div>Integer</div></div><div><div>String</div></div><div><div>MonitoredPath</div></div><div><div>void</div></div><div><div>void</div></div><div><div>void</div></div><div><div>String</div></div><div><div>void</div></div></div></div>

<div><div><div></div></div><div>MonitoringScheduler</div></div>
<div><div><div></div></div><div><div><div>snapshotRepository</div></div><div><div>monitoredPathRepository</div></div><div><div>nodeRepository</div></div><div><div>LOG</div></div><div><div>createSnapshot()</div></div><div><div>compareWithOldSnapshot(MerkleTree, List<Node>) void</div></div><div><div>createSnapshot(MonitoredPath) void</div></div><div><div>markDeletedNode(List<Node>, MerkleTree) void</div></div><div><div>compareNode(Node, List<Node>) void</div></div></div></div>

<div><div><div></div></div><div>MerkleTree</div></div>
<div><div><div></div></div><div><div><div>linearizedNodes</div></div><div><div>monitoredPath</div></div><div><div>root</div></div><div><div>LOG</div></div><div><div>getMonitoredPath()</div></div><div><div>get()</div></div><div><div>buildTreeRecursive(Node, Snapshot, File?)</div></div><div><div>buildMerkleTree(MonitoredPath, Snapshot) void</div></div><div><div>combineHashes(List<Node>) String</div></div><div><div>getDirectoryHash(String, File?) String</div></div><div><div>createNodeOrDirector(File, Snapshot, File?) Node</div></div><div><div>calculateSplit(Node) int</div></div><div><div>hash(String) String</div></div><div><div>createNodeOrFile(File, Snapshot) Node</div></div><div><div>getLinearizedNodes()</div></div><div><div>List<Node></div></div></div></div>

<div><div><div></div></div><div>GPTMessage</div></div>
<div><div><div></div></div><div><div><div>content</div></div><div><div>role</div></div><div><div>setContent(String) void</div></div><div><div>getRole()</div></div><div><div>setRole(String) void</div></div><div><div>getContent()</div></div><div><div>String</div></div></div></div>

<div><div><div></div></div><div>TraceSentryException</div></div>
<div><div><div></div></div><div><div><div>status</div></div><div><div>getResponse()</div></div><div><div>HttpStatus</div></div><div><div>ErrorResponse</div></div></div></div>

<div><div><div></div></div><div>InternalServerErrorException</div></div>	<div><div><div></div></div><div>BadRequestException</div></div>	<div><div><div></div></div><div>ConflictException</div></div>	<div><div><div></div></div><div>UnprocessableException</div></div>
--	---	---	--

<div><div><div></div></div><div>GlobalExceptionHandler</div></div>
<div><div><div></div></div><div><div><div>handleException(TraceSentryException)</div></div><div><div>ResponseEntity<ErrorResponse></div></div></div></div>

<div><div><div></div></div><div>InspectController</div></div>
<div><div><div></div></div><div><div><div>inspectionService</div></div><div><div>inspectionService</div></div><div><div>search(String)</div></div><div><div>String</div></div></div></div>

<div><div><div></div></div><div>InspectionService</div></div>
<div><div><div></div></div><div><div><div>inspect(String, String)</div></div><div><div>String</div></div></div></div>

<div><div><div></div></div><div>GPTInspectionService</div></div>
<div><div><div></div></div><div><div><div>objectMapper</div></div><div><div>OPENAPIURL</div></div><div><div>restTemplate</div></div><div><div>getPrompt(String, String)</div></div><div><div>inspect(String, String)</div></div><div><div>prepareRequest(String) HttpEntity<String></div></div></div></div>

<div><div><div></div></div><div>GPTRequest</div></div>
<div><div><div></div></div><div><div><div>messages</div></div><div><div>DEFAULT_MODEL</div></div><div><div>model</div></div><div><div>getModel()</div></div><div><div>getMessage()</div></div><div><div>List<GPTMessage></div></div><div><div>String</div></div><div><div>String</div></div><div><div>List<GPTMessage></div></div></div></div>

<div><div><div></div></div><div>SearchController</div></div>
<div><div><div></div></div><div><div><div>search(String, String, String, boolean)</div></div><div><div>SearchResponseDTO</div></div></div></div>

<div><div><div></div></div><div>SearchStrategyFactory</div></div>
<div><div><div></div></div><div><div><div>create(SearchMode, Pattern)</div></div><div><div>SearchStrategy</div></div></div></div>

<div><div><div></div></div><div>SearchStrategy</div></div>
<div><div><div></div></div><div><div><div>matches(Path)</div></div><div><div>boolean</div></div></div></div>

<div><div><div></div></div><div>SubStringsSearchStrategy</div></div>
<div><div><div></div></div><div><div><div>substrings</div></div><div><div>matches(Path)</div></div><div><div>List<String></div></div><div><div>boolean</div></div></div></div>

