

IU INTERNATIONALE HOCHSCHULE

MASTER INFORMATIK

MODUL DLMCSPSE01\_D

PROJEKT: SOFTWARE ENGINEERING

# Threat Intelligence Plattform

Johannes Liebscher

Matrikelnummer: 32209071

Prof. Dr.

Dirk SIMON

12. Januar 2026

# Inhaltsverzeichnis

<b>1 Projektdokumentation</b>	<b>8</b>
1.1 Projektübersicht . . . . .	8
1.2 Projektziele . . . . .	9
1.3 Risikomanagement . . . . .	9
1.4 Zeitplan . . . . .	11
1.5 Meilensteine . . . . .	11
1.6 Projektstrukturplan . . . . .	12
1.7 Entwicklung . . . . .	17
<b>2 Anforderungsdokument</b>	<b>29</b>
2.1 Stakeholder . . . . .	29
2.2 Funktionale Anforderungen . . . . .	31
2.3 Nicht-funktionale Anforderungen . . . . .	36
<b>3 Spezifikationsdokument</b>	<b>39</b>
3.1 Datenmodell . . . . .	39
3.2 Geschäftsprozesse . . . . .	44
3.3 Geschäftsregeln . . . . .	46
3.4 Systemschnittstellen . . . . .	47
3.5 Benutzerschnittstellen . . . . .	48
<b>4 Architekturdokument</b>	<b>51</b>
4.1 Technologieübersicht . . . . .	51
4.2 Architekturübersicht . . . . .	54
4.3 Struktur . . . . .	57
4.4 Verhalten . . . . .	67
<b>Literaturverzeichnis</b>	<b>70</b>
<b>Abbildungsverzeichnis</b>	<b>74</b>
<b>Tabellenverzeichnis</b>	<b>75</b>

<b>5</b>	<b>Anlagen</b>	<b>76</b>
5.1	Stakeholderanalyse . . . . .	76
5.2	JSON Schema . . . . .	78
5.3	Sequenzdiagramme Querformat . . . . .	81
5.4	Zertifikate . . . . .	84

# Glossar

API	Application Programming Interface, Schnittstelle, über die Softwarekomponenten miteinander kommunizieren.	53
Backend	Serverseitige Anwendung, welche Datenverarbeitung und Datenbankzugriffe bereitstellt.	8, 46, 49
CISO	Der Chief Information Security Officer ist die Verantwortliche Person für Informationssicherheit in einer Organisation.	32
CORS	Cross-Origin Resource Sharing (CORS) ist ein Sicherheitsmechanismus von Webbrowsern, der festlegt, ob Webanwendungen Ressourcen von einem anderen Ursprung (Origin) abrufen dürfen. In Flask-Anwendungen wird CORS verwendet, um den Zugriff von Frontends auf Backend-APIs zu erlauben oder zu beschränken.	54
CRUD	Abkürzung für Create, Read, Update, Delete. Dies sind die grundlegenden Operationen zur Datenmanipulation.	14
CVE	Common Vulnerabilities and Exposures ist ein System zur Identifikation und Benennung von öffentlich bekannten Sicherheitslücken und anderen Schwachstellen. Es wird vom US-amerikanischen National Cybersecurity FFRDC betrieben und von der Mitre Corporation gepflegt.	44
CVSS	Das Common Vulnerability Scoring System ist ein Industriestandard zur Bewertung des Schweregrades von Sicherheitslücken in Computer-Systemen.	44
Cypher	Abfragesprache für die Neo4j-Datenbank zur Manipulation und Analyse von Graphdaten.	49, 53

cytoscape.js	Cytoscape ist eine open source Bibliothek für die Analyse und Visualisierung von Graphendaten. Sie ist in Java Script geschrieben.	55
DOM	Das Document Object Model (DOM) ist eine baumartige Repräsentation eines HTML- oder XML-Dokuments im Browser. Es ermöglicht Skriptsprachen wie JavaScript, Struktur, Inhalt und Darstellung einer Webseite dynamisch zu lesen und zu verändern.	55
Frontend	Benutzerschnittstelle eines Softwaresystems und Gegenstück zum Backend. Es wird hier im Browser realisiert.	8, 46, 49
IoC	Indicator of Compromise, technische Spur eines Angriffs, beispielsweise IP-Adresse, Datei-Hash oder Domain, die auf eine Kompromittierung hinweist.	9
JSON	JSON (JavaScript Object Notation) ist ein kompaktes, textbasiertes Format zur Darstellung strukturierter Daten. Es wird verwendet, um Daten zwischen Server und Client auszutauschen. JSON ist lesbar für Menschen und maschinenverarbeitbar. Es basiert auf der Darstellung von Objekten durch Schlüssel-Wert-Paare und unterstützt unter anderem Listen, Zahlen, Zeichenketten, Booleans und Null-Werte.	48, 49, 53, 54, 61, 66
Kampagne	Eine koordinierte Folge von Angriffsaktivitäten eines Threat Actors gegen ein oder mehrere Ziele.	9
Malware	Bösartige Software, die Systeme schädigt, kompromittiert oder kontrolliert, beispielweise Trojaner, Ransomware oder Viren.	9, 34
MITRE ATT&CK	Wissensbasis über bekannte Angriffsvektoren und Methoden von Cyberangreifern, entwickelt durch MITRE.	41

Neo4j	Eine auf Graphen basierende NoSQL-Datenbank, die Entitäten und deren Beziehungen speichern und abfragen kann.	46, 49
Props	In Svelte sind Props (kurz für Properties) ein Mechanismus, um Daten unidirektional und von oben nach unten von einer übergeordneten Komponente an eine untergeordnete Komponente zu übergeben.	61, 62
REST	REST (Representational State Transfer) ist ein Architekturstil für Webservices, der auf den HTTP-Standard aufbaut. Er ermöglicht den Zugriff auf Ressourcen (Daten) über standardisierte HTTP-Methoden wie GET, POST, PUT und DELETE. REST zeichnet sich durch eine stateless Kommunikation aus, das heißt, jeder Request enthält alle Informationen, die der Server benötigt, um ihn zu verarbeiten.	49
Singleton	Das Singleton ist ein Entwurfsmuster, das sicherstellt, dass eine Klasse nur eine einzige Instanz besitzt und einen globalen Zugriffspunkt auf diese Instanz bereitstellt.	68
STIX	Structured Threat Information Expression ist eine Standard zum Ausdrücken von Threat Intelligence Daten, Cyberbedrohungen und beobachtbaren Informationen. Die zu diesem Zeitpunkt aktuelle Version ist STIX Version 1.2 (Struse und Darley, 2019).	41
String Concatenation	String-Konkatenation bezeichnet das direkte Zusammensetzen von Zeichenketten, etwa bei der Erzeugung von SQL- oder HTML-Ausdrücken. Im Kontext der Input-Sanitisierung gilt sie als unsicher, da ungeprüfte Benutzereingaben zu Injections wie SQL- oder XSS-Angriffen führen können.	67

TDD	Test Driven Development ist ein Entwicklungsansatz, bei dem Tests vor der Implementierung geschrieben werden, um Code-Qualität und Design zu verbessern. Dieser läuft ab in drei Stufen: 1. Schreiben des Tests; 2. Schreiben des Code, sodass der Test erfolgreich ist; 3. Refactoring des Codes	26, 29
Threat Actor	Ein individueller oder kollektiver Angreifender, der bössartige Aktivitäten, absichtlich Bedrohungen verursacht.	9, 33, 41–44
Threat Intelligence	Systematische Sammlung, Analyse und Bewertung von Informationen über Bedrohungen und Angreifende mit dem Ziel, IT-Infrastrukturen proaktiv zu schützen.	8, 33–36, 38, 41, 43, 53, 55, 58
TTPs	Abkürzung für <i>Tactics, Techniques and Procedures</i> , typische Verhaltensmuster und Methoden von Angreifenden.	33, 34, 46
WSGI	Das Web Server Gateway Interface (WSGI) ist eine standardisierte Schnittstelle zwischen Python-Webanwendungen und Webservern. Flask basiert auf WSGI.	54

# 1 Projektdokumentation

Cyberangriffe sind eine Herausforderung für Unternehmen, Organisationen und öffentliche Einrichtungen. Um sich wirksam gegen Angriffe schützen zu können, benötigen Entscheidungsträger und Sicherheitsexperten Informationen über aktuelle Bedrohungen, deren Herkunft und Vorgehensweise, so genannte Threat Intelligence Daten. Aus diesen Informationen müssen die aktuell relevanten Zusammenhänge erarbeitet werden. Eine Graphendatenbank kann die Struktur dieser Daten abbilden. Die Analyse kann visuell über eine interaktive Benutzerschnittstelle durchgeführt werden. Ein Bericht fasst die Ergebnisse für die weitere Verwendung zusammen. Eine solches System wird mit diesem Projekt umgesetzt.

## 1.1 Projektübersicht

Ziel des Projekts ist es, im Rahmen des Kurses Projekt: Software Engineering ein individuelles studentisches Experiment durchzuführen. Im Zentrum steht die Erprobung technischer Möglichkeiten zur Konzeption, Umsetzung und Dokumentation einer funktionalen Plattform zur Analyse von Bedrohungsdaten (Cyber Threat Intelligence Plattform). Im Rahmen dieses Projekts wird eine Webanwendung entwickelt, die Cyber Threat Intelligence Daten grafisch visualisiert. Die Daten werden in einer Graphendatenbank gespeichert und über eine Serverarchitektur aus Backend und Frontend verarbeitet. Das Projekt gliedert sich in drei Phasen:

- **Konzeptionsphase:**

Erstellung der Projektdokumentation, des Anforderungsdokuments und des Spezifikationsdokuments.

- **Erarbeitungs- und Reflexionsphase:**

Umsetzung der Softwarearchitektur, Implementierung des Programmcodes, Dokumentation, Erstellung von beschreibenden UML-Diagrammen.

- **Finalisierungsphase:**

Erstellen des Testdokuments, durchführen von Qualitätssicherung, Abstract zur Reflexion.

## 1.2 Projektziele

### Modellierung von Entitäten und Beziehungen

Ziel ist die Entwicklung eines Graphendatenmodells, das zentrale Entitäten aus dem Bereich der Cyber Threat Intelligence, wie beispielsweise Threat Actor, IoC, Malware und Kampagne, sowie deren Beziehungen abbildet. Diese Struktur bildet die Grundlage für die darauffolgenden Analyse- und Visualisierungsfunktionen. Das Modellierungsziel ist erreicht, wenn die Entitäten und Beziehungen gemäß 3.1 Datenmodell implementiert und mit Testdaten befüllt sind. Das Ziel soll bis zum Ende von Sprint 2 abgeschlossen sein.

### Suchfunktion

Über das Webfrontend soll es Nutzenden möglich sein die Datenbank nach gewünschten Entitäten zu durchsuchen. Die Suchfunktion enthält eine Freitextsuche mit Vorschlagfunktion. Gefundene Elemente können ausgewählt und anschließend visualisiert werden. Das Ziel soll am Ende von Sprint 3 abgeschlossen sein.

### Visualisierung verknüpfter Entitäten

Entitäten und ihre Beziehungen sollen in einer Graphenstruktur im Webfrontend dargestellt werden. Die Visualisierung dient der Übersicht und Veranschaulichung relevanter Zusammenhänge. Als Mindestfunktionalität müssen Entitäten und deren Verknüpfungen darstellbar und verschiebbar sein. Als zusätzliche Funktionalität ist Erweiterbarkeit und Reduzierbarkeit der Knoten geplant. Die Implementierung soll bis zum Ende von Sprint 3 abgeschlossen sein.

### Berichtserstellung

Die Plattform soll eine Berichtsgenerierung auf Basis der durchgeführten Analysen ermöglichen. Dieser Bericht basiert auf einem Ausschnitt der Visualisierung und Informationen zu den dargestellten Entitäten. Jeder Bericht enthält einen Titel, den Analysezeitpunkt sowie die benutzerdefinierte Auswahl der relevanten Analyseobjekte. Die Berichtsfunktion erlaubt eine gezielte Archivierung und Weiterverwendung der Analyseergebnisse. Ein Bericht wird zum Download bereit gestellt. Das Ziel gilt als erreicht, sobald mindestens ein vollständiger Beispielbericht erzeugt werden kann. Die Implementierung soll bis zum Ende von Sprint 3 abgeschlossen sein.

## 1.3 Risikomanagement

Die identifizierten Projektrisiken werden in Tabelle 1.1 dargestellt. Für jedes Risiko werden Beschreibung, Eintrittswahrscheinlichkeit, potenzieller Schaden, Priorität und Frühwarnindikatoren sowie geplante Maßnahmen zur Risikosteuerung angegeben. Die Priorität ergibt sich aus Eintrittswahrscheinlichkeit und Schaden. Risiken mit hoher Schadenshöhe werden auch bei niedriger Eintrittswahrscheinlichkeit kontinuierlich überwacht.

Risiko	Beschreibung	EW	Schaden	Priorität	Frühwarnindikatoren	Maßnahmen
Technologische Unsicherheit durch die Komplexität von Cypher-Queries	Für komplexe Abfragen von Knoten und Beziehungen müssen Cypher-Queries erstellt werden. Diese sind bei langen Traversierungen verschachtelt und kompliziert. Werden sie nicht richtig erstellt, sind solche Abfragen falsch oder nur eingeschränkt möglich.	mittel	hoch	kritisch	Cypher-Queries erfordern mehr als 5 Verschachtelungsebenen; hoher Anteil ungeplanter Research-Aufgaben	gezielte Lern- und Testphasen; schrittweise Erweiterung der Funktionalität
Nachträgliche Architekturänderungen	Erkenntnisse während der Entwicklung, sowie Feedbackschleifen können Änderungen an zentralen Architekturentscheidungen erforderlich machen und Refactoring-Aufwand verursachen.	mittel	hoch	kritisch	Refactoring über mehr als 3 Module gleichzeitig; unvorhergesehene Kopplung zwischen Systemkomponenten aufgrund von Architekturschwächen; Feedback erfordert 20 Stunden oder mehr Refactoring.	Einplanung von Refactorings als Backlog-Elemente; iterative Architekturentwicklung
Integrationsprobleme zwischen Backend und Frontend	Fehlende oder instabile Integration zwischen Frontend, Backend gefährdet die Funktionsfähigkeit der Plattform.	niedrig	hoch	hoch	mehrmalige Änderung der Schnittstellendefinition; verlängerte Nutzung von Mock-Komponenten	Frühzeitige Integrationstests; dokumentierte JSON-Schnittstelle; inkrementelle Integration der Komponenten
Unrealistische Aufwandsschätzung und Zeitplanung	Abweichungen zwischen geschätztem und tatsächlichem Aufwand können zu Verzögerungen im Projektverlauf führen.	mittel	mittel	moderat	Mehr als 2 nicht abgeschlossene Arbeitspakete pro Sprint; wiederholtes Verschieben von Backlog-Elementen; nicht erreichte Sprint-Ziele	Teilen von zu großen Paketen; regelmäßige Reviews; Anpassung von Story Points und Prioritäten

Tabelle 1.1: Übersicht der identifizierten Projektrisiken (EW = Eintrittswahrscheinlichkeit)

## 1.4 Zeitplan

Die Zeitplanung unterscheidet die drei Phasen. Die Konzeptionsphase ist mit 40 Stunden angesetzt. Für die Umsetzungsphase sind drei Sprints mit jeweils 40 (gesamt 120) Stunden geplant. Die Finalisierungsphase findet ebenfalls als Sprint mit 40 Stunden statt. Die Projektdauer wird insgesamt für 200 Stunden geplant.

Phase	Sprint	Stunden	Start ge-plant	Ende ge-plant	Start tatsächlich	Ende tatsächlich
Phase I	Planung	40	01.05.25	30.05.25	01.05.25	11.08.25
Phase II	Sprint 1	40	22.09.25	31.10.25	22.09.25	31.10.25
Phase II	Sprint 2	40	01.11.25	10.11.25	01.11.25	10.11.25
Phase II	Sprint 3	40	11.11.25	30.11.25	11.11.25	15.12.25
Phase III	Sprint 4	40	20.01.26	10.02.26		

Tabelle 1.2: Zeitlicher Projektplan

### Erläuterung zu den Sprint-Zeiträumen aus der Retrospektive

Die Entwicklung erfolgte im Rahmen eines Teilzeitstudiums (20 Stunden pro Woche) parallel zu beruflichen und familiären Verpflichtungen. Die große Lücke zwischen Phase I und II von 11.08.25 bis 22.09.25 entstand durch einen internationalen Umzug, der in dieser Zeit durchgeführt wurde. Die unregelmäßigen Sprint- und Arbeitspaket-Zeiträume resultieren aus der variablen Verfügbarkeit von Entwicklungszeit. Jeder Sprint umfasste einen Arbeitsaufwand von circa 40 Stunden, die je nach verfügbaren Zeitfenstern auf unterschiedlich lange Kalenderzeiträume verteilt wurden. Beispielsweise ist der zweite Sprint nur 10 Tage lang, da ich Urlaub in KW 45 (03.11.25 - 07.11.25) genommen hatte um den kritischsten Teil der Entwicklungsarbeit am Stück durchzuführen. Die Zeit- und Projektplanung wird kritisch in der Reflexion (Phase III) behandelt.

## 1.5 Meilensteine

Die folgenden Meilensteine markieren die wesentlichen Projektphasen und Entscheidungspunkte während der Entwicklung:

ID	Beschreibung	Zeitpunkt	Bedeutung	Status
M1	Technische Machbarkeit bewiesen	Ende Sprint 1 (voraussichtlich)	Neo4j-Integration funktionsfähig, Data-Mapper implementiert, technisches Risiko minimiert, Architektur bestätigt	Erreicht am 31.10.25

ID	Beschreibung	Zeitpunkt	Bedeutung	Status
M2	End-to-End-Datenpipeline etabliert	Mitte Sprint 2 (voraussichtlich)	Vollständige Datentransformation Neo4j zu Cytoscape funktioniert, kritische technische Hürde überwunden, Basis für alle weiteren Features geschaffen	Erreicht am 07.11.25
M3	MVP funktionsfähig	Ende Sprint 2 (voraussichtlich)	Kernanwendungsfälle demonstrierbar (Suche, Visualisierung, Bericht), minimale funktionale Anforderungen erfüllt, präsentationsfähiger Zustand erreicht	Erreicht am 13.11.25
M4	Projektziel erreicht	Ende Sprint 3 (voraussichtlich)	Alle funktionalen Anforderungen implementiert, System dokumentiert, bereit für finale Testphase, abgabefähiger Zustand	Erreicht am 15.12.25
M5	Release 1.0 / Finale Abgabe	Ende Sprint 4 (voraussichtlich Ende Jan. 2026)	Alle Tests abgeschlossen, Dokumentation vollständig, System produktionsreif und abgabefertig	Offen

## 1.6 Projektstrukturplan

Der folgende Projektstrukturplan zeigt die oberste Gliederungsebene des Projekts mit den drei Hauptphasen: Planung, Entwicklung sowie Testing und Finalisierung. Die Arbeitspaket-Gruppen sind hierarchisch nach psp-codes (Project-Management-Institute, 2019, Seite 31) strukturiert.

Tabelle 1.4: Projektstrukturplan - gesamtes Projekt - tabellarisch

PSP-Code	Arbeitspaket-Gruppe	Beschreibung	Start	Ende
<b>1</b>	<b>CTI-Plattform</b>	<b>Gesamtprojekt zur Entwicklung einer Threat Intelligence Plattform</b>	01.05.25	31.01.26
<b>1.1</b>	<b>Planung</b>	<b>Konzeption und Spezifikation des Projekts</b>	01.05.25	11.08.25

<b>PSP-Code</b>	<b>Arbeitspaket-Gruppe</b>	<b>Beschreibung</b>	<b>Start</b>	<b>Ende</b>
1.1.1	Anforderungsdokument	Stakeholderanalyse, Anforderungen, User Stories	01.05.25	01.06.25
1.1.2	Spezifikationsdokument	Technische Spezifikation, Datenmodell	01.06.25	01.07.25
1.1.3	Architekturdokument	Detaillierte Architekturkonzepte, Technologie-Stack	01.09.25	31.11.25
1.1.4	Benutzeranleitung	Installation und Ausführung der Anwendung	18.01.25	20.01.25
<b>1.2</b>	<b>Entwicklung</b>	<b>Implementierung der Plattform</b>	22.09.25	15.12.25
1.2.1	Architektur Grundstruktur	Projektsetup, Tools, Repository, Grundstruktur	22.09.25	31.10.25
1.2.2	Datenbank Integration	Neo4j Setup, Treiber, Beispieldaten	22.09.25	31.10.25
1.2.3	Rudimentäres Frontend	Svelte-Projekt, Layout, API-Anbindung, Visualisierung	22.09.25	31.10.25
1.2.4	Backend-Kommunikation	API-Contracts, Routen, Error-Handling	01.10.25	10.11.25
1.2.5	Request-Verarbeitung	Service-Layer, Queries	01.10.25	10.11.25
1.2.6	Rudimentärer Bericht	HTML-Generierung, Download	01.10.25	10.11.25
1.2.7	Suchfunktion	Autocomplete, Fuzzy Matching, Filter, Tests	11.11.25	30.12.25
1.2.8	Visualisierung	Kontexttiefe, Knotenerweiterung, Zeitstrahl, UX-Polish	11.11.25	30.12.25
1.2.9	Bericht erweitern	Strukturierung, Templates	11.11.25	30.12.25
<b>1.3</b>	<b>Testing &amp; Finalisierung</b>	<b>Qualitätssicherung und Projektabschluss</b>	20.01.26	10.02.26
1.3.1	Testdokument	Teststrategie, Testprotokoll	20.01.26	27.01.26
1.3.2	Abstract	Making-of, Reflexion	28.01.26	30.01.26
1.3.3	Finalisieren	Qualitätssicherung und Puffer	01.02.26	10.02.26

### Projektstrukturplan: Entwicklung (1.2)

Der folgende detaillierte Projektstrukturplan ist die Planungsannahme für alle fachlichen Arbeitspakete der Anwendung.

Tabelle 1.5: Projektstrukturplan - Entwicklung 1.2 - tabellarisch

PSP-Code	Arbeitspaket	Beschreibung	Start	Ende
<b>1.2</b>	<b>Entwicklung</b>	<b>Implementierung der CTI-Plattform</b>	22.09.25	30.11.25
<b>1.2.1</b>	<b>Architektur Grundstruktur</b>	<b>Projektsetup und Grundlagen</b>	22.09.25	24.09.25
1.2.1.1	Git-Hub Repository anlegen	Private Github Repository für das Projekt erstellt, README, Lizenz hinzugefügt	22.09.25	22.09.25
1.2.1.2	Tools vorbereiten	NeoVim, Linters, Dev-Tools eingerichtet	23.09.25	23.09.25
1.2.1.3	Ordnerstruktur anlegen	Verzeichnisstruktur Backend, Frontend erstellt	24.09.25	24.09.25
1.2.1.4	Projektumgebung einrichten	Python venv und Svelte vite ist eingerichtet	24.09.25	24.09.25
<b>1.2.2</b>	<b>Datenbank Integration</b>	<b>Neo4j Setup und Datenmodell</b>	25.09.25	12.10.25
1.2.2.1	Neo4j Setup lokal	Datenbank läuft lokal, zugreifbar	25.09.25	28.09.25
1.2.2.2	Treiber (Python) ins Backend integrieren	Treiber kann Verbindung aufbauen, Operationen ausführen	29.09.25	04.10.25
1.2.2.3	Datenmodell integrieren, Migrationsskripte schreiben	Datenmodell in die DB integriert	05.10.25	07.10.25
1.2.2.4	Beispiel-Daten erstellen und einpflegen	Datenbank enthält Beispiel-Daten	08.10.25	10.10.25
1.2.2.5	Integrationstests Backend zu Datenbank	Einfache CRUD-Tests laufen gegen Neo4j	11.10.25	12.10.25
<b>1.2.3</b>	<b>Rudimentäres Frontend</b>	<b>Frontend-Grundstruktur</b>	18.10.25	27.10.25
1.2.3.1	Svelte-Projekt initialisieren	Frontend-Projekt erzeugt, startet erfolgreich	18.10.25	18.10.25
1.2.3.2	Grund-Layout aus Mockup übernehmen	Layout sichtbar, grobe Komponenten vorhanden	19.10.25	19.10.25
1.2.3.3	API-Schnittstelle technisch vorbereiten	API-Calls vom Frontend möglich	20.03.25	22.10.25
1.2.3.4	Visualisierung Platzhalter (statischer Graph)	Graph sichtbar, klickbar	25.03.25	27.10.25

PSP-Code	Arbeitspaket	Beschreibung	Start	Ende
<b>1.2.4</b>	<b>Backend-Kommunikation</b>	<b>API-Schnittstellen und Request-Handling</b>	01.11.25	03.11.25
1.2.4.1	API-Contracts festlegen	Endpoints, Format, Error, Mögliche Requests festgelegt	01.11.25	01.11.25
1.2.4.2	Routen für Entität-GET implementieren	Endpoints erreichbar, liefern Daten zurück	02.11.25	02.11.25
1.2.4.3	Logging, Error-Handling Grundstruktur	Fehler werden korrekt behandelt und geloggt	03.11.25	03.11.25
<b>1.2.5</b>	<b>Request-Verarbeitung</b>	<b>Backend-Logik und Datenbankzugriff</b>	04.11.25	08.11.25
1.2.5.1	Querys für Requests erstellen	Cypher Queries Grundfunktionen vorhanden	04.11.25	05.11.25
1.2.5.2	Service-Layer für Querys implementieren	Services greifen die richtigen Querys zu	04.11.25	06.11.25
1.2.5.3	API-Antworten implementieren	Requests werden mit erwartetem Ergebnis beantwortet	04.11.25	06.11.25
1.2.5.4	Integrationstests für End-to-End Anfragen	Tests prüfen gesamte Kette (Request-Response)	07.11.25	07.11.25
<b>1.2.6</b>	<b>Rudimentärer Bericht</b>	<b>Berichtsgenerierung und Download</b>	08.11.25	09.11.25
1.2.6.1	Daten aus Frontend entgegennehmen	Selektion von Daten möglich	08.11.25	09.11.25
1.2.6.2	Bericht erzeugen	Standard Bericht wird als HTML aus den aktuellen Daten im Frontend erzeugt	08.11.25	09.11.25
1.2.6.3	Download bereitstellen	Bericht als Download verfügbar	08.11.25	09.11.25
<b>1.2.7</b>	<b>Suchfunktion</b>	<b>Erweiterte Suchfunktionalität</b>	11.11.25	15.11.25
1.2.7.1	Autocomplete Grundfunktion implementieren	Treffer-Vorschläge bei Eingabe sichtbar	11.11.25	12.11.25
1.2.7.2	Fuzzy Matching für Entitätssuche	Tippfehler werden toleriert	12.11.25	13.11.25
1.2.7.3	Filteroptionen ergänzen (Typ, Zeit, Kontext)	Filterbar nach Typ / Zeitraum etc.	14.11.25	15.11.25

<b>PSP-Code</b>	<b>Arbeitspaket</b>	<b>Beschreibung</b>	<b>Start</b>	<b>Ende</b>
<b>1.2.8</b>	<b>Visualisierung</b>	<b>Erweiterte Visualisierungsfunktionen</b>	17.11.25	23.11.25
1.2.8.1	Kontexttiefe steuerbar machen (Hops)	Nutzer kann Graph-Detailtiefe steuern	17.11.25	20.11.25
1.2.8.2	Reduzieren / Erweitern von Knoten umsetzen	Knoten sind reduzier- / erweiterbar	21.11.25	21.11.25
1.2.8.3	Zeitstrahl-Analyse hinzufügen	Daten im Zeitverlauf visualisierbar	22.11.25	22.11.25
1.2.8.4	UX-Polish (Transitions, Animationen, Labels)	Verbesserte Benutzerführung / Optik	23.11.25	23.11.25
<b>1.2.9</b>	<b>Bericht erweitern</b>	<b>Erweiterte Berichtsfunktionen</b>	24.11.25	29.11.25
1.2.9.1	Bericht strukturieren	Bericht mit einheitlichen strukturierten Elementen erzeugt	24.11.25	24.11.25
1.2.9.2	Templates für unterschiedliche Berichte bereitstellen	Mehrere Berichtsarten auswählbar	24.11.25	27.11.25
1.2.9.3	Benutzerdefinierter Bericht	Elemente sind einzeln auswählbar, Textfeld für individuelle Eingaben vorhanden	28.11.25	29.11.25

## 1.7 Entwicklung

### Github Repository

Das Repository für dieses Projekt unter [https://github.com/InsanityJoJo/Projekt\\_Software\\_Engineering\\_IU.git](https://github.com/InsanityJoJo/Projekt_Software_Engineering_IU.git) erreichbar.

### Übersicht

Der Fortschritt des Projekt ist durch ein Gantt-Diagramm dargestellt (1.1). Der Fortschritt der Sprints wird durch Burn-Down-Charts gemessen. Die drei Entwicklungssprints werden dokumentiert mit den Artefakten: *Sprint Planung*, *Sprint Backlog*, *Sprint Review* und *Sprint Retrospektive*. Die initiale Planung sah den Einsatz eines Kanban-Boards zur Priorisierung vor. In der Praxis erwies sich das Sprint Backlog jedoch als ausreichend strukturiert. Diese Entscheidung folgt dem agilen Prinzip "Responding to change over following a plan" Beck et al., 2001.

### Gantt-Diagramm

Das Gantt-Diagramm visualisiert den zeitlichen Ablauf der technischen Arbeitspakete während der Entwicklungsphase (1.2). Die Darstellung erfolgt auf Wochenbasis (Kalenderwochen). Die genauen Abschluss-Daten sind in den Backlogs dokumentiert.

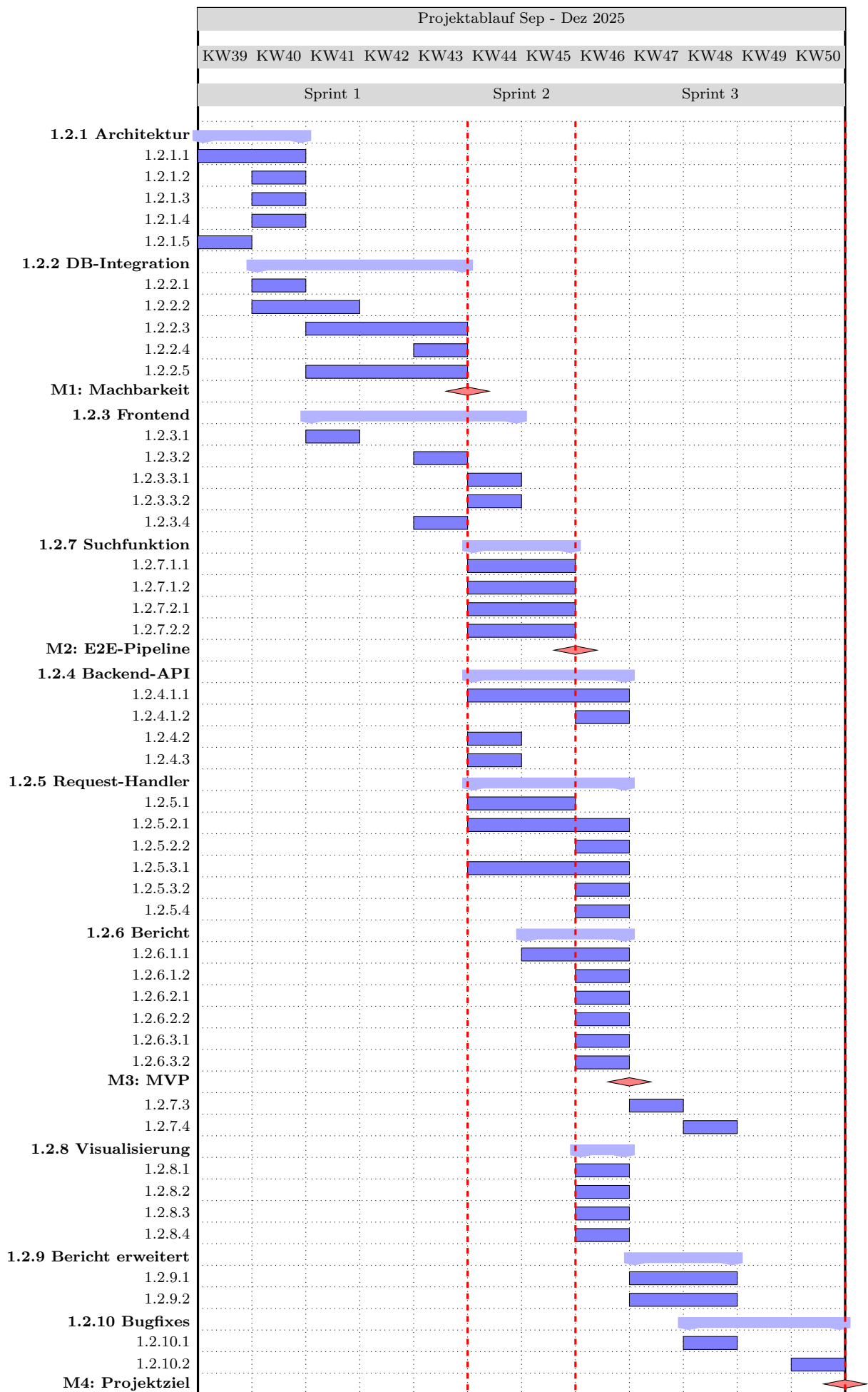


Abbildung 1.1: Gantt Chart der Entwicklung. Erstellt mit Python.

## Sprint 1: Grundstruktur und Datenbankintegration

Zeitraum: 22.09.25 - 31.10.25

### Sprint Planung

Ziel dieses Sprints ist die Vorbereitung der Entwicklung, die Grundstruktur der Systeme zu schaffen und die Anbindung der Datenbank an das Backend. In das Sprint-Backlog werden alle Arbeitspakete der Teilaufgaben 1.2.1, 1.2.2 und 1.2.3 aufgenommen, da diese die Basis des Projekts darstellen und jede weitere Entwicklung auf diesen Paketen aufbaut. Die Pakete sind mit Story Points, nach ihrem jeweiligen relativen Aufwand bewertet. Ziel ist ebenfalls Erfüllung des Meilensteins *M1: Technische Machbarkeit bewiesen*. Die Zeit dieses Sprints ist auf 39 Tage angesetzt.

### Sprint-Backlog 1

Tabelle 1.6: Backlog Sprint 1

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
<b>1.2.1</b>	<b>Architektur</b>	<b>Projektsetup abgeschlossen</b>	<b>5</b>	<b>Done</b>	22.09.25	28.09.25
1.2.1.1	Repository anlegen	Github-Repo erstellt, konfiguriert, README, Lizenz hinzugefügt.	1	Done	22.09.25	22.09.25
1.2.1.2	Tools vorbereiten	NeoVim, Linters, Dev-Tools eingerichtet	1	Done	28.09.25	28.09.25
1.2.1.3	Ordnerstruktur anlegen	Verzeichnisstruktur für Backend und Frontend erstellt	1	Done	28.09.25	28.09.25
1.2.1.4	Projektumgebung einrichten	Python venv und Svelte vite eingerichtet	2	Done	28.09.25	28.09.25
<b>1.2.2</b>	<b>DB-Integration</b>	<b>Datenbank einsatzbereit</b>	<b>16</b>	<b>Done</b>	03.10.25	29.10.25
1.2.2.1	Neo4j Setup	Datenbank läuft lokal, zugreifbar	2	Done	03.10.25	03.10.25
1.2.2.2	TDD	TDD Setup mit Pytest steht	2	Done	06.10.25	12.10.25
1.2.2.3	Treiber integrieren	Treiber kann Verbindung aufbauen, Operationen ausführen	3	Done	03.10.25	12.10.25

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
1.2.2.4	Docker Setup	Docker Setup für die Entwicklung implementiert	2	Done	12.10.25	18.10.25
1.2.2.5	Import Implementieren	Write Operationen über CLI implementiert	3	Done	12.10.25	29.10.25
1.2.2.6	Beispieldaten	Scripte für Datenimport und Datengenerierung füllen die DB mit Beispieldaten	2	Done	19.10.25	29.10.25
1.2.2.7	Integrationstests Backend-DB	Einfache CRUD-Tests laufen gegen Neo4j	2	Done	29.10.25	29.10.25
<b>1.2.3</b>	<b>Frontend</b>	<b>Frontend initialisiert</b>	<b>8</b>	<b>Done</b>	18.10.25	02.11.25
1.2.3.1	Svelte-Projekt initialisieren	Frontend-Projekt erzeugt, startet erfolgreich	2	Done	18.10.25	19.10.25
1.2.3.2	Grund-Layout übernehmen	Layout sichtbar, grobe Komponenten vorhanden	2	Done	29.10.25	29.10.25
1.2.3.4	Visualisierung Platzhalter	Dummy-Graph sichtbar, klickbar	1	Done	29.10.25	29.10.25
1.2.3.3	Frontend-API-Client	Frontend kann API ansprechen	3	-	01.11.25 (Sprint 2)	02.11.25 (Sprint 2)
<b>Gesamt Sprint 1:</b>			<b>26</b>			

## Sprint 1 - Burn-Down Chart

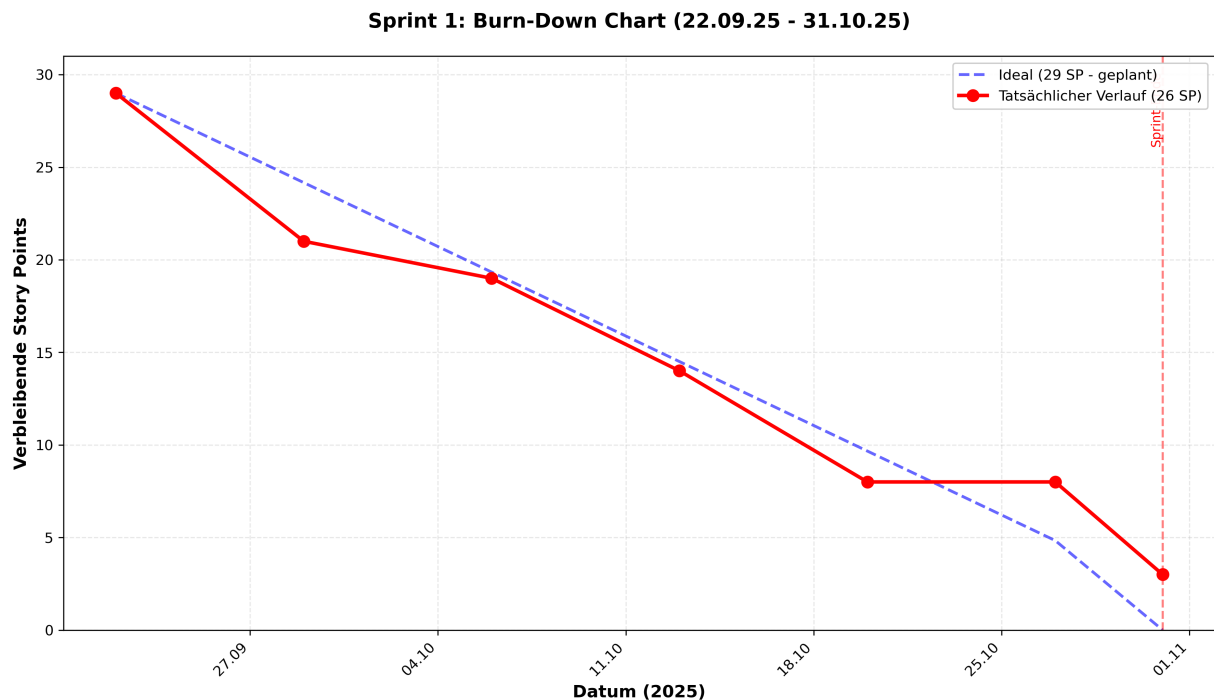


Abbildung 1.2: Burn-Down Chart Sprint 1 - Grundstruktur und Datenbankintegration (Python)

Der Chart Abbildung 1.2 zeigt den Fortschritt von Sprint 1 über 39 Tage. Die blaue gestrichelte Linie repräsentiert den ursprünglich geplanten Scope von 29 Story Points. Die rote Linie repräsentiert den tatsächlichen Verlauf. Dieser ist zu Beginn der Phase noch steil und passt sich in der Mitte des Sprints an die Ideallinie an. Gegen Ende des Sprints flacht die Linie ab. Es wurden nicht alle Pakete in diese Sprint abgeschlossen. 3 Story Points wurden in Sprint 2 verschoben.

### Sprint Review

In diesem Sprint wurden 26 der geplanten 29 Story Points erreicht. Implementiert wurden Backend, Frontend und Datenbank. Die Server werden über ein Docker-Setup verwaltet. Das Backend wurde erfolgreich an die Datenbank angeschlossen und diese wurde mit Beispieldaten befüllt. Ein rudimentäres Frontend mit Platzhaltern wurde implementiert. Die Frontend API (1.2.3.3) wurde begonnen konnte jedoch aufgrund Zeitmangels nicht fertiggestellt werden. Das Paket wurde in Sprint 2 verschoben. In diesem Sprint wurde der Meilenstein *M1: Machbarkeit bewiesen* erreicht. Nächstes Ziel für Sprint 2: Implementierung der Ende-zu-Ende-Kette und der Basisfunktionen.

### Sprint Retrospektive

#### Was lief gut?

Die Zeit in diesem Sprint war gut bemessen und ich konnte weitgehend konstant an der Anwendung arbeiten.

## Was lief schlecht?

Die Fachlichen Pakete erwiesen sich zu unstrukturiert für die technische Entwicklung. Aufgrund einer unerwarteten Verpflichtung in den letzten Oktoberwochen, konnte ich die Aufgaben nicht wie geplant durchführen daher wurde das Arbeitspaket 1.2.3.3 nicht vollständig bearbeitet.

## Action Items

- Arbeitspakete werden in den nächsten Sprints verfeinert und besser an technische Entwicklung angepasst.

## Sprint 2: Backend-Kommunikation und Requests

*Zeitraum: 01.11.25 - 10.11.25*

### Sprint Planung

In diesem Sprint liegt der Fokus auf der Verarbeitung von Requests und das Erreichen des Meilensteins *M2: E2E-Pipeline*. Die fachlichen Arbeitspakete sind angepasst, teilweise aufgespalten und bewertet. Die Pakete 1.2.4.1, 1.2.5.2, 1.2.5.3, 1.2.7.1 und 1.2.7.3 wurden in technisch sinnvolle Sprint-Elemente geteilt. Die Ende-zu-Ende-Kette wird anhand der Suchfunktion erreicht. Die Suchfunktion, FA-01 (Kapitel 2.2) muss Autovervollständigung leisten um dem Nutzenden Ergebnisse vorzuschlagen, dargestellt in 3.5. Dies ist einen essenzieller Bestandteil der Plattform und wurde daher mit erhöhter Priorität vorgezogen. Dieser Sprint ist auf 10 Tage angesetzt und wird als Block abgearbeitet.

### Sprint-Backlog 2

Tabelle 1.7: Backlog Sprint 2

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
<b>1.2.3</b>	<b>Frontend (Fortsetzung)</b>	<b>API-Anbindung fertig</b>	<b>3</b>	<b>Done</b>	03.11.25	04.11.25
1.2.3.3	Backend-API-Anbindung	API-Integration vollständig	3	Done	03.11.25	04.11.25
<b>1.2.4</b>	<b>Backend-Kommunikation</b>	<b>API-Grundstruktur implementiert</b>	<b>5</b>	<b>Done</b>	04.11.25	13.11.25
1.2.4.1.1	API-Contracts festlegen <sup>1</sup>	Endpoints, Format, Error, Requests festgelegt.	1	Done	05.11.25	05.11.25
1.2.4.2	Routen für GET implementieren	Endpoints erreichbar, liefern Daten zurück	2	Done	04.11.25	04.11.25

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
1.2.4.3	Error-Handling	Fehler werden korrekt behandelt	2	Done	04.11.25	04.11.25
<b>1.2.5</b>	<b>Request-Handler</b>	<b>Backend-Logik implementiert</b>	<b>11</b>	<b>Done</b>	04.11.25	13.11.25
1.2.5.1	Querys für Requests erstellen	Cypher Queries Grundfunktionen vorhanden	4	Done	04.11.25	07.11.25
1.2.5.2.1	Service-Layer für Querys implementieren <sup>2</sup>	Services greifen die richtigen Querys zu	3	Done	05.11.25	10.11.25
1.2.5.3.1	handlers <sup>3</sup>	handler greifen auf Services zu	4	Done	05.11.25	10.11.25
<b>1.2.6</b>	<b>Berichtfunktion</b>	<b>In Sprint 3 verschoben</b>	<b>-</b>	<b>Verscho-ben</b>		-
Arbeitspaket 1.2.6 wurde nicht bearbeitet vollständig in Sprint 3 verschoben						
<b>1.2.7</b>	<b>Suchfunktion</b>	<b>Basis-Suchfunktionen</b>	<b>11</b>	<b>Done</b>	01.11.25	14.11.25
1.2.7.1.1	Autocomplete Backend-Logik	Backend liefert Autocomplete-Vorschläge	4	Done	01.11.25	07.11.25
1.2.7.1.2	Autocomplete Frontend	User wird eine Auswahl präsentiert	2	Done	02.11.25	07.11.25
1.2.7.2	Fuzzy Matching	Fuzzy Matching funktioniert	3	Done	02.11.25	07.11.25
1.2.7.3.1	Label-Filter <sup>4</sup>	Label-Filter integriert	2	Done	02.11.25	10.11.25
<b>Gesamt Sprint 2:</b>			<b>30</b>			

<sup>1</sup>1.2.4.1.2 wird in Sprint 3 abgeschlossen

<sup>2</sup>1.2.5.2.2 wird in Sprint 3 abgeschlossen

<sup>3</sup>1.2.5.3.2 wird in Sprint 3 abgeschlossen

<sup>4</sup>1.2.7.3.2 wird in Sprint 3 abgeschlossen

## Sprint 2 - Burn-Down Chart

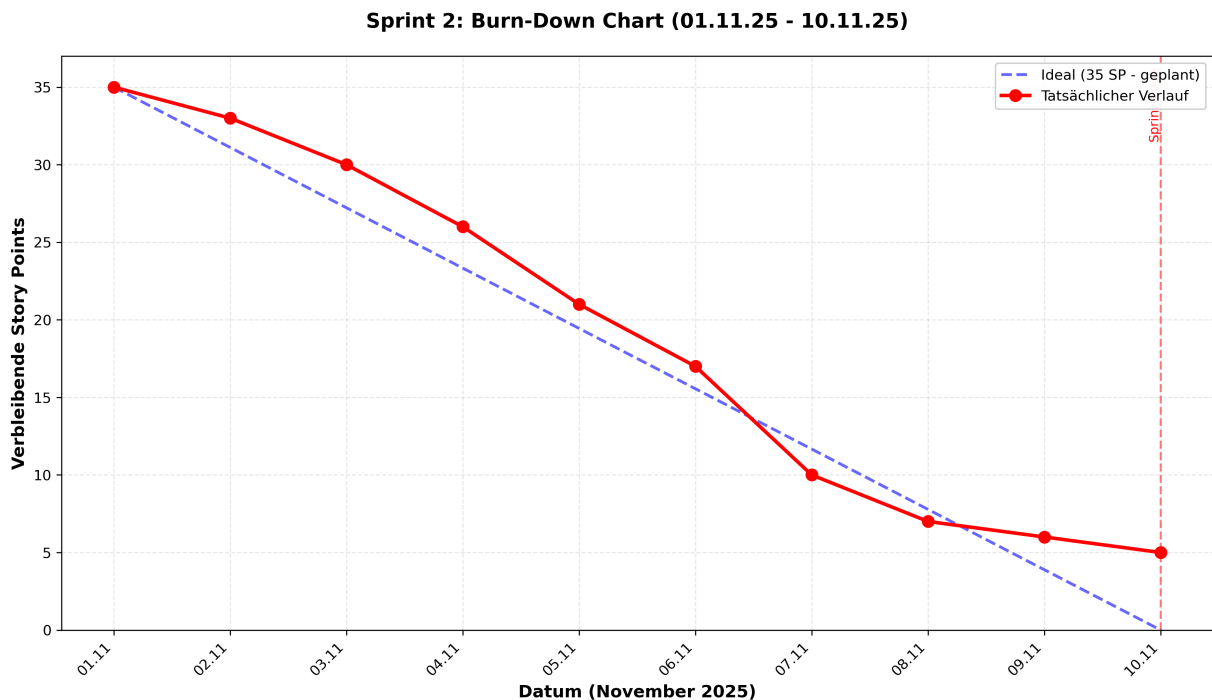


Abbildung 1.3: Burn-Down Chart Sprint 2- Backend-Kommunikation und Requests (Python)

Sprint 2 umfasste ursprünglich 35 Story Points über 10 Arbeitstage. Der tatsächliche Verlauf (rote Linie) zeigt initial guten Fortschritt in den ersten Tagen. Gegen Ende verflachte der Fortschritt. 5 Story Points wurden in Sprint 3 verschoben.

### Sprint Review

In diesem Sprint wurden 30 der geplanten 35 Story Points erreicht. Neue Funktionen sind die Suchfunktion (FA-01) mit Autocomplete (FA-01.1) und Fuzzy Search (FA-01.2) sowie die Labelfilter (Teil von FA-01.3). Die Visualisierung einzelner Nodes (Teil von FA-02) ist ebenfalls erfolgreich.

Das Arbeitspaket 1.2.6 (Berichtsgenerierung, 5 SP) wurde aus Zeitmangel vollständig in den Sprint 3 verschoben. Trotz nicht eingehaltenem Sprint-Ziel wurde der Meilenstein *M2: End-to-End-Datenpipeline etabliert* am 07.11.25 erreicht.

Nächstes Ziel für Sprint 3: Berichtsgenerierung und Finalisierung der Anwendung.

### Sprint Retrospektive

#### Was lief gut?

Der Sprint wurde in einem Block durchgeführt. Die Arbeitspakete wurden auf die Entwicklung angepasst. Daher erfolgte Sprint 2 trotz Schwierigkeiten effizienter als in Sprint 1. Die Velocity betrug 3 SP pro Tag, in Sprint 1 waren es 0,67 SP pro Tag.

#### Was lief schlecht?

Die erhöhte Komplexität der Cypher-Queries (1.2.5.1, 1.2.7.1.1, 1.2.7.2) für die Implementierung der Grundfunktionen, Autocomplete und Fuzzy Matching führten zu langen Recherche

Zeiten. Dieser zeitliche Verzug sorgte dafür, dass das Sprint Ziel von 35 SP nicht eingehalten werden konnte. Damit ist das Risiko „Technologische Unsicherheit durch die Komplexität von Cypher-Queries“ eingetreten. Die Maßnahmen zur Minderung waren nicht adäquat. Gezielte Lernphasen wurden vor der Entwicklung investiert (Anlage 5.4, Seite 85), diese waren nicht ausreichend. Die schrittweise Erweiterung der Funktionalität wurde durchgeführt, diese war zwar zielführend, adressierte aber nicht das Risiko. Eine schrittweise Entwicklung wird grundsätzlich bei dieser Art der Implementierung angewendet (TDD).