<div><div><div></div></div><div>PatternSearchStrategy</div></div>
<div><div><div></div></div><div><div><div>pattern</div></div><div><div>matches(Path)</div></div><div><div>Pattern</div></div><div><div>boolean</div></div></div></div>

<div><div><div></div></div><div>CacheSearchStrategy</div></div>

<div><div><div></div></div><div>LogSearchStrategy</div></div>

<div><div><div></div></div><div>Constants</div></div>
<div><div><div></div></div><div><div><div>DEVON_MODULE_NAME</div></div><div><div>LOG_SEARCH_STRING</div></div><div><div>IS_ORIENT_VARIABLE</div></div><div><div>SYSTEM_PROMPT</div></div><div><div>DB_NAME</div></div><div><div>CACHE_SEARCH_STRING</div></div><div><div>JAR_EXTENSION</div></div></div></div>

<div><div><div></div></div><div>SQLiteConfig</div></div>
<div><div><div></div></div><div><div><div>password</div></div><div><div>url</div></div><div><div>username</div></div><div><div>driverClassName</div></div><div><div>dataSource() DataSource</div></div><div><div>getSqlTest(Path) String</div></div></div></div>

<div><div><div></div></div><div>AppConfig</div></div>
<div><div><div></div></div><div><div><div>restTemplate()</div></div><div><div>RestTemplate</div></div></div></div>

<div><div><div></div></div><div>ModelMapperConfig</div></div>
<div><div><div></div></div><div><div><div>modelMapper()</div></div><div><div>ModelMapper</div></div></div></div>

<div><div><div></div></div><div>DemonApplication</div></div>
<div><div><div></div></div><div><div><div>mail(StringID)</div></div><div><div>void</div></div></div></div>

A.6 Klassendiagramm Lib

-
- Falls das Diagramm nicht optimal dargestellt wird, kann es im Abgabeordner unter folgenden Pfaden gefunden werden: `docs/diagrams/intellij/lib.uml` (kann nur mit IntelliJ IDEA geöffnet werden) bzw. `docs/assets/lib_class_diag.png`
 - Diagramm erstellt mit IntelliJ IDEA Ultimate.

© ↗ ErrorResponse

Ⓢ Ⓜ status

int

Ⓢ Ⓜ message

String

© ↗ WipeFileDTO

Ⓢ Ⓜ path

String

Ⓢ Ⓜ remove

boolean

© ↗ SnapshotDTO

Ⓢ Ⓜ id

Integer

Ⓢ Ⓜ timestamp

LocalDateTime

© ↗ SearchResponseDTO

Ⓢ Ⓜ numberOfFiles

int

Ⓢ Ⓜ files

List<String>

Ⓔ ↗ SearchMode

Ⓢ Ⓜ LOG

Ⓢ Ⓜ PATTERN

Ⓢ Ⓜ CACHE

Ⓢ Ⓜ FULL

© ↗ SnapshotComparisonDTO

Ⓢ Ⓜ comparison

List<String>

Ⓢ Ⓜ snapshotIds

List<Integer>

Ⓢ Ⓜ path

String

© ↗ MonitoredPathDTO

Ⓢ Ⓜ id

Integer

Ⓢ Ⓜ mode

SearchMode

Ⓢ Ⓜ path

String

Ⓢ Ⓜ createdAt

LocalDate

Ⓢ Ⓜ noSubdirs

boolean

Ⓢ Ⓜ pattern

String

© ↗ CreateMonitorPathDTO

Ⓢ Ⓜ mode

SearchMode

Ⓢ Ⓜ pattern

String

Ⓢ Ⓜ noSubdirs

boolean

Ⓢ Ⓜ path

String

© ↗ MonitoredChangesDTO

Ⓢ Ⓜ monitoredPath

String

Ⓢ Ⓜ endSnapshotCreation

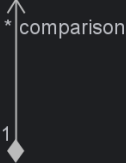
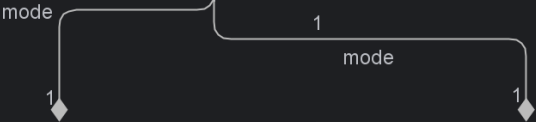
LocalDateTime

Ⓢ Ⓜ startSnapshotCreation

LocalDateTime

Ⓢ Ⓜ comparison

List<SnapshotComparisonDTO>






A.7 Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter angefertigt haben.

Alle verwendeten Quellen und Inhalte sind vollständig und ordnungsgemäss angegeben. Wörtlich oder inhaltlich übernommene Stellen sind als solche gekennzeichnet.

Wir erklären uns damit einverstanden, dass die vorliegende Arbeit in elektronischer Form mit geeigneter Software überprüft werden darf.

17. Januar 2025

		
_____ J. Scherer	_____ L. Scherer	_____ L. Ammann