Das Verschieben von 1.2.6, das nicht erreichte Sprint-Ziel und das wiederholte Verschieben von Backlog-Elementen in den nächsten Sprint erfüllt alle Frühwarnindikatoren für das Risiko *Unrealistische Aufwandseinschätzung und Zeitplanung*. Der Grund hierfür sind die Cypher Querys (siehe oben). Es werden die vorgesehenen Maßnahmen zur Minderung umgesetzt.

Eine weitere Herausforderung war das Einhalten und die Implementierung der festgelegten Architektur, die durch Refactoring Arbeiten am 05.11.25 umgesetzt wurde. Die Architektur wurde zu Beginn nicht streng eingehalten. Das daraus resultierende Refactoring kostete zusätzlich Zeit.

### **Action Items**

- Für komplexe Aufgaben wie das Erstellen von Cypher Queries werden Prototypen vor der eigentlichen Entwicklung erstellt.
- Elemente für Sprint 3 neu anpassen.
- Kurze Architektur-Reviews alle 3 Tage durchführen.

## **Sprint 3: Erweiterte Features und Finalisierung**

*Zeitraum: 11.11.25 - 15.12.25*

### **Sprint Planung**

Ziel dieses Sprints ist das Erreichen des Meilensteins *M3: MVP Funktionsfähig* und *M4: Projektziel erreicht*. Die Arbeitspakete wurden verfeinert. Die verbleibenden Arbeitspakete für die Backend-Entwicklung sind zusammengeführt und in drei einzelne Arbeitspakete aufgespalten: 1.2.4.1.2 Request-handling, 1.2.5.2.2 komplexe Queries und 1.2.5.3.2 Exception-handling. Die Gruppen 1.2.6 Bericht und 1.2.9 Bericht-Erweiterung sind zusammengeführt. Unter 1.2.10 ist eine Teilaufgabe für Validierung und Bugfixes hinzugefügt. Der Sprint ist für 34 Tage angesetzt. Die erste Woche ist als Block geplant.

### **Sprint-Backlog 3**

Tabelle 1.8: Backlog Sprint 3

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
<b>1.2.4 und 1.2.5</b>	<b>Request-Handler (Fortsetzung)</b>	<b>Finalisierung</b>	<b>9</b>	<b>Done</b>	05.11.25	13.11.25
1.2.4.1.2	Requests	handler für jeden Request implementiert	2	Done	05.11.25	13.11.25
1.2.5.2.2	Komplexe Queries	Erweiterte Cypher Queries funktionieren	5	Done	05.11.25	13.11.25
1.2.5.3.2	Handlers Stabilität	Exception Handling ist implementiert	2	Done	05.11.25	13.11.25
<b>1.2.6 und 1.2.9</b>	<b>Bericht</b>	<b>Basis und erweiterte Funktionen</b>	<b>12</b>	<b>Done</b>	12.11.25	13.11.25
1.2.6.1	Datenbereitung für Bericht	Generator erhält alle notwendigen Daten	1	Done	12.11.25	13.11.25
1.2.6.2	Basis-HTML-Generator	Einfacher HTML-Bericht wird generiert	3	Done	13.11.25	13.11.25
1.2.6.3	PrintToPDF	Bericht kann per Print to PDF heruntergeladen werden	1	Done	13.11.25	13.11.25
1.2.9.1	Bericht strukturieren	Strukturierter Bericht implementiert	2	Done	18.11.25	20.11.25
1.2.9.2	Templates bereitstellen	Mehrere Berichtsarten auswählbar	2	Done	18.11.25	20.11.25
1.2.9.3	Benutzerdefinierter Bericht	Einzelauswahl und Textfeld vorhanden	3	Done	06.12.25	11.12.25
<b>1.2.7</b>	<b>Suchfunktion (Fortsetzung)</b>	<b>Zeitfilter implementiert</b>	<b>3</b>	<b>Done</b>	14.11.25	17.11.25
1.2.7.3.2	Zeit-Filter ergänzen	Filterbar nach Zeitraum	3	Done	14.11.25	14.11.25
<b>1.2.8</b>	<b>Visualisierung</b>	<b>Erweiterte Visualisierung abgeschlossen</b>	<b>12</b>	<b>Done</b>	11.11.25	13.11.25
1.2.8.1	Kontexttiefe steuerbar	Nutzer kann Graph-Detailtiefe steuern	3	Done	12.11.25	13.11.25
1.2.8.2	Knoten erweitern/reduzieren	Knoten sind erweiter-/reduzierbar	2	Done	12.11.25	13.11.25

PSP-Code	Beschreibung	Akzeptanzkriterium	SP	Status	Start	Ende
1.2.8.3	Zeitstrahl-Analyse	Entitäten sind im Zeitverlauf visualisierbar	5	Done	11.11.25	14.11.25
1.2.8.4	UX-Polish	Dunkle Hintergründe, Helle Schriften implementiert, Box Anpassungen abgeschlossen	2	Done	11.11.25	17.11.25
<b>1.2.10</b>	<b>Bugfixes und Validation</b>	<b>Validierung implementiert und Bugs entfernt</b>	<b>4</b>	<b>Done</b>	17.11.25	14.12.25
1.2.10.1	Inputvalidation umsetzen	Input wird sanitized und validiert	2	Done	17.11.25	18.11.25
1.2.10.2	fix: Timeline-Analysis	Utf8 Encoding, Style Issues behoben	2	Done	14.12.25	14.12.25
<b>Gesamt Sprint 3:</b>			<b>40</b>			

### Sprint 3 - Burn-Down Chart

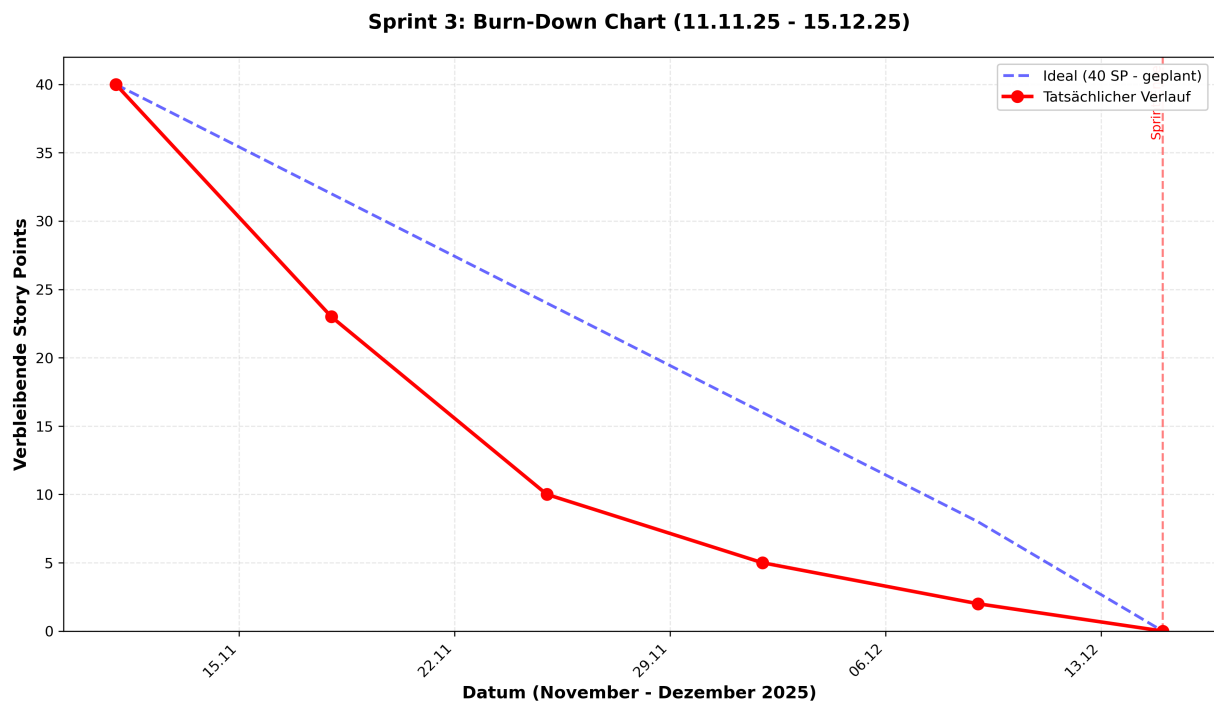


Abbildung 1.4: Burn-Down Chart Sprint 3 - Erweiterte Features und Finalisierung (Python)

Sprint 3 umfasste 40 Story Points über 34 Arbeitstage. Der tatsächliche Verlauf (rote Linie) zeigt sehr schnelle Entwicklung in den ersten 10 Tagen. Der so gewonnene Vorsprung ermöglichte

das Abschließen aller offene Arbeitspakete.

## **Sprint Review**

Im Sprint wurden alle 40 Story Points abgeschlossen. Die Meilensteine *M3: MVP Funktionsfähig* und Meilenstein *M4: Projektziel erreicht*, wurden am 13.11.25 und am 15.12.25 erreicht. Die Plattform erfüllt alle funktionalen Anforderungen und ist bereit für die finale Testphase.

## **Sprint Retrospektive**

### **Was lief gut?**

Die Vorarbeit aus Sprint 2, speziell im Bereich des *queryBuilder*, ermöglichte eine effiziente Implementierung der weiteren Requests im Backend. Der Einsatz von TDD sorgte für Stabilität im Code. Trotz erhöhtem initialem Entwicklungsaufwand traten über Module hinweg nur wenige Fehler auf.

Im Backend wurden 335 Unit-Tests mit einer Abdeckung von 80% umgesetzt, im Frontend 145 Unit-Tests für JavaScript-Module mit einer Abdeckung von 98 %.

Der frühe Abschluss eines Großteils der Story Points führte zu einem zeitlichen Puffer, der für Qualitätssicherung und Finalisierung genutzt werden konnte.

Die Velocity sank im Vergleich zu Sprint 2 von 3,0 auf 1,18 in Sprint 3, was auf das größere Zeitfenster und Qualitätssicherungsaufgaben zurückzuführen ist. Die Entwicklung war in diesem Sprint schnell aber nicht gehetzt.

### **Was lief schlecht?**

Der Einsatz von *Pylint* erfolgte erst gegen Ende des Sprints, wodurch Refactoring-Arbeiten in einer späten Phase anfielen. Diese waren durch den vorhandenen Zeitpuffer umsetzbar. Durch eine frühere oder parallele Nutzung von *Pylint* hätte das Zeitfenster anderweitig genutzt werden können. Refactoring zog häufig auch Anpassungen an bestehenden Mocks und Tests mit sich, was den Aufwand in dieser Phase erhöhte.

## **Action Items**

- Statische Codeanalyse (z. B. *Pylint*) frühzeitig und kontinuierlich in den Entwicklungsprozess integrieren.
- TDD beibehalten, Refactoring-Phasen jedoch explizit einplanen und als festen Bestandteil der Entwicklung betrachten.

## 2 Anforderungsdokument

### 2.1 Stakeholder

Die Stakeholder sind für dieses Projekt in Gruppen aufgeteilt. Diese Aufteilung erfolgt aufgrund großer Unterschiede des tatsächlichen Einflusses auf dieses Projekt. Es ist nicht geplant die Plattform als Produkt zur Verfügung zu stellen. Daher sind die Gruppen Nutzer, Regulierungs- und Kontrollinstanz sowie der Hosting-Anbieter nur theoretisch vorhanden. Die Anforderungen aus diesen Gruppen werden aus meiner eigenen Erfahrung erzeugt.

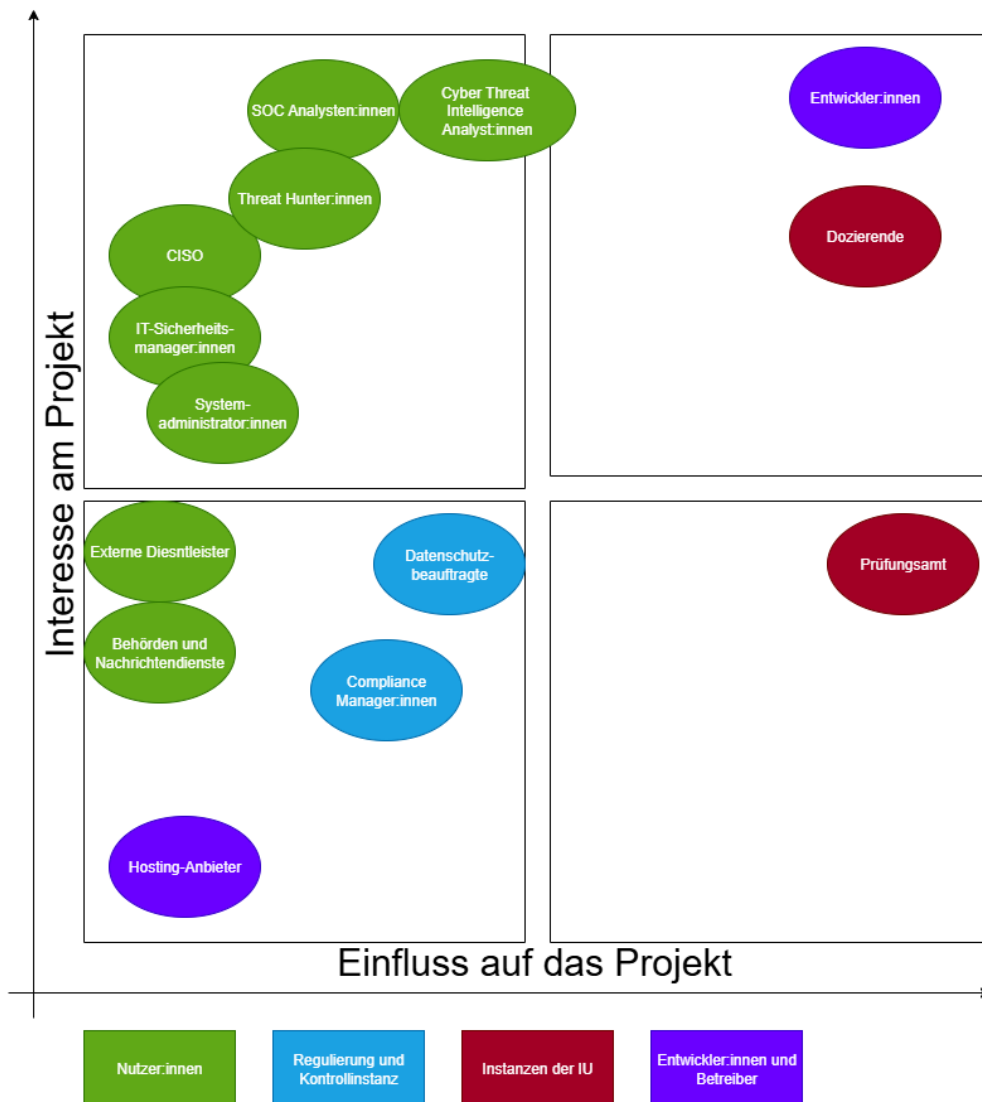


Abbildung 2.1: Stakeholdermatrix basierend auf Einfluss und Interesse am Projekt.

## **Gruppe Nutzer:innen**

Die Stakeholder dieser Gruppe sind potenzielle Kunden:innen, welche die Plattform während ihrer täglichen Arbeit nutzen.

- Cyber Threat Intelligence Analyst:innen sammeln und bewerten Informationen über Bedrohungen für die Systeme von Organisationen.
- SOC-Analyst:innen arbeiten in einem Security Operations Center. Sie überwachen und analysieren Sicherheitsereignisse in Echtzeit.
- CISOs sind für die gesamte Informationssicherheit eines Unternehmens verantwortlich und entwickeln strategische Sicherheitsrichtlinien.
- IT-Sicherheitsmanager:innen leiten Sicherheitsprojekte und -teams, implementieren Sicherheitsrichtlinien und -verfahren.
- Systemadministrator:innen verwalten und warten die IT-Systeme eines Unternehmens und stellen dessen Schutz sicher.
- Threat Hunter suchen proaktiv nach Cyber-Bedrohungen um potenzielle Angriffe zu identifizieren.
- Behörden und Nachrichtendienste behandeln Cyberkriminalität und Bedrohungen aus dem Cyberraum für staatliche Einrichtungen.
- Externe Dienstleister stellen Services für andere Unternehmen bereit. Sie haben ein erhöhtes Bedürfnis für Sicherheit, da ein Angriff die Kompromittierung sämtlicher Kunden bedeuten kann.

## **Gruppe Regulierungs- und Kontrollinstanzen**

Die Stakeholder dieser Gruppe haben eine Kontrollfunktion. Die Verwendung oder Anschaffung der Plattform betrifft ihren Verantwortungsbereich.

- Datenschutzbeauftragte tragen die Verantwortung für die rechtmäßige Datenverarbeitung, Datenspeicherung und Sicherheit der personenbezogenen Daten in einer Organisation.
- Compliance-Manager:innen implementieren Kontrollsysteme, um die Einhaltung von Rechtsvorschriften und organisationsinterner Vorgaben durchzusetzen.

## **Gruppe Entwickler:innen und technische Betreiber**

Diese Stakeholder sind an der Entwicklung und dem Betrieb der Plattform beteiligt.

- Entwickler:innen sind alle an der Entwicklung der Plattform beteiligten Personen. In diesem Fall ist das meine Person.
- Hosting-Anbieter sind Organisationen, welche unter Vertrag stehen die Plattform, den Nutzer:innen verfügbar zu machen. Diese Plattform wird für dieses Projekt lokal betrieben.

## **Gruppe Instanzen der IU**

Diese Gruppe hat die Besonderheit, dass sie eine Kontroll- und Bewertungsinstanz darstellt. Sie ist die Gruppe mit dem größten Einfluss auf das Projekt.

- Betreuende Dozierende, gibt Feedback und bewertet die Umsetzung.
- Prüfungsamt gibt den Rahmen des Projekts vor.

Als Grundlage der Stakeholderanalyse dient BMI, 2021. Eine ausführliche Darstellung der Stakeholder-Analyse befindet sich in Anhang 5.1. Diese wurde mit „Microsoft Excel“, 2025 erstellt.

## 2.2 Funktionale Anforderungen

Die Funktionalen Anforderungen sind in drei Gruppen gegliedert. Sie sind über ihre ID identifizierbar.(FA: Funktionale Anforderung, US: User Story)

---

### Gruppe 01 Suchfunktion

#### **FA-01:** Entitätensuche

**US-01:** Als Threat Intelligence-Analyst:in möchte ich gezielt nach Entitäten wie Threat Actor oder TTPs suchen können, um Bedrohungen und Zusammenhänge zu identifizieren.

**Beschreibung:** Das System bietet eine kontextuelle Suchfunktion, mit der Nutzer:innen Entitäten finden und deren Beziehungen im Threat Intelligence-Kontext analysieren können.

**Akzeptanzkriterien:** Die Suche liefert relevante Entitäten aus der Datenbank.

#### **FA-01.1:** Autovervollständigung

**US-01.1:** Als Threat Intelligence-Analyst:in möchte ich während der Eingabe von Suchbegriffen Vorschläge erhalten, um schneller und fehlerfreier Entitäten finden zu können.

**Beschreibung:** Während der Eingabe eines Suchbegriffs werden passende Entitäten vorgeschlagen. Dies verbessert die Benutzerfreundlichkeit und reduziert Tippfehler.

**Akzeptanzkriterien:** Bei der Eingabe von mindestens drei Zeichen werden relevante Entitätsnamen angezeigt, die zur Auswahl übernommen werden können.

#### **FA-01.2:** Fuzzy Matching

**US-01.2:** Als Threat Intelligence-Analyst:in möchte ich auch bei leichten Tippfehlern oder unterschiedlichen Schreibweisen passende Entitäten finden, damit ich die Analyse nicht durch ungenaue Begriffe verliere.

**Beschreibung:** Die Suche erkennt auch leicht fehlerhafte oder variierende Schreibweisen und gibt entsprechende Treffer aus.

**Akzeptanzkriterien:** Bei Eingabe eines unvollständigen oder leicht fehlerhaften Begriffs, beispielsweise APT28 statt APT-28, erscheinen trotzdem relevante Suchergebnisse.

### **FA-01.3:** Suchfilter

**US-01.3:** Als Threat Intelligence-Analyst:in möchte ich Entitäten anhand von Typen, beispielsweise Malware, TTPs oder Zeitfenstern filtern, um gezielter relevante Daten finden zu können.

**Beschreibung:** Nutzer:innen können die Suchergebnisse filtern, um gezieltere Analysen durchzuführen.

**Akzeptanzkriterien:** Suchabfragen können durch mindestens zwei Filter eingeschränkt werden.

### **FA-01.4:** Einstellbare Kontexttiefe

**US-01.4:** Als Threat Intelligence-Analyst:in möchte ich die Tiefe der angezeigten Beziehungen konfigurieren können z.B. 1 bis 3 Hops, um nur relevante Zusammenhänge zu sehen.

**Beschreibung:** Die Benutzeroberfläche erlaubt es, beim der Suche festzulegen, wie viele Beziehungsstufen zu anderen Entitäten angezeigt werden sollen.

**Akzeptanzkriterien:** Auswahlmöglichkeit vor der Suche, bei der Nutzer:innen die Beziehungstiefe zwischen 1 und 3 einstellen. Die Visualisierung zeigt gesuchte Entitäten in der gewünschten Tiefe.

---

## **Gruppe 02 Visualisierung**

### **FA-02:** Visualisierung

**US-02:** Als Threat Intelligence-Analyst:in möchte ich die Beziehungen zwischen Entitäten visuell als Graph dargestellt bekommen, um Muster, Cluster oder Ketten in Bedrohungsszenarien zu erkennen.

**Beschreibung:** Nutzer:innen können Entitäten und ihre Beziehungen in einer graphenbasierten Oberfläche visualisieren, um Zusammenhänge intuitiv zu erfassen.

**Akzeptanzkriterien:** Die Visualisierung zeigt relevante Knoten und Kanten.

### **FA-02.1:** Interaktive Visualisierung

**US-02.1:** Als Threat Intelligence-Analyst:in möchte ich Entitäten und deren Beziehungen in einer interaktiven Graphendarstellung bewegen, um komplexe Zusammenhänge besser zu verstehen.

**Beschreibung:** Nutzer:innen können Elemente auswählen und bewegen.

**Akzeptanzkriterien:** Nach einer Suche wird ein interaktiver Graph angezeigt. Knoten lassen sich verschieben (Drag & Drop).

### **FA-02.2:** Iteraktives Node-Handling

**US-02.2:** Als Threat Intelligence-Analyst:in möchte ich einzelne Knoten im Graph gezielt erweitern oder reduzieren können, um relevante Zusammenhänge dynamisch ein- oder auszublenen.

**Beschreibung:** Die Visualisierung erlaubt es, durch Klick auf einen Knoten dessen direkte Beziehungen dynamisch nachzuladen oder auszublenden. Dies hilft bei der fokussierten Analyse komplexer Netze.

**Akzeptanzkriterien:** Nutzer:innen können durch Interaktion z.B. Kontextmenü oder Doppelklick verbundene Knoten ein- oder ausblenden. Der Graph wird dabei neu skaliert.

### **FA-02.3:** Zeitstrahlvisualisierung

**US-02.3:** Als Threat Intelligence-Analyst:in möchte ich Entitäten mit Zeitbezug auch als Zeitstrahl visualisieren können, um chronologische Abfolgen nachvollziehen zu können.

**Beschreibung:** Für Entitäten mit Zeitstempeln z.B. Kampagnen oder Vorfälle kann eine alternative Darstellung als Zeitstrahl aktiviert werden.

**Akzeptanzkriterien:** Bei Auswahl geeigneter Entitäten kann ein Zeitstrahl generiert werden, auf dem die Positionen entsprechend ihres Zeitstempels dargestellt sind.

---

## **Gruppe 03 Bericht**

### **FA-03:** Berichtsgenerierung

**US-03:** Als Entscheidungsträger:in möchte ich zusammengefasste Berichte zu Bedrohungen oder Kampagnen erhalten, um fundierte Entscheidungen treffen und Sicherheitsmaßnahmen ableiten zu können.

**Beschreibung:** Das System ermöglicht die Erstellung standardisierter Berichte auf Basis ausgewählter Entitäten, die für interne Kommunikation oder Entscheidungsfindung genutzt werden können.

**Akzeptanzkriterien:** Der Bericht enthält Metadaten, Beziehungen und Kontextinfos. Export erfolgt als standardisiertes Format.

### **FA-03.1:** Auswahlfunktion

**US-03.1:** Als Threat Intelligence-Analyst:in möchte ich im Graph einzelne oder mehrere Entitäten zur Berichtsgenerierung auswählen können, um nur relevante Informationen zu exportieren.

**Beschreibung:** Die Benutzeroberfläche erlaubt es, per Mehrfachauswahl oder Kontextmenü Entitäten für den Bericht zu markieren.

**Akzeptanzkriterien:** Nutzer:innen können eine oder mehrere Entitäten auswählen.

### FA-03.2: Strukturierter Bericht

**US-03.2:** Als Threat Intelligence-Analyst:in möchte ich strukturierte Berichte auf Basis der ausgewählten Entitäten erstellen können, um Erkenntnisse weiterzugeben oder zu archivieren.

**Beschreibung:** Die Anwendung generiert Berichte mit automatisch formatierten Inhalten: Beschreibung der Entitäten, Metadaten, Verbindungen, und ggf. Diagramme.

**Akzeptanzkriterien:** Der Bericht enthält einen Titel, Inhaltsverzeichnis, Abschnitte zu den Entitäten und Beziehungen.

### FA-03.3: Berichtvorlagen

**US-03.3:** Als Threat Intelligence-Analyst:in möchte ich zwischen verschiedenen Vorlagen wählen, um spezifische Analyseziele zu adressieren.

**Beschreibung:** Das System bietet vorkonfigurierte Templates, beispielsweise für Management oder Analyse.

**Akzeptanzkriterien:** Mindestens drei Standardvorlagen stehen zur Auswahl.

## Story Map

Die Funktionalen Anforderungen sind hier nach Priorität geordnet dargestellt. Die Priorität ergibt sich aus den mindestens erforderlichen Funktionen.

Priorität	ID	Name
<b>Hoch</b>	FA-01	Entitätensuche
	FA-02	Visualisierung
	FA-03	Berichtsgenerierung
	FA-02.1	Interaktive Visualisierung
	FA-02.2	Interaktives Node-Handling
	FA-03.1	Auswahlfunktion
	FA-03.2	Strukturierter Bericht
<b>Mittel</b>	FA-01.1	Autovervollständigung
	FA-01.4	Einstellbare Kontexttiefe
	FA-02.3	Zeitstrahlvisualisierung
<b>Niedrig</b>	FA-01.2	Fuzzy Matching
	FA-01.3	Suchfilter
	FA-03.3	Berichtvorlagen

Tabelle 2.1: Story Map geordnet nach Priorität

## ULM-Use-Case-Diagramm

Das UML Use-Case-Diagramm (OMG, 2017) wurde mit draw.io (Ltd, 2025) erstellt.

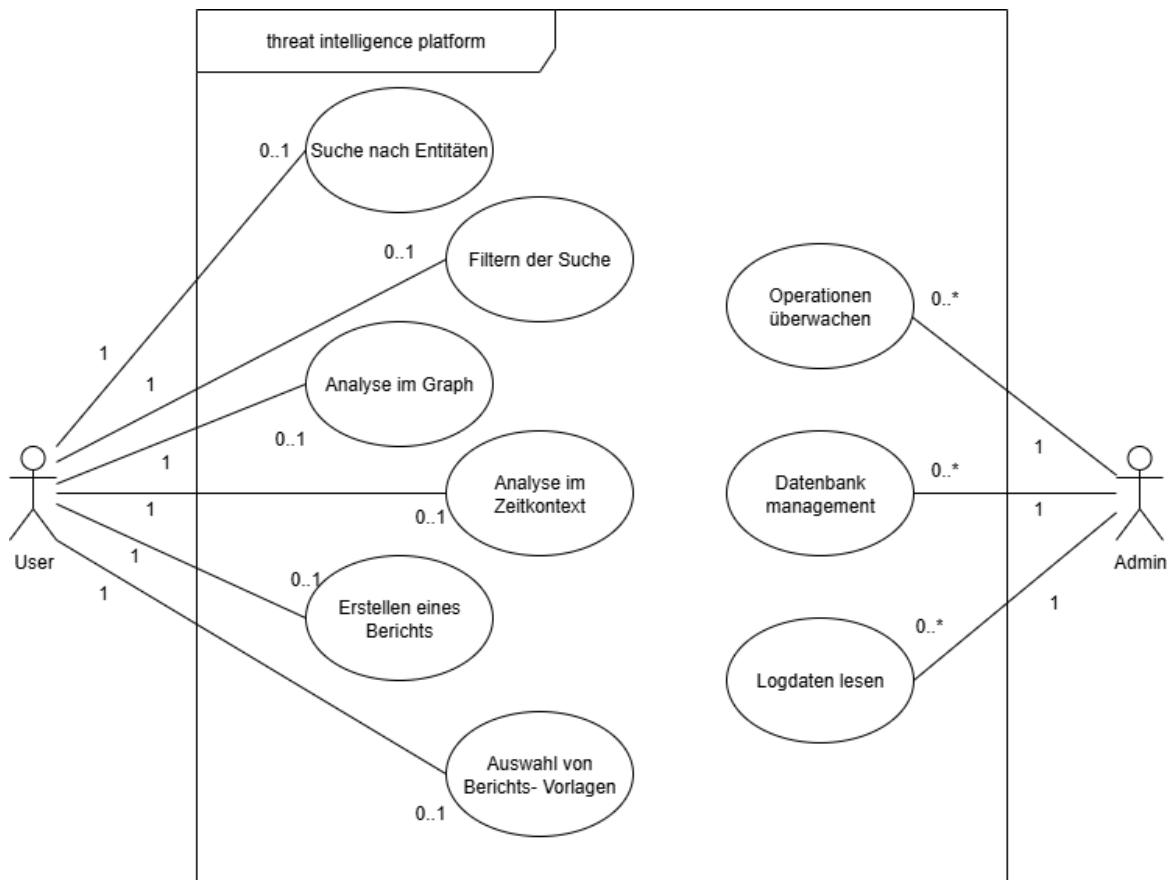


Abbildung 2.2: UML-Use-Case-Diagramm erstellt mit draw.io

## Analyse des Systemkontext

Im Systemkontext stehen die Stakeholder, hier als User dargestellt. Zudem ist der Administrator für die Sicherstellung der Funktion erkennbar. Die Interaktion mit den Usern findet über die eine Benutzerschnittstelle statt. Als Dokumente sind die Vorgaben der IU für dieses studentische Projekt relevant.

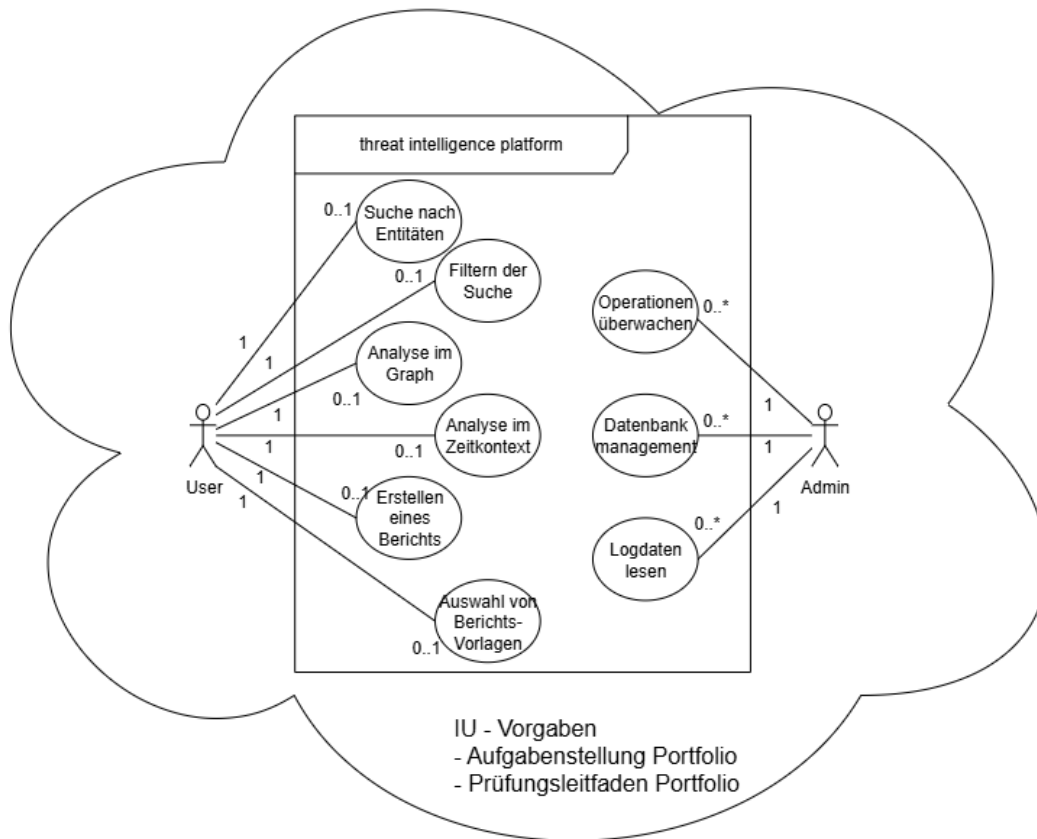


Abbildung 2.3: Analyse des Systemkontext, erstellt mit draw.io

## 2.3 Nicht-funktionale Anforderungen

**NFA-01:** Intuitive Visualisierung

**US-NFA-01:** Als Threat Intelligence-Analyst:in, ohne tiefgehendes technisches Vorwissen, möchte ich visuelle Darstellungen wie Graphen und Zeitachsen leicht verständlich nutzen können, um schnell Zusammenhänge zu erkennen.

**Beschreibung:** Die Visualisierung verwendet klare Farben und Labels. Elemente sind interaktiv (Zoom, Tooltips, Hover) und bieten kontextuelle Erklärungen.

**Akzeptanzkriterien:**

- Jeder Entitätstyp hat eine eindeutige Farbe (z. B. ThreatActor: rot, Malware: schwarz, Campaign: orange)
- Beziehungen werden mit beschrifteten, gerichteten Pfeilen dargestellt

- Informationen zu den Entitäten können per Links-Klick abgerufen werden. Sie werden in einem Informationsmenü dargestellt.
- Knoten können per Drag-and-Drop neu angeordnet werden.

**Hinweis zu NFA-02:** Die geplante Anforderung NFA-02 (Maximale Ladezeit für Graphen) wurde während der Entwicklung als nicht relevant eingestuft. Neo4j's Graph-Traversierung erfolgt mit  $O(1)$  Komplexität pro Hop (Khan, 2016). Bei der implementierten Limitierung auf maximal 100 Knoten und 3 Hops Tiefe treten keine relevanten Verzögerungen auf. Die Performance des Zielsystem kann zudem nicht durch den Entwickler kontrolliert werden.

**NFA-03:** Eingabevalidierung<sup>1</sup>

**US-NFA-03:** Als Administrator:in möchte ich sicherstellen, dass Benutzereingaben validiert werden, um Cypher-Injection und zu verhindern.

**Beschreibung:** Alle Benutzereingaben in der Suchleiste werden gegen definierte Regeln validiert, normiert und escaped, bevor sie an das Backend übermittelt werden. Dies verhindert die direkte Einbindung von Nutzereingaben in Cypher-Queries.

**Akzeptanzkriterien:**

- Eingaben sind zwischen 3 und 100 Zeichen lang
- Erlaubte Zeichen: A–Z, a–z, 0–9, . - \_ @ sowie Leerzeichen
- Frontend führt Unicode-Normalisierung und Trimming durch
- Eingaben werden in Lowercase standardisiert
- Keine direkte String-Interpolation in Cypher-Queries (Verwendung von Parametern)

**Hinweis zu NFA-04:** Die geplante Anforderung NFA-04 (Wiederverwendbarer Neo4j-Driver) wurde während der Entwicklung als nicht relevant eingestuft. Neo4j bietet einen offiziellen Treiber für Python. Dieser ist im System durch einen Wrapper (`GraphDBDriver`) eingebunden.

**NFA-05:** Lesezugriff für Nutzer:innen

**US-NFA-05:** Als Administrator:in möchte ich, dass Benutzer:innen die Daten nicht verändern können, um die Integrität der Informationen sicherzustellen.

**Beschreibung:** Nutzer:innen haben nur lesenden Zugriff auf die Daten.

**Akzeptanzkriterien:** Schreibende Operationen sind nur Admins vorbehalten. Nutzer:innen haben nur Lesezugriff.

**NFA-06:** Wartbarkeit durch modulare Architektur

---

<sup>1</sup>Diese Anforderung ersetzt die ursprünglich geplante NFA-03 (Asynchrone Verarbeitung). Da für diese Plattform kein Import über das Frontend vorgesehen ist, existiert kein Anwendungsfall. Die Eingabevalidierung wurde während der Entwicklungsphase als kritischer für die Sicherheit der Anwendung identifiziert.

**US-NFA-06:** Als Entwickler:in möchte ich, dass die Codebasis modular aufgebaut ist, um Änderungen effizient und risikoarm vornehmen zu können.

**Beschreibung:** Das System ist in klar getrennte Module gegliedert (z.B. Datenzugriff, Visualisierung, Berichtserstellung), sodass Änderungen lokal begrenzt und testbar sind.

**Akzeptanzkriterien:** Codeänderungen in einem Modul erfordern keine Anpassungen in anderen. Module sind entkoppelt. Es existieren Tests pro Modul mit mindestens 80% Testabdeckung.

**NFA-07:** Dokumentation

**US-NFA-07:** Als neue:r Entwickler:in möchte ich über eine vollständige technische Dokumentation verfügen, um mich schnell in das System einarbeiten und es effizient weiterentwickeln zu können.

**Beschreibung:** Die Entwickler:innen-Dokumentation umfasst Architekturdiagramme, API-Beschreibungen sowie Setup-Anleitungen.

**Akzeptanzkriterien:** Die technische Dokumentation ist vollständig im Repository enthalten. Der Quellcode ist kommentiert.

**NFA-08:** Benutzerfreundliches Handbuch für Endnutzer:innen

**US-NFA-08:** Als Nutzer:in möchte ich auf ein Handbuch zugreifen können, um die Funktionen des Systems nutzen zu können.

**Beschreibung:** Das Benutzerhandbuch beschreibt zentrale Anwendungsfälle wie Suche, Visualisierung, Berichtsgenerierung und Konfiguration der Ansicht, ergänzt durch Screenshots und Beispiele.

**Akzeptanzkriterien:** Es existiert ein als PDF-Handbuch mit: Startanleitung, Suchbeispiele, Visualisierungsoptionen und Berichtserstellung. Die Anleitung ist mit Screenshots versehen.

Priorität	ID	Name	ISO-Kriterium	Unterkategorie
Hoch	NFA-01	Intuitive Visualisierung	Benutzerfreundlichkeit	Bedienbarkeit
Hoch	NFA-03	Eingabevalidierung	Sicherheit	Input-Validierung
Hoch	NFA-05	Lesezugriff für Analyst:innen	Sicherheit	Zugangskontrolle
Mittel	NFA-06	Wartbarkeit durch modulare Architektur	Wartbarkeit	Modularität
Mittel	NFA-07	Technische Dokumentation	Wartbarkeit	Analysierbarkeit
Mittel	NFA-08	Benutzerfreundliches Handbuch	Benutzerfreundlichkeit	Erlernbarkeit

Tabelle 2.2: Nicht-funktionale Anforderungen nach ISO/IEC 25010

## 3 Spezifikationsdokument

### 3.1 Datenmodell

Für die Analyse von Threat Intelligence-Daten eignet sich ein Graphen-Datenmodell. Ziel einer Analyse ist beispielsweise, ob eine bestimmte IP-Adresse auf einen Angreifer hindeutet. Für eine solche Analyse müssen viele Informationen kombiniert werden. Diese Kombination lässt sich übersichtlich in einem gerichteten Graph mit Nodes und Beziehungen darstellen. In diesem Datenmodell werden die Nodes als Klassen aufgebaut und mit einem englischen Substantiv bezeichnet. Die Beziehungen werden zur Abstraktion hier als Assoziationsklassen geführt und mit einem englischen Verb in Großbuchstaben bezeichnet. Die Datenmodellierung in Neo4j folgt den offiziellen Namenskonventionen Neo4j-Inc., 2025b:

- **Knoten-Labels:** PascalCase, Singular (z. B. `ThreatActor`, `Malware`)
- **Beziehungen:** SCREAMING\_SNAKE\_CASE (z. B. `USES`, `BASED_ON`)

Der Aufbau des Datenmodells orientiert sich an STIX v2.1 (Struse und Darley, 2019). Dieser Standard ist in diesem Projekt jedoch nicht vollständig umgesetzt. Die Bewertung der Zuverlässigkeit der Informationen ist in diesem Datenmodell nicht umgesetzt.

#### Node-Klassen

Die Node-Klassen stellen Konzepte dar, die typischerweise in der Threat Intelligence verwendet werden.

##### Klasse `AttackPattern`

Ein Angriffsmuster beschreibt Taktiken, Techniken und Verfahren (TTP), die Angreifer nutzen um ein Ziel anzugreifen. Sie spezifizieren ein bestimmtes Vorgehen, oft auf Basis von MITRE ATT&CK (MITRE, 2025).

Die Klasse besitzt folgende Attribute: Name und Beschreibung. Ein Angriffsmuster kann beispielsweise von einer Malware oder einer Campaign benutzt oder von einem Report beschrieben werden.

##### Klasse `Campaign`

Eine Kampagne bezeichnet eine zusammenhängende Operation oder Angriffsserie, häufig auf bestimmte Ziele ausgerichtet. Sie wird Umgangssprachliche auch als „Welle“ bezeichnet.

Die Klasse besitzt folgende Attribute: Name, Beschreibung, Beginn- und Startzeitpunkt. Eine Kampagne kann beispielsweise von einem Threat Actor gestartet werden. Sie kann ein An-

griffsmuster benutzen und Malware einsetzen.

### **Klasse Identity**

Die Identität beschreibt eine natürlichen oder juristischen Person im Kontext der Bedrohungslage. Die Klasse besitzt folgende Attribute: Name, Rolle, Kontaktdaten. Ein Threat Actor oder eine Organisation können beispielsweise mit einer Identität verknüpft sein.

### **Klasse Incident**

Ein Incident bezeichnet einen konkreten Sicherheitsvorfall mit beobachteten Auswirkungen.

Die Klasse besitzt folgende Attribute: Name, Beschreibung, Erkennungsdatum und Beseitigungsdatum. Ein Incident kann beispielsweise durch einen Indicator erkannt werden. Ein Incident muss auf eine Organisation zielen.

### **Klasse Indicator**

Ein Indikator ist eine Auffälligkeit in einem System, dass auf eine schadhafte oder bösartige Aktivität hindeutet. Ein Indikator muss auf mindestens einer Beobachtung basieren. Der Indikator ist die Bewertung dieser Beobachtungen.

Die Klasse besitzt folgende Attribute: Name, Beschreibung, erstes und letztes Auftreten. Ein Indikator ist immer mit mindestens einer Beobachtung über die Beziehung „basiert auf“ verknüpft.

### **Klasse Malware**

Eine Malware wird auch als Schadsoftware bezeichnet. Sie wird oft über ihren Typ (beispielsweise Trojaner, Locker, Crypto-Ransomware, ...) und ihrer Familienzugehörigkeit (beispielsweise Lockbit, Trickbot, Emotet, njRat, ...) eingeordnet.

Die Klasse besitzt folgende Attribute: Name, Familie, Typ, Beschreibung, erstes und letztes Auftreten. Eine Schadsoftware kann beispielsweise durch einen Indikator erkannt oder durch einer Kampagne genutzt werden.

### **Klasse Observable**

Ein Observable ist ein beobachtbares technisches Artefakt. Es kann sich beispielsweise um den Hashwert einer Datei, eine IP-Adresse oder eine E-Mail-Adresse handeln. Es stellt eine reine Beobachtung dar. Die Klasse Observable ist eine Container-Klasse, die ein technisches Artefakt enthält. Folgende technische Artefakte sind im Datenmodell integriert: E-Mail-Adresse, IPv4, Datei, URL, Domain. Die Klasse Observable besitzt die Attribute: Auftreten, Name und Beschreibung. Die Klassen der technischen Artefakte enthalten Attribute für die spezifische Beschreibung.

So kann ein Indicator of Compromise (IoC) basierend auf einer IP-Adresse durch eine Instanz der Klasse `Indicator - BASED_ON -> Observable - BASED_ON -> IPv4Address` dargestellt werden. Beobachtung und Interpretation werden getrennt und somit ist ein Grundprinzip der Threat Intelligence eingehalten.

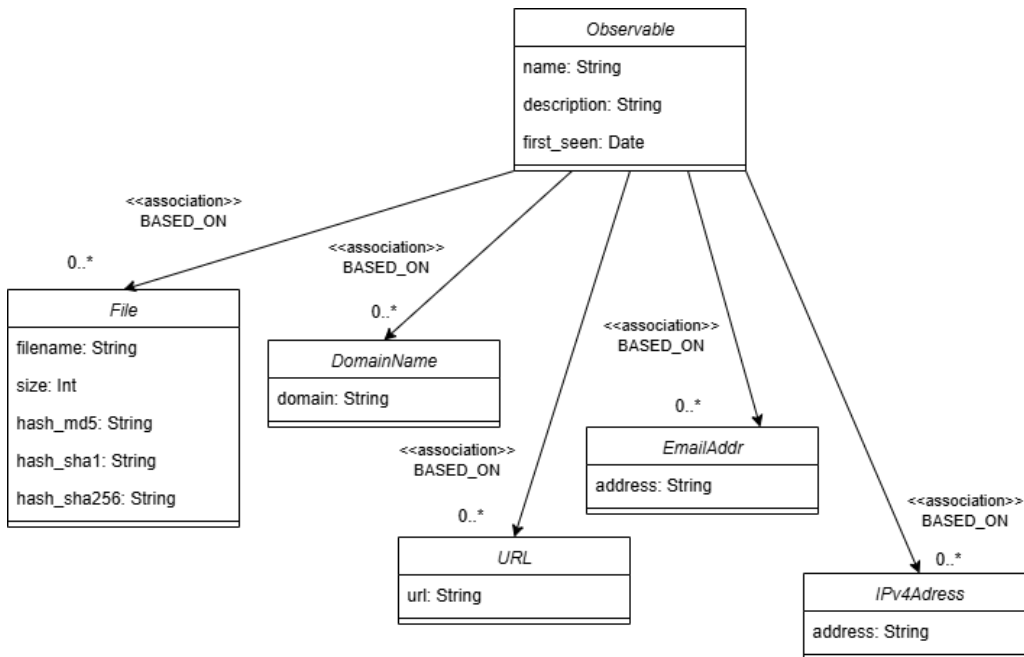


Abbildung 3.1: Datenmodell der Container-Klasse Observable und die Klassen der Artefakte.

### Klasse Organization

Eine Organisation ist das Ziel oder die betroffene Einheit eines Angriffs. Bei einer Organisation kann es sich beispielsweise um ein Unternehmen oder eine öffentliche Einrichtung handeln.

Die Klasse besitzt folgende Attribute: Name, Sector, Region und Beschreibung. Eine Organisation kann beispielsweise Ziel eines Incidents sein.

### Klasse Report

Reports werden von Sicherheitsforschern und Analysten erstellt. Sie sind eine Informationssammlung zu einem sicherheitsrelevanten Thema. Beispielsweise können Informationen über einen bestimmten Sicherheitsvorfall enthalten, über einen Threat Actor informieren oder die Signifikanz einer Sicherheitslücke behandeln. Ein Report ist eine Basis für vielseitige Informationen.

Die Klasse besitzt folgende Attribute: Title, Beschreibung, Quelle und Erscheinungsdatum. Ein Report kann mit jeder anderen Entität über die Beziehung „beschreibt“ verknüpft sein.

### Klasse ThreatActor

Ein Threat Actor ist ein individueller oder kollektiver Angreifer, welcher gezielte Aktivitäten gegen Organisationen oder Systeme durchführt.

Die Klasse besitzt folgende Attribute: Name, Aliase, Typ, Motivation, Beschreibung, erstes und letztes Auftreten. Ein Threat Actor kann beispielsweise über die Beziehung „benutzt“ mit einem Werkzeug oder über „startet“ mit einer Kampagne verknüpft werden.

### Klasse Tool

Ein Tool beschreibt ein legitimes oder speziell entwickeltes Werkzeug zur Interaktion mit Systemen. Es kann sich dabei um Software- oder Hardwaretools handeln. Im Threat Intelligence-Kontext werden solche Werkzeuge zur Durchführung von Angriffen benutzt.

Die Klasse besitzt folgende Attribute: Name, Version und Beschreibung. Ein Tool kann beispielsweise von einem Threat Actor benutzt oder einem Report beschrieben werden.

## Klasse Vulnerability

Eine Vulnerability ist eine Schwachstelle in Software, die ausgenutzt werden kann. Schwachstellen werden über eine CVE-ID identifiziert und über einen CVSS Score in ihrer Schwere bewertet.

Die Klasse besitzt folgende Attribute: CVE ID, CVSS-Score, Datum des Bekanntwerdens und Beschreibung. Eine Schwachstelle kann beispielsweise mit einem Observable in Verbindung stehen oder durch einen Report beschrieben werden.

## Beziehungen

Die Beziehungen sind als Assoziationsklassen abstrahiert. Sie verbinden Node-Klassen. Assoziationsklassen besitzen die Attribute Quelle sowie erstes und letztes Auftreten. Die Klasse „beschreibt“ (DESCRIBES), besitzt nur das Quellenattribut. Im Graphen-Modell geben die Art der Beziehung und die Richtung zusätzliche Informationen zum Gesamtkontext.

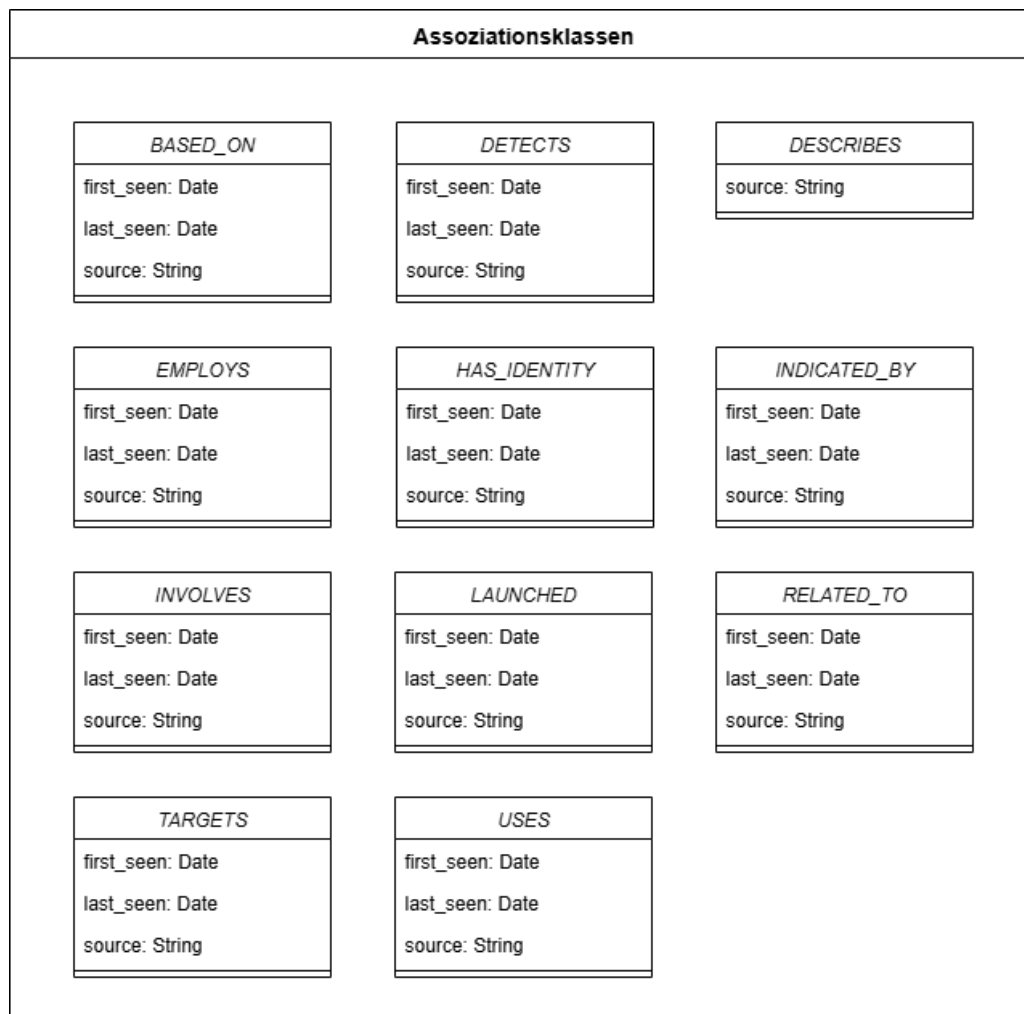
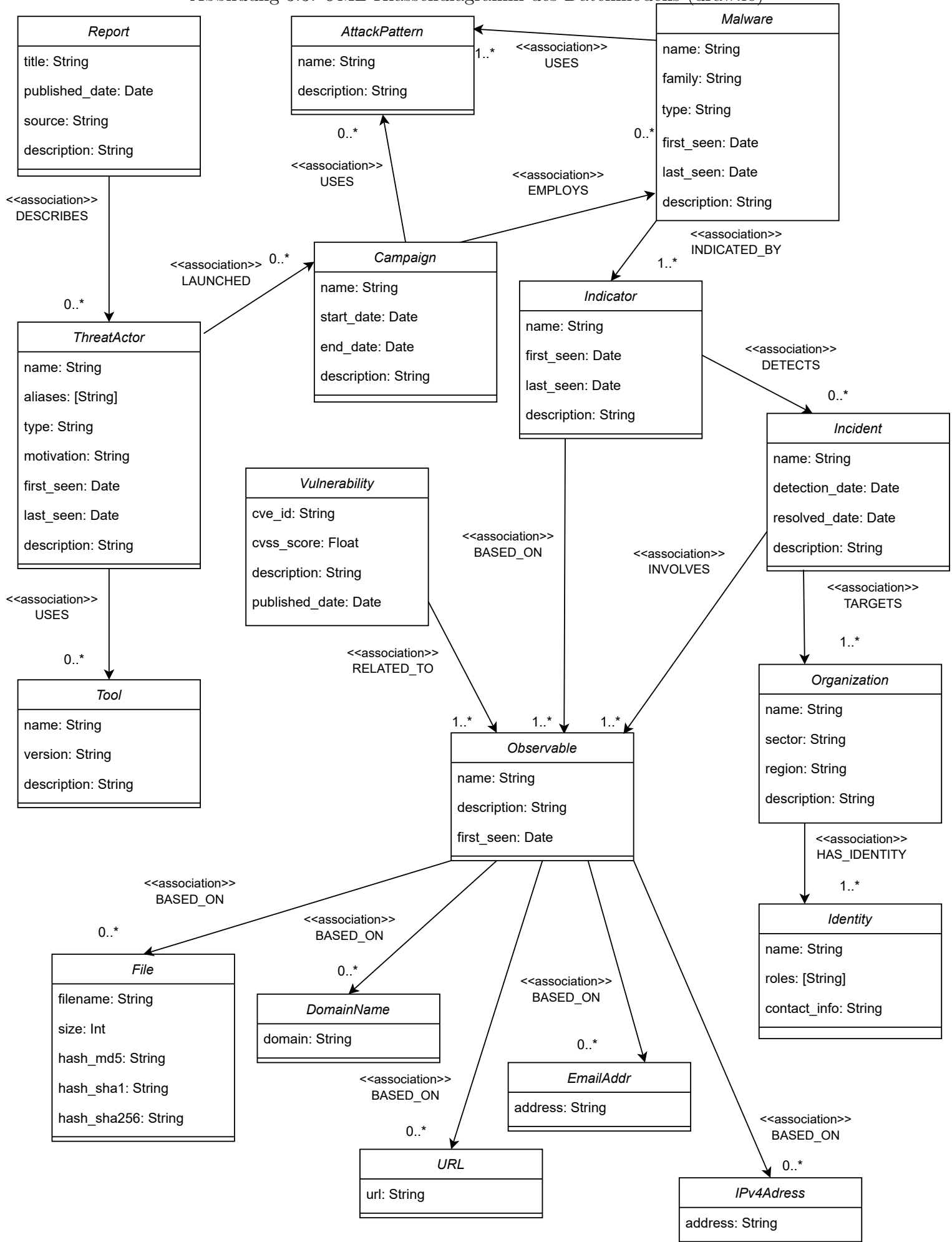


Abbildung 3.2: Übersicht über die Assoziationsklassen(draw.io)

Abbildung 3.3: UML Klassendiagramm des Datenmodells (draw.io)



Grundsätzlich sind alle Beziehungen zwischen den Nodes möglich. Ein Report kann beispielsweise als Quelle jede Node beschreiben. Im folgenden UML-Klassendiagramm ist eine beispielhafte Verknüpfung der Klassen zu sehen.

## Graph-Datenmodell

Hier ist das Datenmodell als Graph durch die Datenbank Neo4j modelliert.

Es bietet eine alternative Perspektive auf die Zusammenhänge.

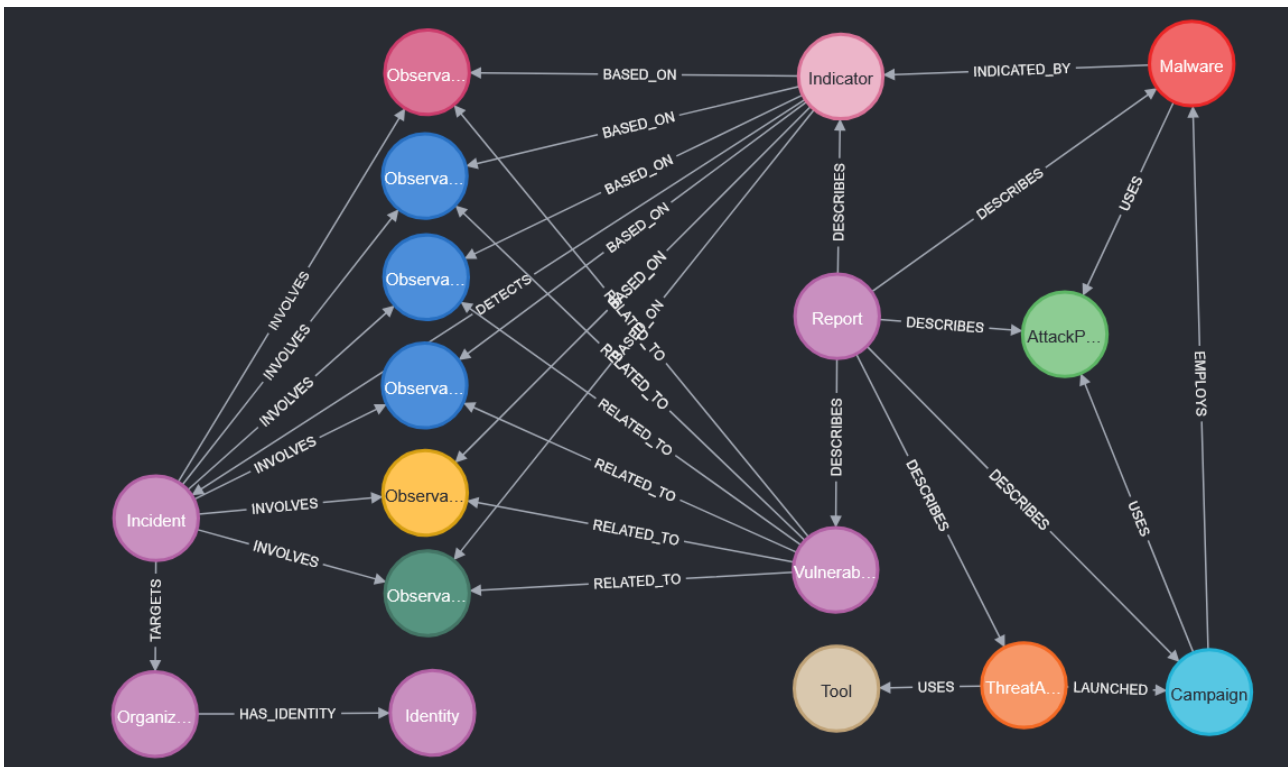
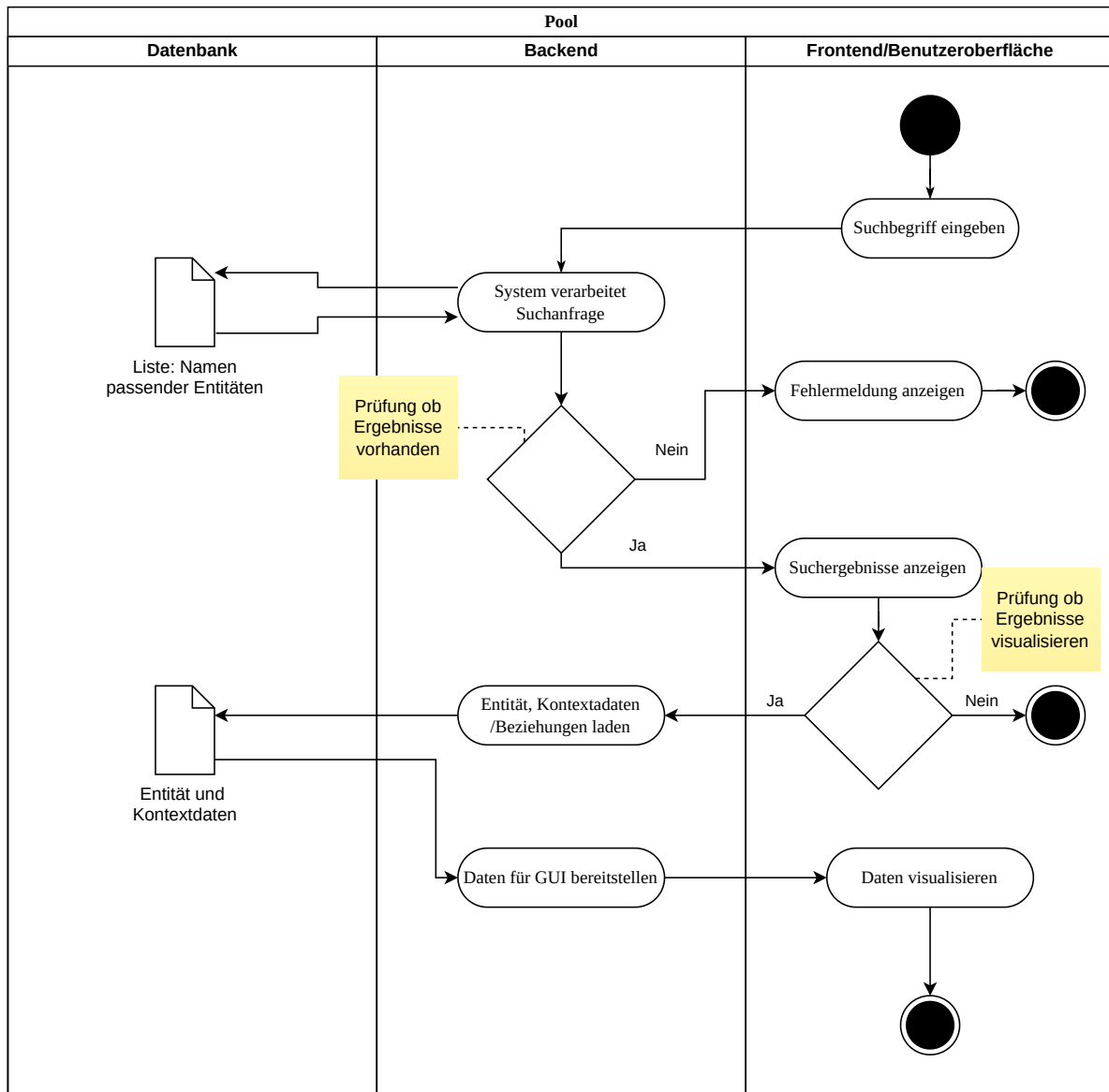


Abbildung 3.4: Datenmodell als Graphenmodell erstellt mit Neo4j

## 3.2 Geschäftsprozesse

**GP-01: Suche und Visualisierung** Nutzer:innen können nach Entitäten wie Threat Actors, Malware oder TTPs suchen. Die Suchanfrage wird an das Backend weitergeleitet, dort wird ein Anfrage an die Datenbank gestellt, ob Ergebnisse vorhanden sind. Trifft dies zu wird in der Sucheingeabe das Ergebnis angezeigt und kann mit einer Eingabe visualisiert werden. Sind keine Daten vorhanden erscheint eine Fehlermeldung. Wird die Visualisierung ausgeführt, werden die Entitäten, die zusammenhängenden Kontextdaten und Beziehungen vom Backend angefordert. Die Daten werden aus der Datenbank geladen und sie werden für die Visualisierung im Frontend bereitgestellt. Die Ergebnisse werden in der Graphen-Ansicht visualisiert. So wird ein Überblick über relevante Zusammenhänge zu Suchbegriff erzeugt.

Abbildung 3.5: UML-Aktivitätsdiagramm für den zentralen Prozess GP-01, Suche und Visualisierung (draw.io)



**GP-02: Erweitern einer Entität** Die Visualisierung erlaubt es, eine Entität durch aktive Auswahl zu erweitern. Nutzer:innen können rechts auf die Entität klicken und „erweitern“ auswählen. Dann werden alle Verbindungen dieser Entität mit einem Hop Entfernung geladen.

**GP-03: Reduzieren einer Entität** Um die Übersichtlichkeit zu wahren, kann die Kontextvisualisierung gezielt reduziert werden. Nutzer:innen können rechts auf eine Entität klicken und „reduzieren“ auswählen. Dann wird die Entität und ihre Verbindungen ausgeblendet.

**GP-04: Zeitstrahl-Visualisierung** Optional können Benutzer:innen Entitäten mit einem Zeitbezug auf einer Zeitachse darstellen. Mit dem Button „Time-Analysis“ wird ein Zeitstrahl mit allen Entitäten generiert, die temporale Daten besitzen.

**GP-05: Bericht generieren** Benutzer:innen können auf Basis der aktuellen Visualisierung und Auswahl einen Bericht generieren. Dies wird über ein Kontextmenü erreicht. Der Bericht enthält ausgewählte Entitäten, Beziehungen und Metadaten.

**GP-06: Datenimport durch Administrator:innen** Daten werden ausschließlich durch Administrator:innen über ein Terminal-Skript oder Kommandozeilentool importiert. Dabei wird eine JSON-Datei verwendet. Vor dem Import prüft das System automatisch auf Datenintegrität, Dubletten und erforderliche Pflichtfelder. Der Importvorgang protokolliert sämtliche Änderungen in einem Audit-Log. Treten hier Probleme auf müssen die problematischen Daten durch die Administrator:in manuell bewertet werden.

### 3.3 Geschäftsregeln

#### **GR-01 Eindeutige IDs**

Jede Entität und Beziehung muss eine eindeutige ID besitzen.

#### **GR-02 Validierung beim Import**

Vor dem Einfügen in die Datenbank müssen Daten validiert werden: Format, Konsistenz und Dublettenprüfung sind verpflichtend.

#### **GR-03 Beziehungen nur zwischen existierenden Entitäten**

Beziehungen dürfen nur zwischen existierenden Entitäten angelegt werden. Verwaiste Relationen sind unzulässig.

#### **GR-04 Mindestlänge für Suchanfragen**

Eine Suchanfrage muss mindestens drei alphanumerische Zeichen enthalten.

#### **GR-05 Initiale Kontexttiefe**

Die initiale Kontextvisualisierung ist auf eine Beziehungsebene (1 Hops) beschränkt. Eine Erweiterung erfordert eine aktive Benutzeraktion.

#### **GR-06 Minimalanforderungen Bericht**

Ein Bericht darf nur erzeugt werden, wenn mindestens eine Entität und deren Kontextdaten vorhanden sind.

## **GR-07 Versionierung und Zeitstempel für Berichte**

Berichte erhalten eine Versions ID und einen Zeitstempel.

## **GR-08 Lesezugriff für das Frontend**

Das Frontend erlaubt ausschließlich lesenden Zugriff.

# **3.4 Systemschnittstellen**

## **Server-Client Kommunikation**

Die Plattform verwendet eine Client-Server Architektur mit einer Schnittstelle zwischen dem Python-basierten Backend (Server) und dem Svelte-Frontend (Client). Diese Schnittstelle stellt die Datenübertragung sicher. Sie ermöglicht eine standardisierte und erweiterbare Kommunikationsstruktur.

Die Kommunikation findet über HTTP statt. Da das Frontend in diesem Projekt über `https://localhost` läuft ist HTTPS nicht erforderlich. Für eine Anwendung im Internet wäre HTTPS unumgänglich.

Die Client-Server Kommunikation findet über Request-Response statt. Der Architekturstil ist REST. Die HTTP-Methoden sind eingeschränkt da auf dem Frontend nur lesender Zugriff möglich sein soll. Daher werden nur GET implementiert. DELETE, PUT, POST sind nicht vorgesehen da diese Funktionen im Backend administrativ gesteuert werden. Als Dateiformat wird JSON verwendet. Das Schema ist in Anlage 5.2 aufgeführt.

Zur Erstellung wurde das Metaschema 07 (Wright und Andrews, 2018) genutzt.

## **Datenbank-Backend Kommunikation**

Die Neo4j-Datenbank dient der dauerhaften Speicherung von Daten im System. Der Zugriff findet über einen Neo4j-Python Treiber statt. Für Neo4j wird die Query-Sprache Cypher verwendet.

### 3.5 Benutzerschnittstellen

Die Benutzerschnittstelle (GUI) wird über das Web-Frontend realisiert. Die erste Dialogmaske zeigt die Überschrift, ein Textfeld für die Suche und zwei Buttons. Der erste Button trägt wird für das Absenden der Suche genutzt. Er trägt ein „Lupe“-Icon zur Beschreibung der Funktion. Der zweite Button trägt ein „Zahnrad“-Icon. Mit diesem Button wird ein Kontextmenü für Einstellungen aufgerufen. Im Footer der Seite sind die Version der Plattform, das Copyright und ein Link zu Impressum festgehalten.

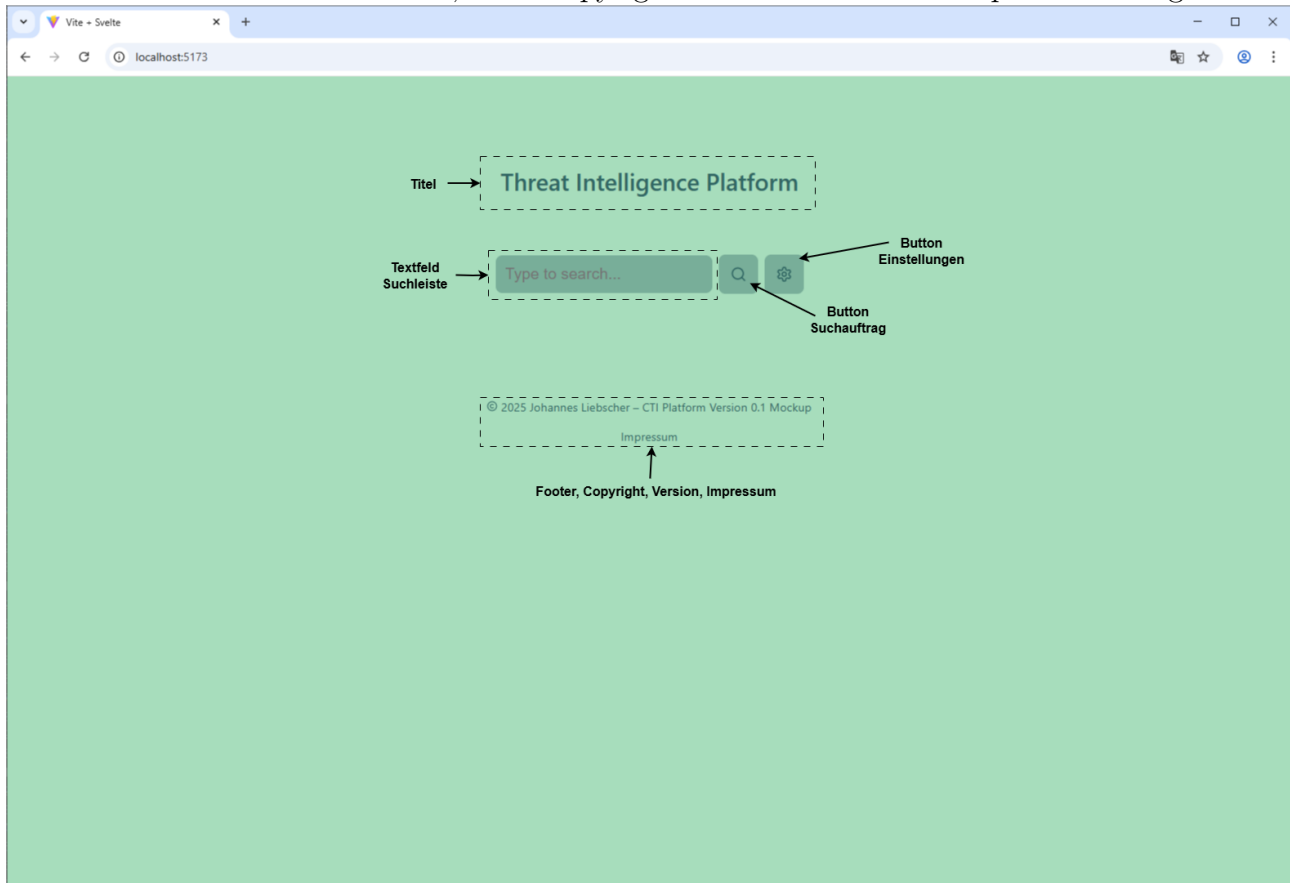


Abbildung 3.6: Mockup des ersten GUI Zusatandes

Wird die Suche ausgeführt, ordnen sich die Elemente der Oberfläche neu an. Es erscheint die interaktive Graphen Ansicht zwischen der Suchleiste und dem Footer. Neben den zwei bisherigen Buttons erscheinen zwei weitere, der erste für die Zeitanalyse mit dem „Chart“-Icon und der zweite für die Berichtsgenerierung mit einem „Datei“-Icon.

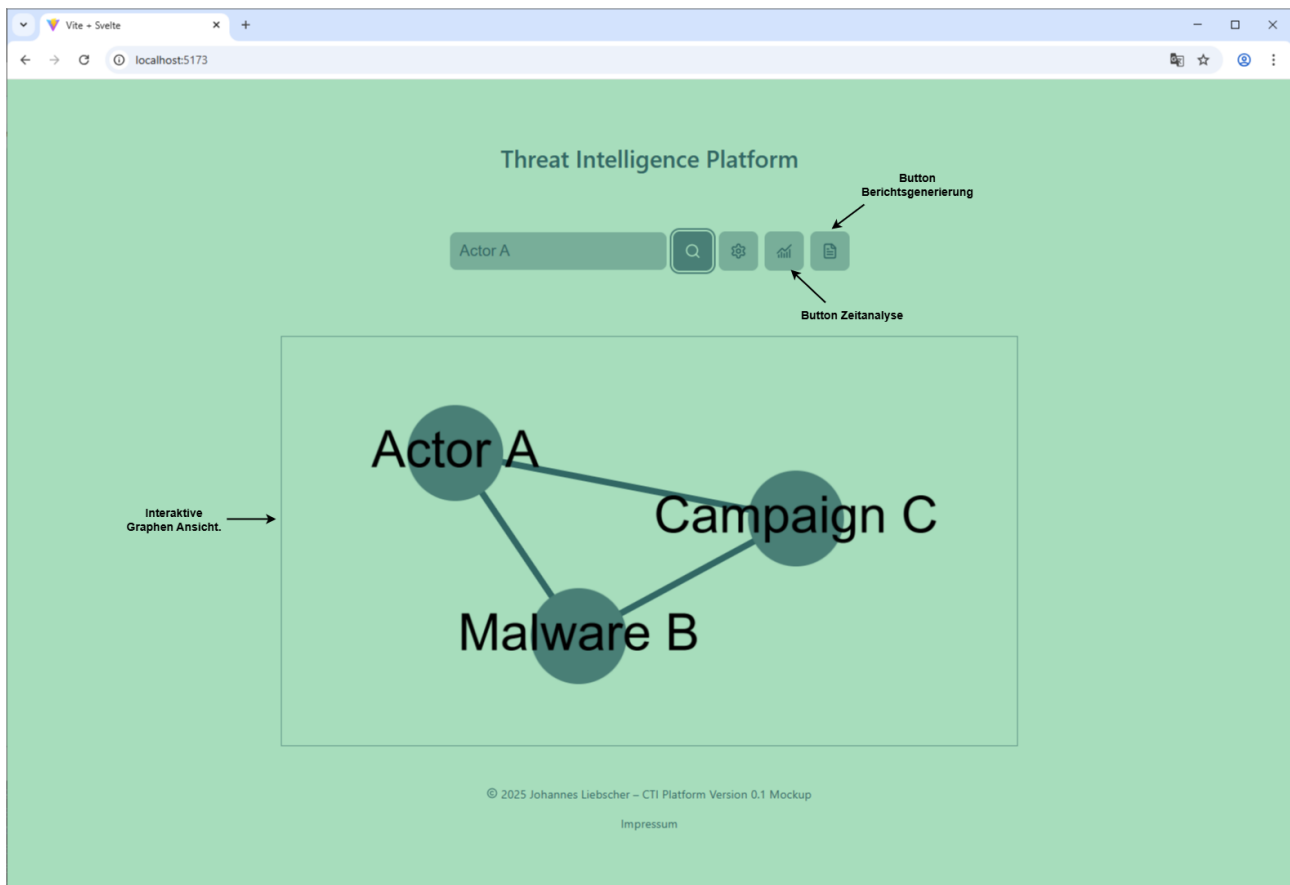


Abbildung 3.7: Mockup des zweiten GUI Zusatzandes

Über den „Zeitanalyse“-Button wird ein Zeitstrahl erstellt welcher unter der Graphen-Ansicht angezeigt wird. Auf diesem Zeitstrahl werden alle Elemente der Suche, die Zeit-Attribute besitzen, chronologisch dargestellt.

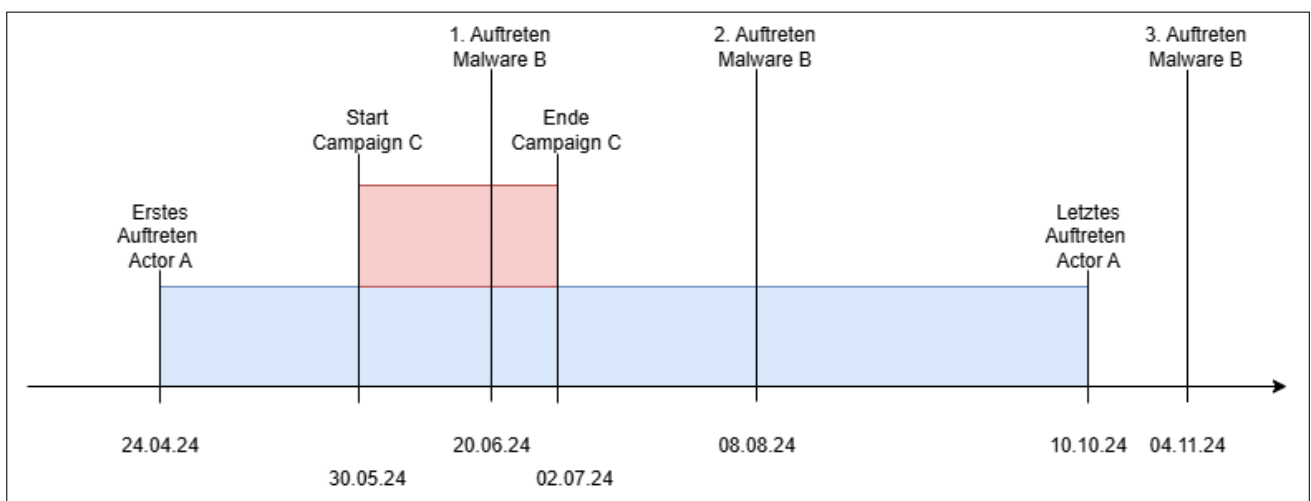


Abbildung 3.8: Skizze der Zeitstrahlanalyse

Über den Button Berichtsgenerierung wird geöffnet sich ein Kontextfenster in welchen eine Vorlage ausgewählt werden kann. Anschließend können gesuchte Elemente hinzugefügt werden. Es ist auch möglich einen Ausschnitt aus dem Graphen an den Bericht anzufügen. Der fertige Bericht wird dann zum Download zur Verfügung gestellt.

## Eingabevalidierung

Die Suchleiste erlaubt Benutzer:innen die Eingabe von Suchbegriffen zur Identifikation von Entitäten im System. Um Missbrauch durch beispielsweise Cypher-Injection zu verhindern, müssen die Eingaben validiert werden.

Validierungsregeln:

- Mindestlänge: 3 Zeichen
- Maximallänge: 100 Zeichen
- Erlaubte Zeichen: (A–Z, a–z, 0–9, .-\_@)
- Leerzeichen sind zwischen Zeichen erlaubt, aber nicht am Anfang oder Ende
- Nicht erlaubte Zeichen: Steuerzeichen, Emojis, Steuersequenzen, Codefragmente

Maßnahmen:

- Normierung der Eingabe
- Unicode Normalisierung
- Trimmen von führenden und nachfolgenden Leerzeichen
- Standardisierung in Lowercase für die Verarbeitung
- Vor Übergabe an das Backend: Escape aller Eingaben
- Keine direkte Einbindung in Queries.
- Ungültige Eingaben zeigen der Benutzer:in eine Fehlermeldung

## 4 Architekturdokument

### 4.1 Technologieübersicht

#### Neo4j

Neo4j ist eine Graphdatenbank, die Daten als Knoten und Beziehungen speichert. In diesem Projekt wird die Community Edition verwendet. (Neo4j-Inc, 2025a) Diese ist mit der GPLv3 Lizenz veröffentlicht (Neo4j-Inc., 2025c). Eine Graphdatenbank eignet sich deshalb besonders gut für Threat Intelligence Daten, da diese relational vernetzt sind. Wenn ausgewertet wird, sind oft Ketten von Zusammenhängen wichtig. Neo4j speichert die Links eines Knotens zu vernetzten Knoten direkt im Knoten selbst. Daher ist die Traversierung von Knoten zu Knoten konstant  $O(1)$ , unabhängig von der Gesamttiefe (Khan, 2016). Eine lange Kette von Knoten und Beziehungen lässt sich daher sehr effizient auswerten. Bei einer SQL-Datenbank wären hier für jeden Knoten Index-look-ups und JOIN Operationen nötig. Neo4j verwendet die Abfragesprache Cypher.

#### Python

Python ist unter der Python Software Foundation License Version 2 veröffentlicht (Python-Software-Foundation, 2025a). Das Backend ist in Python geschrieben, da hier bereits Kenntnisse des Entwicklers vorliegen. Python bietet Bibliotheken wie Flask (Ronacher, 2025b) für eine einfache API und ist durch Neo4j mit einem offiziellen Treiber unterstützt. Benutzte Standardbibliotheken (Python-Software-Foundation, 2025b):

1. **os:** wird benutzt um Umgebungsvariablen zu importieren.
2. **sys:** wird in main.py benutzt um zu verhindern, dass das Backend startet wenn keine Datenbankverbindung möglich ist.
3. **logging:** bietet ein logging framework. Es wird im Backend für Terminal-Informationen und Debugging benutzt. Die Einrichtung für das System befindet sich in logger.py.
4. **typing:** aus dieser Bibliothek wird hauptsächlich Dict, List, Any und Optional verwendet, um ein flexibles befüllen und auslesen der Datenbank zu ermöglichen. Typecasts werden erst zu Laufzeit durchgeführt, viele mögliche Eigenschaften der Knoten sind optional.
5. **enum:** wird benutzt um typensichere und validierbare Strings festzulegen.
6. **pathlib:** findet Anwendung, bei der Übergabe von Pfaden, beispielsweise dem Pfad der JSON-Datei, beim initialen befüllen der Datenbank.

7. **dataclasses:** wird für die Generierung der ImportResult-Klasse in `import_service.py` verwendet. Diese fasst den aktuellen Import zusammen.
8. **datetime:** wird beim Import für Messung der Dauer verwendet.
9. **json:** bietet Codierung und Decodierung für JSON-Dateien. Das Backend arbeitet beim Import und über Web-Requests nur mit JSON.
10. **random:** wird zur Generierung von zufälligen Beispieldaten verwendet.
11. **argparse:** wird zum lesen optionaler Kommandozeilenargumente verwendet. Die Generierung der Beispieldaten erlaubt es die Anzahl der Knoten festzulegen, den Seed anzuzeigen und die Ausgabedatei zu bestimmen.
12. **tempfile:** findet Anwendung beim testen. Mit `tempfile` werden temporäre JSON-Dateien für verschiedene Testfälle erstellt.
13. **unittest:** wird zum Erstellen von Mockdaten benutzt.

Verwendete Third-Party-Libraries:

1. **neo4j:** Offizieller Treiber für die Neo4j-Graphdatenbank (Neo4j-Inc., 2025c), veröffentlicht unter Apache 2.0 Lizenz (Neo4j-Inc, 2025d).
2. **Flask:** Webframework für das Backend (Ronacher, 2025b). Flask eignet sich aufgrund der Unterstützung für Routing und REST-APIs sowie der flexiblen Projektstruktur ohne erzwungene Konventionen. Lizenz: BSD-3-Clause (Ronacher, 2010).
3. **flask-cors:** Flask-Erweiterung für CORS-Handling (Dolphin, 2025a). Sie wird für CORS benötigt, da die Kommunikation mit dem Frontend über verschiedene Ports und URIs abläuft. Lizenz: MIT (Dolphin, 2025b).
4. **werkzeug:** Bibliothek von Entwicklungswerkzeugen für WSGI-Anwendungen. Im System wird `werkzeug.exceptions` für einheitliche HTTP-Fehlermeldungen und Statuscodes verwendet (Ronacher, 2025a). Lizenz: BSD-3-Clause (Ronacher, 2007).
5. **pytest:** Python-Testframework mit Fixture-System und Testautomatisierung (Krekel, 2015b). Es wurde aufgrund der einfachen Implementierung und des Fixture-Mechanismus ausgewählt. Lizenz: MIT (Krekel, 2015a).

## Svelte

Svelte ist ein Frontend-Framework, dass zur Build-zeit in optimierten Javascript-Code kompiliert wird, anstatt eine Runtime library zu verwenden (Svelte, 2025a). Dies ermöglicht eine gute Performance auch bei großen interaktiven Graphen mit hunderten Knoten und Beziehungen. Svelte ist unter der MIT-Lizenz veröffentlicht (Svelte, 2025b). Verwendete Funktionen aus der Svelte Standardbibliothek:

1. **mount:** wird auf Frameworkebene verwendet um die Haupt-App-Komponente einzubinden.

2. **onMount:** wird auf Komponentenebene als Anker genutzt, um eine Komponente ins DOM zu setzen. Dies wird in diesem Projekt für alle Frontendkomponenten wie Graphview und Timeline Analysis verwendet.
3. **onDestroy:** ist der Anker für „clean-up“ Operationen.
4. **createEventDispatcher:** ermöglicht Events zwischen Komponenten. Beispielsweise das Auswählen eines Zeitstrahl-Elements in der Timeline Analysis und das Hervorheben des korrespondierenden Knotens in Graphview.

Benutzte Third-Party-Libraries:

1. **lucide-svelte:** Open-souce Bibliothek für Vektorgrafiken und Symbole (Bemis, 2025a). Aus dieser Bibliothek werden die Symbole „Search“, „Settings“, „FileText“, „ChartNoAxesCombined“, „Copyright“ und „X“ verwendet. Lucide Symbole sind visuell ansprechend und aussagekräftig. Lucide-Svelte ist unter der ISC Lizenz veröffentlicht (Bemis, 2025b).
2. **cytoscape.js:** Bibliothek zum rendern von interaktiven Graphen in Javascript (Franz et al., 2016). In diesem Projekt werden die Threat Intelligence Daten in der Frontend-Komponente Graphview durch diese Bibliothek gerendert. Cytoscape ist unter der MIT Lizenz veröffentlicht (Franz, 2024). Cytoscape wird aufgrund der Performance bei Graphen mit vielen Knoten und der einfachen Implementierung interaktiver Elemente ausgesucht.
3. **Charts.js mit chartjs-adaptor-date-fns:** Bibliothek für das erstellen von einfachen Graphen (Downie, 2025). Sie wird für das Erstellen des Graphen im Timeline Analysis Komponenten verwendet. Charts.js ist unter der MIT-Lizenz veröffentlicht (Swayne, 2024). Chart.js ist ausreichend um einen aussagekräftigen Graphen für die Zeitanalyse zu erstellen. Es gibt alternative Bibliotheken wie beispielsweise Plotly.js oder D3.js, diese haben mehr Funktionen, welche für den Zeitanalyse-Graph aber nicht nötig sind.
4. **vitest:** Testframework für Frontends in vite (Sheremet et al., 2025). Es wird für Test-automatisierung des Frontends und Mocking verwendet. Vitest ist unter der MIT-Lizenz veröffentlicht (Sheremet, 2025). Es wurde aufgrund der Nutzung von vite und der Automatisierung von Unittests ausgewählt.

## HTML, CSS, JS

HTML ist als Teil einer Svelte-Komponente geschrieben und beim Build-Prozess in DOM-Code übersetzt. CSS ist global in `app.css` und local an den jeweiligen Komponenten eingesetzt. Komponenten-CSS ist automatisch isoliert, sodass keine Stilkonflikte entstehen. JavaScript wird in script-Blöcken jeder Komponente verwendet. Zusätzlich sind normale .js-Dateien, wie beispielsweise `inputValidation.js` eingebunden. Der Svelte-Compiler erzeugt daraus kompaktes, performantes Vanilla-JS ohne Virtual DOM. HTML, CSS und JavaScript basieren auf offenen Web-Standards, die von WHATWG, W3C bzw. ECMA definiert werden. Die Spezifikationen sind öffentlich und dürfen frei implementiert und genutzt werden (World-Wide-Web-Consortium, 2025 und Mozilla-Foundation, 2025).

## Werkzeuge

**Docker:** Docker abstrahiert das Betriebssystem, die Umgebung und die Abhängigkeiten. Es wird in diesem Projekt genutzt um eine portable und reproduzierbare Anwendung sicherzustellen. Die drei Server müssen plattformunabhängig und mit den richtigen Abhängigkeiten funktionieren. Über Docker werden zudem Entwicklungsumgebung, Testumgebung und Produktionsumgebung verwaltet. Genutzt wird die Docker-engine (Docker-Inc, 2025a). Docker ist für ein Projekt dieser Art unter der Apache 2.0 Lizenz veröffentlicht (Docker-Inc, 2025b).

**vite:** Build- und Entwicklungswerkzeug für Javascript (You, 2025a), lizenziert unter der MIT Lizenz (You, 2025b). Das Plugin vite-plugin-svelte (Göpel, 2025a) macht Svelte mit vite kompatibel, veröffentlicht unter der MIT Lizenz (Göpel, 2025b). Vite wurde aufgrund der Einfachheit und der guten Kompatibilität mit Svelte ausgesucht.

**npm:** Paketmanager für Node.js/JavaScript/Svelte, welcher in diesem Projekt genutzt wurde (npm-Inc, 2025a). Die npm Inc. erlaubt eine freie Nutzung im Rahmen der Nutzungsbedingungen (npm-Inc, 2025b).

**pip:** Paketmanager für Python, welcher in diesem Projekt benutzt wurde (Gedam, 2025). Die Python Software Foundation erlaubt eine freie Nutzung im Rahmen der Nutzungsbedingungen (Durbin, 2025).

**Git:** Software zur Versionskontrolle, die in diesem Projekt verwendet wurde (Torvalds, 2025).

**GitHub:** Plattform zum Hochladen und Teilen von Software. Sie bietet Möglichkeiten zum Kollaborieren und verwendet Git zur Versionskontrolle. GitHub erlaubt es Nutzer:innen Code unter eigenem Copyright und Lizenz zu veröffentlichen (Terms of Service Github-Inc, 2025).

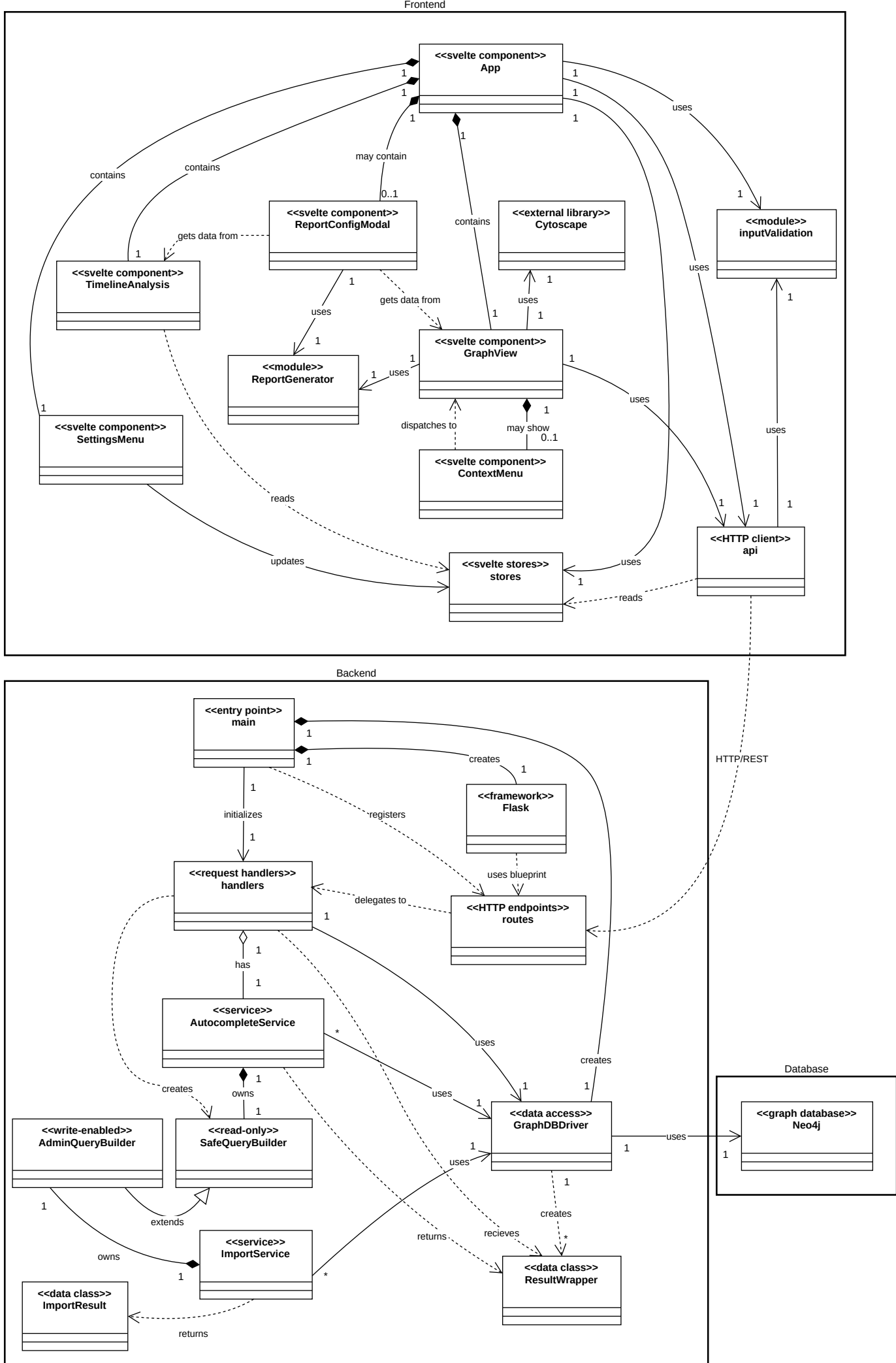
**pylint:** Linter für Python-code (Sassoulas, 2025). Dies Werkzeug wird eingesetzt für statische Codeanalysen des Backends. Lizenz: GPLv2 (Sassoulas, 2021)

**NeoVim:** Texteditor, der individuell konfiguriert und erweitert werden kann. Dieser wurde zum schreiben des Quellcodes verwendet. Lizenz: Apache 2.0 (zeertzjq et al., 2025).

## 4.2 Architekturübersicht

Das hier in Abbildung 4.1 dargestellte Klassendiagramm zeigt die vollständige Systemarchitektur der Threat Intelligence Anwendung, die als klassische Drei-Schichten-Architektur aufgebaut ist: Frontend (Präsentations-Schicht), Backend (Logik-Schicht) und Graphendatenbank (Persistenzschicht). Die Kommunikation zwischen den Schichten erfolgt über definierte Schnittstellen. Die Client-Server-Architektur wurde gewählt, da die Plattform als Webanwendung bereit gestellt wird.

Abbildung 4.1: Klassendiagramm Systemüberblick (draw.io)



## Frontend-Schicht

Das Frontend basiert auf dem Svelte-Framework und ist komponentenbasiert strukturiert. Die **App**-Komponente bildet den zentralen Einstiegspunkt und orchestriert alle untergeordneten UI-Komponenten. Sie besitzt (*Composition*) die Hauptkomponenten **GraphView** für die Graph-Visualisierung, **TimelineAnalysis** für die zeitliche Darstellung von Bedrohungen, sowie **SettingsMenu** und **ReportConfigModal** für Konfiguration und Reporting. Die **api**-Klasse fungiert als HTTP-Client und stellt die einzige Verbindung zum Backend dar. Sie kommuniziert über HTTP/REST mit der Backend-API und verwendet **inputValidation** zur Eingabesani-tisierung vor dem Versenden von Requests.

Für die zentrale Zustandsverwaltung nutzt die Anwendung **stores** (Svelte Stores), die reaktive State-Variablen wie Kontexttiefe, Zeitfilter und Label-Filter verwalten. Diese werden automatisch im Speicher des Browsers gespeichert.

## Backend-Schicht

Das Backend folgt einer hierarchischen Schichtenarchitektur. In ähnlichen Plattformen zur Visualisierung von Threat Intelligence Daten, beispielsweise KAVAS (Böhm et al., 2018, Seite 9-10) wurde eine MVC-Architektur (Gamma et al., 1994, Kapitel 1.2) gewählt. In diesem System ist kein *Model* nach MVC implementiert, stattdessen wird das *data-mapper pattern* (Fowler, 2013, Seite 165) verwendet. Die Daten bleiben als Graph-Struktur in *Neo4j* und werden bei Bedarf als Dictionaries an die Anwendungsschicht übertragen. Dies ermöglicht es die native Graph-Struktur direkt zu nutzen. Das Schichtenmodell bietet erhöhte Performance für Kommunikation und Datenbankoperationen (Tu, 2023, Seite 37).

Das Backend ist in sechs logische Schichten unterteilt:

- 1. API-Schicht (**routes**, **Flask**) - HTTP-Routing
- 2. Anwendungsschicht (**handlers**) - Request-Handling und Business Logik
- 3. Service-Schicht (**AutocompleteService**, **ImportService**) - Domain-Logik
- 4. Query-Schicht (**SafeQueryBuilder**, **AdminQueryBuilder**) - Query-Konstruktion
- 5. Datenzugriffsschicht (**GraphDBDriver**) - Datenbankzugriff
- 6. Datenbankschicht (**Neo4j**) - Graph-Persistierung

Die Architektur nutzt das Dependency Injection Pattern (Seemann, 2019, Kapitel 1, 1.5 und folgende) zur losen Kopplung: Das **main**-Modul erstellt die zentrale **GraphDBDriver**-Instanz und injiziert diese in **handlers**, welches wiederum Services instanziiert und orchestriert.

Das **routes**-Modul definiert alle HTTP-Endpoints (z.B. `‘/api/autocomplete‘`, `‘/api/node/name‘`) und delegiert eingehende Requests an das **handlers**-Modul. Das **handlers**-Modul koordiniert die eigentliche Geschäftslogik. Es erhält den **GraphDBDriver** und Services per Dependency Injection (*Aggregation*) und verwaltet deren Zusammenspiel. **handlers** erstellt bei Bedarf temporär **SafeQueryBuilder**-Instanzen zur sicheren Query-Konstruktion und führt die Queries über den **GraphDBDriver** aus. **AutocompleteService** bietet Autovervollständigung und

Suchfunktionen. Der **ImportService** ermöglicht den Import von Threat Intelligence Daten aus JSON-Dateien. Beide Services teilen sich die gleiche **GraphDBDriver**-Instanz (**Association** mit Multiplizität \*:1) und besitzen jeweils ihren eigenen Query Builder. Der **SafeQueryBuilder** stellt ausschließlich Read-Only Operationen bereit und verhindert durch Validierung Cypher-Injection-Angriffe. Der **AdminQueryBuilder** erweitert (**Inheritance**) den **SafeQueryBuilder** um Schreiboperationen (MERGE, DELETE) und wird ausschließlich für Datenimporte verwendet. Der **GraphDBDriver** kapselt die gesamte Neo4j-Kommunikation und stellt eine einheitliche Schnittstelle für Datenbankoperationen bereit. Er erstellt **ResultWrapper**-Objekte, die Erfolg/Fehler und Daten kapseln.

### Datenbank-Schicht

Die *Neo4j* Graph-Datenbank speichert alle Threat Intelligence Daten. Sie wird ausschließlich über den **GraphDBDriver** mittels *Cypher*-Queries und dem *Bolt*-Protokoll angesprochen.

## 4.3 Struktur

### Frontend

Abbildung 4.2 zeigt die komponentenbasierte Svelte-Architektur.

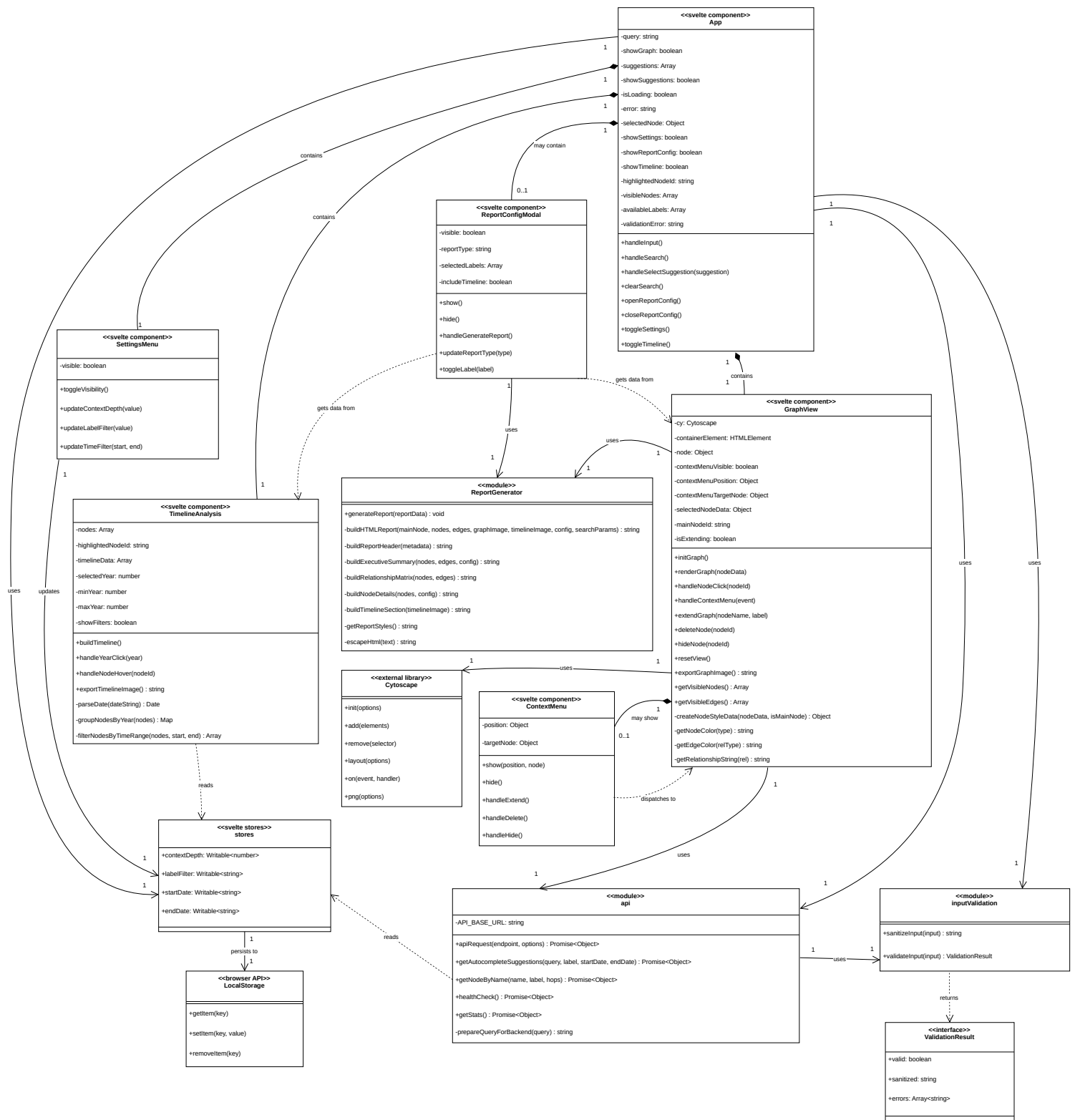


Abbildung 4.2: Klassendiagramm Frontend (draw.io)

## App - Hauptkomponente

`App.svelte` ist der zentrale Einstiegspunkt und Container der gesamten Benutzeroberfläche. Sie besitzt (*Composition*) alle UI-Komponenten und verwaltet den globalen „Application-State“  
Zentrale State-Variablen:

1. `query`: Suchanfrage
2. `showGraph`, `showTimeline`, `showSettings`, `showReportConfig`: UI-Sichtbarkeits-Flags
3. `selectedNode`: Aktuell ausgewählter Knoten
4. `suggestions`: Autocomplete-Vorschläge
5. `isLoading`, `error`: Request-Status

Hauptfunktionen:

1. `handleInput()`: Entprellter Trigger für Autocomplete Funktionen.
2. `handleSearch()`: Führt Suche aus und zeigt Ergebnisse.
3. `handleSelectSuggestion()`: Verarbeitet Autocomplete-Auswahl.
4. `toggleSettings()`, `toggleTimeline()`: UI-Panel Management.
5. `openReportConfig()`: Öffnet den Dialog für die Generierung des Reports.

`App.svelte` koordiniert die Kommunikation zwischen Komponenten und delegiert Daten an Unterkomponenten durch *Props*.

`App.svelte` ist der Einstiegspunkt und Benutzerschnittstelle. Sie stellt Buttons für Suche, Einstellungen, Zeitstrahl-Analyse und Berichtsgenerierung bereit. Die Suche nach Entitäten wird direkt durch `App.svelte` geleistet. Die Requests werden durch das Modul `api.js` in *JSON* an das Backend gesendet.

## Visualisierung über GraphView

`GraphView.svelte` ist verantwortlich für die interaktive Graph-Darstellung mittels `Cytoscape.js`. Sie besitzt (*Composition*) optional ein Kontext-Menü für Rechtsklick-Interaktionen.

Kernattribute:

1. `cy`: Cytoscape-Instanz
2. `node`: Aktuell angezeigte Knoten-Daten
3. `mainNodeId`: ID des Hauptknotens (vor Löschen geschützt)
4. `isExtending`: Loading-State für Graph-Erweiterung

Hauptfunktionen:

1. `initGraph()`: Initialisiert Cytoscape mit Konfiguration.

1. `renderGraph()`: Rendert Nodes und Edges aus API-Dateien.
2. `extendGraph()`: Lädt verbundene Knoten eines Knoten nach.
3. `deleteNode()` / `hideNode()`: Node-Management.
4. `exportGraphImage()`: Exportiert Graph als Base64-PNG für Reports.
5. `getVisibleNodes()` / `getVisibleEdges()`: Extrahiert aktuelle Sichtbarkeit.

Private Helfer-Methoden wie `createNodeStyleData()`, `getNodeColor()`, `getEdgeColor()` kapseln die Style-Logik und lesen CSS-Variablen für konsistente Farbgebung.

### Zeitstrahlvisualisierung durch **TimelineAnalysis**

Die `TimelineAnalysis.svelte` visualisiert Bedrohungsdaten auf einer Zeitachse. Sie liest reaktiv aus `stores.js` (*Dependency*) und synchronisiert mit `GraphView.svelte` für gegenseitiges hervorheben. Hauptattribute:

1. `nodes`: Zu visualisierende Knoten über Props von `App.svelte`.
2. `highlightedNodeId`: Aktuell hervorgehobener Knoten.
3. `timelineData`: Aufbereitete Timeline-Daten gruppiert nach Jahr .
4. `selectedYear`: Aktuell ausgewähltes Jahr für Filterung.

Kernfunktionen:

1. `buildTimeline()`: Gruppiert Nodes nach Zeitstempeln.
2. `handleYearClick()`: Filtert und hebt Nodes eines Jahres hervor.
3. `handleNodeHover()`: Dispatcht Event für GraphView-Highlighting.
4. `exportTimelineImage()`: Exportiert Timeline als Bild für Reports.

Private Methoden wie `parseDate()`, `groupNodesByYear()` und `filterNodesByTimeRange()` behandeln die Datenaufbereitung.

### ContextMenu

`ContextMenu.svelte` ist ein kleines Overlay für Rechtsklick-Aktionen auf Graph-Knoten. Es wird von `GraphView` besessen und dispatcht Events zurück (*Dependency*). Funktionen:

1. `show()`: Zeigt Menü am Knoten.
2. `hide()`: Versteckt Menü.
3. `handleExtend()`: Triggert Graph-Erweiterung.
4. `handleDelete()` / `handleHide()`: Node-Aktionen zum Entfernen.

### SettingsMenu

In `SettingsMenu.svelte` befinden sich UI-Kontrollen für Anwendungseinstellungen. Es aktualisiert (*Association*) direkt die `stores`. Konfigurierbare Parameter:

1. Kontext-Tiefe: 0-3 Hops.

2. Label-Filter: Threat Actor, Malware, Tool, usw.
3. Zeit-Filter: Start- und/oder Enddatum

Änderungen werden sofort in `stores` geschrieben und propagieren reaktiv zu allen abhängigen Komponenten.

## Report Konfiguration

Die Komponente `ReportConfigModal.svelte` ist ein Dialog für Report-Konfiguration. Sie sammelt Einstellungen, holt Daten von `GraphView` und `TimelineAnalysis` (*Dependency*) und generiert den Report via `ReportGenerator`. Kernattribute:

1. `reportType`: Typ des Reports (Executive Summary, Analyst Report, Full Investigation).
2. `selectedLabels`: Zu inkludierende Node-Typen.
3. `includeTimeline`: Soll Timeline-Grafik eingefügt werden?

Die Methode `handleGenerateReport()` arrangiert:

1. Datensammlung von `GraphView` und `TimelineAnalysis`,
2. Report-Generierung via `ReportGenerator` und
3. Öffnen des Reports in neuem Tab.

## API und State Management

### HTTP Client

Das `api.js`-Modul ist der einzige Kommunikationskanal zum Backend. Es kapselt alle HTTP-Requests und verwendet `inputValidation` zur Eingabesanisierung. Hauptfunktionen:

1. `getAutocompleteSuggestions()`: Holt Autocomplete-Vorschläge mit optionalen Filtern
2. `getNodeByName()`: Lädt vollständige Node-Details mit Beziehungen
3. `healthCheck()`: Überprüft Backend-Verfügbarkeit
4. `getStats()`: Holt Datenbankstatistiken

Die private Methode `apiRequest()` ist ein Fetch-Wrapper mit standardisiertem Error-Handling. Die Funktion `getNodeByName()` liest die `contextDepth` aus `stores` (*Dependency*), um die Hop-Tiefe zu bestimmen.

### State Management

Das `stores.js`-Modul definiert Svelte Stores für Frontend-weiten State:

1. `contextDepth`: `Writable<number>` - Graph-Tiefe (Hops)
2. `labelFilter`: `Writable<string>` - Aktueller Label-Filter
3. `startDate, endDate`: `Writable<string>` - Zeitfilter

Jeder Store abonniert sich automatisch für `LocalStorage`-Persistierung (*Association* zu `LocalStorage`). Bei Store-Updates wird der Wert automatisch in `localStorage` geschrieben.

Beim Laden der App werden gespeicherte Werte wiederhergestellt. Die Stores verwenden Svelte's reaktives System: Komponenten die Stores lesen, werden automatisch bei Änderungen neu gerendert.

## Utility Module

### Report Generator

Das Modul `ReportGenerator.js` ist ein rein funktionales Modul ohne State. Es transformiert Graph- und Timeline-Daten in ein vollständiges HTML-Dokument. Hauptfunktion:

1. `generateReport()`: Entry-Point, öffnet Report in neuem Tab

Private Builder-Funktionen:

1. `buildHTMLReport()`: Komplettes HTML-Dokument.
2. `buildReportHeader()`: Titel, Datum, Metadaten.
3. `buildExecutiveSummary()`: High-level Zusammenfassung
4. `buildRelationshipMatrix()`: Beziehungs-Tabelle
5. `buildNodeDetails()`: Detaillierte Node-Informationen
6. `buildTimelineSection()`: Timeline-Grafik einbetten
7. `getReportStyles()`: CSS-Styles für Druckbarkeit

Reports sind self-contained HTML-Dateien mit embedded Base64-Bildern und inline CSS.

### inputValidation

Das `inputValidation.js`-Modul sanitisiert Benutzereingaben vor API-Requests zur Verhinderung von Injection-Angriffen. Funktionen:

1. `sanitizeInput()`: Unicode-Normalisierung (NFKC), Trimming, Lowercase.
2. `validateInput()`: Validierung, gibt `ValidationResult` zurück.

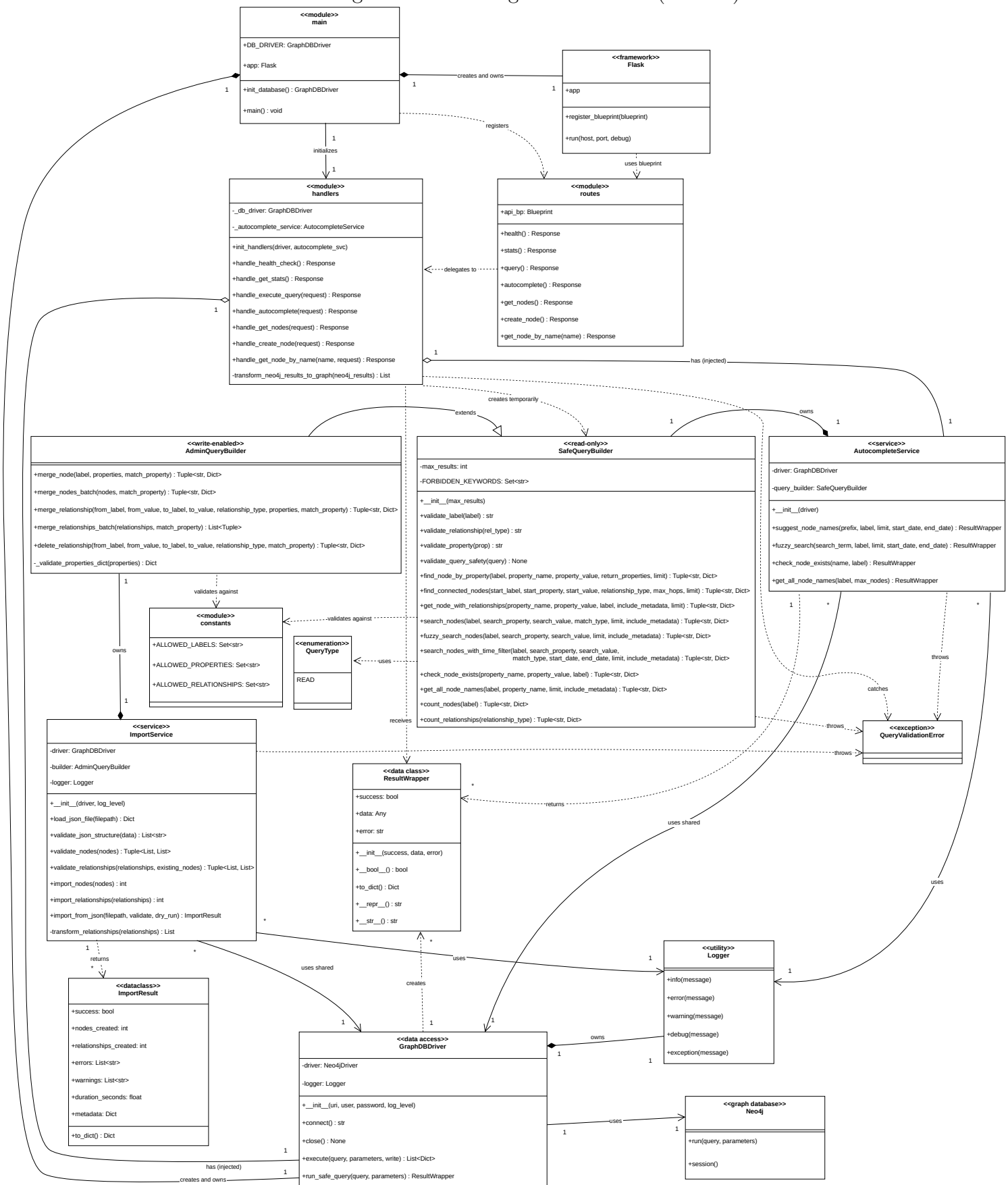
`ValidationResult` enthält:

1. `valid`: Boolean
2. `sanitized`: Bereinigte Eingabe
3. `errors`: Liste von Validierungsfehlern.

### Backend

Abbildung 4.3 zeigt die Python-basierte Server-Architektur mit allen relevanten Klassen, deren Methoden, Attributen und Beziehungen. Die Architektur folgt dem Schichtenprinzip mit Trennung der Verantwortlichkeiten.

Abbildung 4.3: Klassendiagramm Backend(draw.io)



## Einstiegspunkt main

Das `main`-Modul ist der Einstiegspunkt der Anwendung. Es erstellt und besitzt (*Composition*) die `Flask`-Instanz und den `GraphDBDriver`. Die `init_database()`-Methode etabliert die Datenbankverbindung unter Verwendung von Umgebungsvariablen (`NEO4J_URI`, `NEO4J_USER`, `NEO4J_PASSWORD`) und testet die Verbindung. Die `main()`-Funktion organisiert den Startup-Prozess: Datenbankinitialisierung, Handler-Setup und Server-Start.

## API-Schicht

### URL-Routing

Das `routes`-Modul definiert den `Flask Blueprint` `api_bp` mit allen HTTP-Endpoints. Jede Route-Funktion (z.B. `health()`, `autocomplete()`, `get_node_by_name()`) empfängt HTTP-Requests, extrahiert Parameter und delegiert die Verarbeitung an die entsprechenden Handler-Funktionen.

## Anwendungsschicht

### Request-Handling

Das `handlers`-Modul koordiniert die gesamte Request-Verarbeitung und Business-Logik. Es speichert Referenzen zu `_db_driver` und `_autocomplete_service` als private Modul-Variablen. Die `init_handlers()`-Funktion implementiert Dependency Injection für Services. Die Handler-Funktionen folgen einem einheitlichen Pattern:

1. Input-Validierung
2. Service-Koordination (`SafeQueryBuilder` für Query-Bau, `GraphDBDriver` für Ausführung)
3. Fehlerbehandlung (`QueryValidationError`, `Neo4jError`)
4. Response-Formatierung (JSON mit success/error Status)

Die private Methode `transform_neo4j_results_to_graph()` transformiert rohe Neo4j-Ergebnisse in das Frontend-kompatible Graph-Format (*Cytoscape*)).

## Service-Schicht

### AutocompleteService

Der `AutocompleteService` kapselt alle Such- und Autovervollständigungsfunktionen. Er besitzt (*Composition*) einen eigenen `SafeQueryBuilder` und verwendet (*Association*) den gemeinsamen `GraphDBDriver`. Die Hauptmethoden sind:

1. `suggest_node_names()`: Prefix-basierte Suche mit optionalen Zeit- und Label-Filtern
2. `fuzzy_search()`: Fuzzy-Matching für flexiblere Suchen
3. `check_node_exists()`: Schnelle Existenzprüfung
4. `get_all_node_names()`: Bulk-Abfrage für Frontend-Caching<sup>5</sup> (nicht implementiert)

Alle Methoden geben `ResultWrapper` zurück und behandeln Exceptions konsistent.

---

<sup>5</sup>Während der Entwicklung wurde hiermit getestet ob Frontend-Caching beim initialen Aufrufen der Seite eine Option für schnelle Bereitstellung der Namen bei der Autocomplete Suche wäre. Es hat aktuell keinen

## ImportService

Der `ImportService` verwaltet den Import von Threat Intelligence Daten aus JSON-Dateien. Er besitzt (*Composition*) einen `AdminQueryBuilder` für Schreiboperationen. Der Service wird ausschließlich am Backend-Server ausgeführt. Damit sind Nutzerinteraktionen und Datenimport strikt getrennt. Der Import-Workflow umfasst:

1. `load_json_file()`: Datei laden und parsen
2. `validate_json_structure()`: Struktur-Validierung
3. `validate_nodes()` / `validate_relationships()`: Inhaltliche Validierung gegen Whitelists
4. `import_nodes()` / `import_relationships()`: Batch-Import in Neo4j
5. Rückgabe eines `ImportResult` mit Statistiken und Fehlern

Die `iimport_from_json()`-Methode verwaltet den gesamten Prozess und unterstützt einen Dry-Run-Modus.

## ImportResult

Die `ImportResult`-Dataclass kapselt alle Informationen eines Import-Vorgangs: Erfolgs-Status, Anzahl erstellter Nodes/Relationships, Fehler, Warnungen, Dauer und Metadaten. Die `to_dict()`-Methode ermöglicht einfache JSON-Serialisierung.

## Query-Schicht

### SafeQueryBuilder

Der `SafeQueryBuilder` ist die zentrale Sicherheitskomponente. Er konstruiert ausschließlich Read-Only Cypher-Queries und verhindert durch mehrere Mechanismen Injection-Angriffe:

1. `validate_label()`, `validate_relationship()`, `validate_property()`: Whitelist-basierte Validierung gegen constants,
2. `validate_query_safety()`: Überprüft auf verbotene Keywords (DELETE, CREATE, etc.),
3. Alle Queries sind parametrisiert (nie String Concatenation).

Wichtige Query-Builder-Methoden:

1. `find_node_by_property()`: Sucht Nodes anhand eines Property-Wertes.
2. `find_connected_nodes()`: Findet verbundene Nodes mit konfigurierbarer Hop-Tiefe.
3. `search_nodes_with_time_filter()`: Suche mit Zeitfilterung.
4. `count_nodes()` / `count_relationships()`: Zählt Nodes und Relationships.

Alle Methoden geben ein Tuple (`query_string`, `parameters_dict`) zurück.

### AdminQueryBuilder

---

Vorteil, da die Suche sowie schon sehr schnell ist. Daher wurde es nicht umgesetzt.

Der AdminQueryBuilder erweitert (*Inheritance*) den SafeQueryBuilder um Schreiboperationen. Er ist ausschließlich für Datenimporte vorgesehen und nie direkt über die API erreichbar. Zusätzliche Methoden:

1. `merge_node()` / `merge_nodes_batch()`: Node-Erstellung/Update
2. `merge_relationship()` / `merge_relationships_batch()`: Relationship-Management
3. `delete_relationship()` / `delete_node()`: Löschen

Die Batch-Methoden optimieren Performance durch Gruppierung gleichartiger Operationen.

### **QueryValidationError**

Eine Exception-Klasse für Validierungsfehler. Wird geworfen, wenn ungültige Labels, Properties oder Relationships verwendet werden.

### **Datenzugriffsschicht**

#### **GraphDBDriver**

Der GraphDBDriver kapselt die gesamte Neo4j-Kommunikation und ist als **Singleton** implementiert. Er verwaltet den Neo4j-Driver und einen eigenen Logger (Composition). Kernmethoden:

1. `close()`: Sauberes Schließen der Verbindung
2. `execute()`: Führt Cypher-Query aus, wirft Exceptions bei Fehler
3. `run_safe_query()`: Wrapper um `execute()`, fängt Exceptions ab und gibt `ResultWrapper` zurück

Die Methode `execute()` verwendet Neo4j-Transactions und unterscheidet zwischen Read/Write-Operationen.

### **ResultWrapper**

Der ResultWrapper ist eine Datenklasse für konsistente Ergebnisbehandlung. Sie kapselt:

1. `success`: Boolean für Erfolgs-Status
2. `data`: Beliebige Ergebnisdaten
3. `error`: Fehlermeldung bei Misserfolg

Die `__bool__()`-Methode ermöglicht einfache Erfolgs-Checks: `if result: ...`

### **Utilities**

#### **Logger**

Utility-Klasse für strukturiertes Logging mit verschiedenen Log-Levels (`info`, `error`, `warning`, `debug`, `exception`).

#### **constants**

Modul mit Whitelists für erlaubte Labels, Properties und Relationships. Diese Konstanten werden von allen Query Buildern zur Validierung verwendet.

## 4.4 Verhalten

Ein typischer Ablauf des Systems ist das Suchen und Anzeigen eines Knotens. Der Ablauf ist in den folgenden Sequenzdiagrammen dargestellt. Abbildung 4.4 zeigt die Such- und Autocomplete-Funktion. Abbildung 4.5 zeigt die Auswahl eines Such-Vorschlags und die anschließende Darstellung als Graphen. Da die Abläufe aufgrund der Komplexität sehr klein dargestellt sind, stehen Darstellungen im Querformat in Anlage 5.3 zur Verfügung. Die Originale im draw.io Format liegen in den Dokumentationsunterlagen bei.

Abbildung 4.4: Sequenzdiagramm Autocomplete (draw.io)

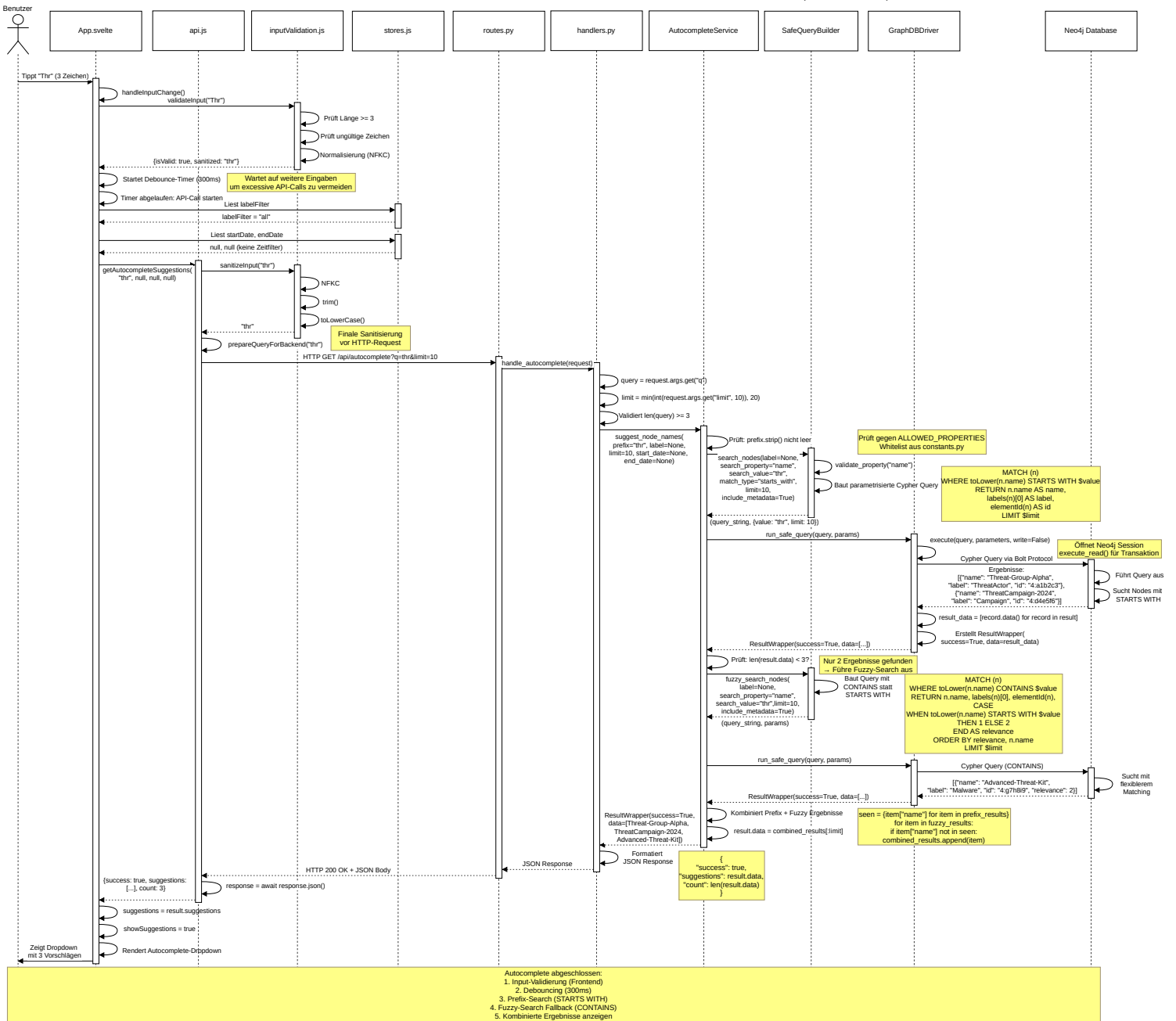
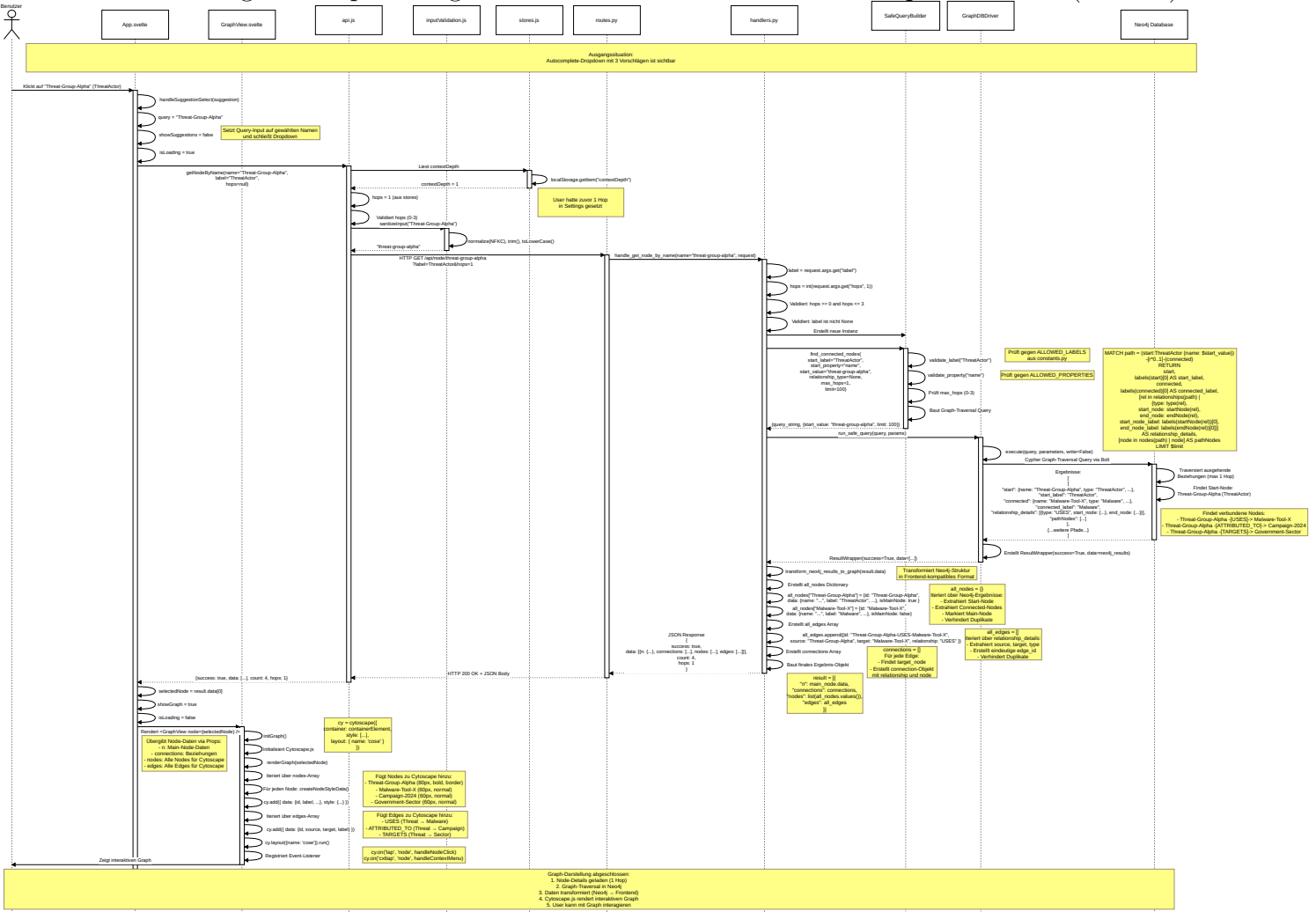


Abbildung 4.5: Sequenzdiagramm Knoten auswählen und in Graph darstellen (draw.io)



# Literatur

- Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for Agile Software Development*. Verfügbar 28. Dezember 2025 unter <https://agilemanifesto.org/>
- Bemis, C. (2025a). Lucide Icons. Verfügbar 9. Dezember 2025 unter <https://lucide.dev>
- Bemis, C. (2025b). Lucide License. Verfügbar 9. Dezember 2025 unter <https://lucide.dev>
- BMI. (2021). *Stakeholderanalyse* [Bundesministerium des Innern und für Heimat]. Verfügbar 29. April 2025 unter [https : / / www . orghandbuch . de / Webs / OHB / DE / OrganisationshandbuchNEU / 4.MethodenUndTechniken / Methoden\\_A\\_bis\\_Z / Stakeholderanalyse/Stakeholderanalyse\\_node.html](https://www.orghandbuch.de/Webs/OHB/DE/OrganisationshandbuchNEU/4.MethodenUndTechniken/Methoden_A_bis_Z/Stakeholderanalyse/Stakeholderanalyse_node.html)
- Böhm, F., Menges, F., & Pernul, G. (2018). Graph-based visual analytics for cyber threat intelligence. *Cybersecurity*, 1(1), 16. <https://doi.org/10.1186/s42400-018-0017-4>
- Docker-Inc. (2025a). Docker Engine. Verfügbar 8. Dezember 2025 unter <https://docs.docker.com/engine/>
- Docker-Inc. (2025b). Docker Lizenz. Verfügbar 8. Dezember 2025 unter <https://github.com/moby/moby/blob/master/LICENSE>
- Dolphin, C. (2025a). flask-cors. Verfügbar 7. Dezember 2025 unter <https://corydolphin.com/flask-cors/>
- Dolphin, C. (2025b). flask-cors/LICENSE at main · corydolphin/flask-cors. Verfügbar 7. Dezember 2025 unter <https://github.com/corydolphin/flask-cors/blob/main/LICENSE>
- Downie, N. (2025). Chart.js. Verfügbar 9. Dezember 2025 unter <https://github.com/chartjs>
- Durbin, E. (2025). Terms of Service - Python Software Foundation Policies. Verfügbar 9. Dezember 2025 unter <https://policies.python.org/pypi.org/Terms-of-Service/>
- Excel — microsoft 365*. (2025). Verfügbar 22. August 2025 unter <https://www.microsoft.com/en/microsoft-365/excel>
- Fowler, M. (2013). *Patterns of enterprise application architecture* (Nineteenth printing). Addison-Wesley.
- Franz, M. (2024). cytoscape.js/LICENSE at unstable · cytoscape/cytoscape.js. Verfügbar 9. Dezember 2025 unter [https : / / github . com / cytoscape / cytoscape . js / blob / unstable / LICENSE](https://github.com/cytoscape/cytoscape.js/blob/unstable/LICENSE)

- Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., & Bader, G. D. (2016). Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2), 309–311. <https://doi.org/10.1093/bioinformatics/btv557>
- Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software* [OCLC: 624525693]. Addison-Wesley.
- Gedam, P. (2025, Dezember). pypa/pip [original-date: 2011-03-06T14:30:46Z]. Verfügbar 9. Dezember 2025 unter <https://github.com/pypa/pip>
- Github-Inc. (2025). GitHub Terms of Service. Verfügbar 9. Dezember 2025 unter <https://docs-internal.github.com/en/site-policy/github-terms/github-terms-of-service>
- Göpel, D. (2025a, Dezember). sveltejs/vite-plugin-svelte [original-date: 2021-03-16T20:54:27Z]. Verfügbar 9. Dezember 2025 unter <https://github.com/sveltejs/vite-plugin-svelte>
- Göpel, D. (2025b). vite-plugin-svelte/LICENSE at main · sveltejs/vite-plugin-svelte. Verfügbar 9. Dezember 2025 unter <https://github.com/sveltejs/vite-plugin-svelte/blob/main/LICENSE>
- Khan, Q. A. (2016, Februar). Making sense of graph databases part 4 - Why are graph databases more efficient than RDMS on connected data. Verfügbar 8. Dezember 2025 unter <http://qtips.github.io/2016/02/part4-rdms-vs-graph>
- Krekel, H. (2015a). License - pytest documentation. Verfügbar 7. Dezember 2025 unter <https://docs.pytest.org/en/stable/license.html>
- Krekel, H. (2015b). pytest documentation. Verfügbar 7. Dezember 2025 unter <https://docs.pytest.org/en/stable/index.html>
- Ltd, J. (2025). *draw.io* [draw.io]. <https://www.drawio.com/>
- MITRE. (2025). *MITRE ATT&CK*®. Verfügbar 9. Juni 2025 unter <https://attack.mitre.org/>
- Mozilla-Foundation. (2025, November). The web standards model - Learn web development — MDN. Verfügbar 9. Dezember 2025 unter [https://developer.mozilla.org/en-US/docs/Learn\\_web\\_development/Getting\\_started/Web\\_standards/The\\_web\\_standards\\_model](https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model)
- Neo4j-Inc. (2025a). Community Edition. Verfügbar 8. Dezember 2025 unter <https://neo4j.com/product/community-edition/>
- Neo4j-Inc. (2025b). *Create a graph data model* [Neo4j graph data platform]. Verfügbar 28. Dezember 2025 unter <https://neo4j.com/docs/getting-started/data-modeling/tutorial-data-modeling/>
- Neo4j-Inc. (2025c, Juni). Neo4j Python Driver 6.0 — Neo4j Python Driver 6.0. Verfügbar 7. Dezember 2025 unter <https://neo4j.com/docs/api/python-driver/current/>
- Neo4j-Inc. (2025d, November). Repository Neo4j Python driver and License. Verfügbar 7. Dezember 2025 unter <https://pypi.org/project/neo4j/>
- npm-Inc. (2025a). npm — Home. Verfügbar 9. Dezember 2025 unter <https://www.npmjs.com/>
- npm-Inc. (2025b). Terms and Licenses — npm Docs. Verfügbar 9. Dezember 2025 unter <https://docs.npmjs.com/policies/terms>
- OMG, O. M. G. (2017, Dezember). Unified Modeling Language, v2.5.1. <https://www.omg.org/spec/UML/>

- Project-Management-Institute. (2019). *Practice Standard for Work Breakdown Structures – Third Edition* (3. Aufl.). Global Standards.
- Python-Software-Foundation. (2025a, Juni). Python License. Verfügbar 7. Dezember 2025 unter <https://docs.python.org/3/license.html>
- Python-Software-Foundation. (2025b, Juli). The Python Standard Library. Verfügbar 7. Dezember 2025 unter <https://docs.python.org/3/library/index.html>
- Ronacher, A. (2007). BSD-3-Clause License — Werkzeug Documentation (3.1.x). Verfügbar 8. Dezember 2025 unter <https://werkzeug.palletsprojects.com/en/stable/license/>
- Ronacher, A. (2010). BSD-3-Clause License — Flask Documentation (3.1.x). Verfügbar 7. Dezember 2025 unter <https://flask.palletsprojects.com/en/stable/license/>
- Ronacher, A. (2025a). HTTP Exceptions — Werkzeug Documentation (3.1.x). Verfügbar 7. Dezember 2025 unter <https://werkzeug.palletsprojects.com/en/stable/exceptions/#error-classes>
- Ronacher, A. (2025b, August). Welcome to Flask - Flask Documentation (3.1.x). Verfügbar 7. Dezember 2025 unter <https://flask.palletsprojects.com/en/stable/>
- Sassoulas, P. (2021). pylint/LICENSE at main · pylint-dev/pylint. Verfügbar 11. Dezember 2025 unter <https://github.com/pylint-dev/pylint/blob/main/LICENSE>
- Sassoulas, P. (2025). pylint-dev/pylint: It's not just a linter that annoys you! Verfügbar 11. Dezember 2025 unter <https://github.com/pylint-dev/pylint>
- Seemann, M. (2019). *Dependency Injection Principles, Practices, and Patterns*. Manning Publications Co. LLC.
- Sheremet, V. (2025). vitest/LICENSE at main · vitest-dev/vitest. Verfügbar 9. Dezember 2025 unter <https://github.com/vitest-dev/vitest/blob/main/LICENSE>
- Sheremet, V., Fu, A., Ogawa, H., & Patak, J. S. (2025). Vitest. Verfügbar 9. Dezember 2025 unter <https://vitest.dev/>
- Struse, R., & Darley, T. (2019, 26. Juli). STIX Version 2.1. Verfügbar 21. Mai 2025 unter [https://docs.oasis-open.org/cti/stix/v2.1/csprd01/stix-v2.1-csprd01.html#\\_Toc16070564](https://docs.oasis-open.org/cti/stix/v2.1/csprd01/stix-v2.1-csprd01.html#_Toc16070564)
- Svelte. (2025a). *Svelte • web development for the rest of us*. Verfügbar 5. Juli 2025 unter <https://svelte.dev/>
- Svelte. (2025b). sveltejs/svelte [original-date: 2016-11-20T18:13:05Z]. Verfügbar 8. Dezember 2025 unter <https://github.com/sveltejs/svelte>
- Swayne, L. (2024). Chart.js/LICENSE.md at master · chartjs/Chart.js. Verfügbar 9. Dezember 2025 unter <https://github.com/chartjs/Chart.js/blob/master/LICENSE.md>
- Torvalds, L. (2025). Git. Verfügbar 9. Dezember 2025 unter <https://git-scm.com/about>
- Tu, Z. (2023). Research on the Application of Layered Architecture in Computer Software Development. *Journal of Computing and Electronic Information Management*, 11(3), 34–38. <https://doi.org/10.54097/jceim.v11i3.08>
- World-Wide-Web-Consortium. (2025). Web Standards. Verfügbar 9. Dezember 2025 unter <https://www.w3.org/standards/>

- Wright, A., & Andrews, H. (2018, 19. März). *JSON Schema*. Verfügbar 29. Dezember 2025 unter <https://json-schema.org/draft-07>
- You, E. (2025a). Vite. Verfügbar 9. Dezember 2025 unter <https://vite.dev>
- You, E. (2025b). vite/LICENSE at main · vitejs/vite. Verfügbar 9. Dezember 2025 unter <https://github.com/vitejs/vite/blob/main/LICENSE>
- zeertzjq, Keyes, J. M., & Laso, J. E. (2025, Dezember). neovim/neovim [original-date: 2014-01-31T13:39:22Z]. Verfügbar 9. Dezember 2025 unter <https://github.com/neovim/neovim>

# Abbildungsverzeichnis

1.1	Gantt Chart der Entwicklung. Erstellt mit Python. . . . .	18
1.2	Burn-Down Chart Sprint 1 - Grundstruktur und Datenbankintegration (Python)	21
1.3	Burn-Down Chart Sprint 2- Backend-Kommunikation und Requests (Python) . .	24
1.4	Burn-Down Chart Sprint 3 - Erweiterte Features und Finalisierung (Python) . .	27
2.1	Stakeholdermatrix basierend auf Einfluss und Interesse am Projekt. . . . .	29
2.2	UML-Use-Case-Diagramm erstellt mit draw.io . . . . .	35
2.3	Analyse des Systemkontext, erstellt mit draw.io . . . . .	36
3.1	Datenmodell der Container-Klasse Observable und die Klassen der Artefakte. . .	41
3.2	Übersicht über die Assoziationsklassen(draw.io) . . . . .	42
3.3	UML Klassendiagramm des Datenmodells (draw.io) . . . . .	43
3.4	Datenmodell als Graphenmodell erstellt mit Neo4j . . . . .	44
3.5	UML-Aktivitätsdiagramm für den zentralen Prozess GP-01, Suche und Visualisierung (draw.io) . . . . .	45
3.6	Mockup des ersten GUI Zusatandes . . . . .	48
3.7	Mockup des zweiten GUI Zusatandes . . . . .	49
3.8	Skizze der Zeitstrahlanalyse . . . . .	49
4.1	Klassendiagramm Systemüberblick (draw.io) . . . . .	55
4.2	Klassendiagramm Frontend (draw.io) . . . . .	58
4.3	Klassendiagramm Backend(draw.io) . . . . .	63
4.4	Sequenzdiagramm Autocomplete (draw.io) . . . . .	68
4.5	Sequenzdiagramm Knoten auswählen und in Graph darstellen (draw.io) . . . . .	69
5.1	Sequenzdiagramm Autocomplete (draw.io) . . . . .	82
5.2	Sequenzdiagramm Knoten auswählen und in Graph darstellen (draw.io) . . . . .	83
5.3	Zertifikat Neo4j Fundamentals erhalten am 14.05.25 . . . . .	84
5.4	Zertifikat Cypher Fundamentals erhalten am 14.05.25 . . . . .	84
5.5	Zertifikat Graph Data Modeling Fundamentals erhalten am 16.05.25 . . . . .	85
5.6	Zertifikat Importing Data Fundamentals erhalten am 18.05.25 . . . . .	85
5.7	Zertifikat Neo4j with Python erhalten am 24.05.25 . . . . .	86

## Tabellenverzeichnis

1.1	Übersicht der identifizierten Projektrisiken (EW = Eintrittswahrscheinlichkeit) .	10
1.2	Zeitlicher Projektplan . . . . .	11
1.4	Projektstrukturplan - gesamtes Projekt - tabellarisch . . . . .	12
1.5	Projektstrukturplan - Entwicklung 1.2 - tabellarisch . . . . .	14
1.6	Backlog Sprint 1 . . . . .	19
1.7	Backlog Sprint 2 . . . . .	22
1.8	Backlog Sprint 3 . . . . .	26
2.1	Story Map geordnet nach Priorität . . . . .	34
2.2	Nicht-funktionale Anforderungen nach ISO/IEC 25010 . . . . .	38

## 5 Anlagen

### 5.1 Stakeholderanalyse

Stakeholderanalyse												
ID:	Stakeholder	Rolle im Projekt	Grad der Betroffenheit [nicht   positiv   negativ]	Art der Betroffenheit [positiv   negativ]	Interesse [hoch   mittel   niedrig]	Einfluss, Macht [hoch   mittel   niedrig]	Erwartungen, Anforderungen, Ziele	Befürchtungen	Einstellung begründen, Ursachen u. Erkenntnisse nennen (Motive)	Steuerungsmaßnahme	Termin, Intervall	Aktueller Status [Analyse, Erkenntnisse, weitere Maßnahmen]
1	Cyber Threat Intelligence Analysten	Nutzer	nicht	positiv	hoch	gering	Auswerten von Bedrohungen, Erstellen von Berichten, Erkennen von Zusammenhängen, Verstehen von Beziehungen	Erwartungen werden nicht erfüllt.	Nur eigene Erfahrungen vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Generierung der Anforderung aus eigener Erfahrung als Cybercrime Analyst
2	Security Operations Center (SOC) Analysten	Nutzer	nicht	positiv	hoch	gering	Nutzen der CTI-Daten zur Bewertung und Bearbeitung von Sicherheitsvorfällen (z.B. Incident Response)	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
3	CISO (Chief Information Security Officer)	Nutzer	nicht	positiv	hoch	gering	Benötigt strategische Berichte, Risikoübersicht, Trendanalysen um Unternehmensentscheidungen auf Basis aktueller Bedrohungslagen treffen zu können	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
4	IT-Sicherheitsmanagement	Nutzer	nicht	positiv	hoch	gering	Nutzen CTI-Daten zur Priorisierung von Maßnahmen wie Patch Management oder Definition von Schutzstrategien	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
5	System-administratoren	Nutzer	nicht	positiv	hoch	gering	Prüfung von IOCs und einlegen in Sicherheitssysteme wie Firewalls und SIEMs.	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
6	Threat Hunters	Nutzer	nicht	positiv	hoch	gering	Verwenden der CTI-Daten aktiv zur proaktiven Suche nach Bedrohungen	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
7	Behörden und Nachrichtendienste	Nutzer	nicht	positiv	hoch	gering	Oft an der Weitergabe oder Austausch von CTI-Daten beteiligt (z.B. CERTs, BSI, Polizeien)	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
8	Externe Dienstleister	Nutzer	nicht	positiv	hoch	gering	CTI-Provider oder MSSPs (Managed Security Service Provider), die entweder selbst Informationen liefern oder die Plattform integrieren.	Erwartungen werden nicht erfüllt.	In dieser Projektphase sind noch keine User vorhanden.	Fokus auf Basis Anforderungen.	Phasenweise	Basis Anforderungen bekannt
9	Datenschutz-beauftragte	Regulierung und Kontrollinstanz	stark	neutral	gering	hoch	Müssen sicherstellen, dass bei der Erfassung und Verarbeitung von Bedrohungsdaten keine Datenschutzgesetze verletzt werden, insbesondere wenn personenbezogene Daten betroffen sind.	Anforderungen an den Datenschutz werden nicht erfüllt.	Planung des Projekts erfolgt als studentisches Experiment zum Erproben von Technologien. Es werden in diesem Projekt kein echten personenbezogenen Daten verwendet.	Keine Verwertung von datenschutzrechtlich relevanten Daten.	Bis zur Zieländerung des Projekts	Keine Verwertung von datenschutzrechtlich relevanten Daten.
10	Compliance Manager	Regulierung und Kontrollinstanz	stark	neutral	gering	hoch	Überprüfen, ob regulatorische Anforderungen (z.B. GDPR, NIS2) eingehalten werden.	Nicht alle Anforderungen sind bekannt.	Da es sich um ein studentisches Experiment handelt, unterliegt das Projekt keinen spezifischen Unternehmens-Compliance-Vorgaben.	Datenschutzrechtliche und sicherheitsrelevante Anforderungen werden in diesem Stadium berücksichtigt, jedoch nicht vollständig umgesetzt.	Bis zur Zieländerung des Projekts	Einholen von Best-Practices, einheitlichen Industriestandards, relevanten Sicherheitsstandards als Orientierung.
11	Entwickler/Team	Technische Betreiber / Entwickler	stark	positiv	hoch	hoch	Entwickelt die Plattform und muss auf die Anforderungen aller anderen Stakeholder eingehen.	Technologische Unsicherheit führt zu einer unsicheren Plattform, die Anforderungen (teilweise) nicht erfüllt.	Die Technologien, welche hier umgesetzt werden sind mir als Solo-Entwickler teilweise noch nicht bekannt. Ich habe keine Erfahrung im Zusammenspiel der verschiedenen Technologien. Einschätzungen bezüglich dem Aufwand sind daher schwer zu schätzen.	Schnelles Feedback durch Prototypen, interaktive Tutorials, Zertifikate und nutzen der Kommunikationswege zum Dozenten.	Phasenweise	Entwicklung eines CRUD-Drivers für Neo4j abgeschlossen. Zertifikate für Neo4 Fundamentals, Cypher, Datenimport, Neo4j-Python erlangt. Svelte (Frontend) Interaktives Tutorial begonnen.
12	Hoster	Technische Betreiber / Entwickler	wenig	positiv	mittel	gering	Sichert die Verfügbarkeit der Plattform.	Zu hohe Kosten, bei geringen Traffic. Lags oder Verfügbarkeitseinschränkung bei zu hohem Traffic.	Für die Umsetzung des Projekts wird auf eine Fremd-Hosting-Lösung verzichtet. Es werden daher keine Analysen zur Netzwerklast und Hosting-Modell durchgeführt.	Plattform wird in einer Form bereitgestellt, die eine Selbsthosten ermöglicht. Front und Backend wird als Monolith organisiert.	Bis zur Zieländerung des Projekts	Bereitstellung nach Wunsch des Dozenten.
13	Dozent	Prüft das Projekt inhaltlich, methodisch und formal. Bewertet die Prüfungsleistung mit einer Note.	wenig	neutral	hoch	hoch	Möchte eine saubere Projektdokumentation, Trennung von einer produktiven Lösung und eines studentischen Experiments zur Erprobung von Technologien, aufgrund von Datenschutzaspekten.	Fehler im Projekt führen zu schlechten Noten. Unklare Erwartungen. Unklare Kommunikation. Schlechte Erreichbarkeit.	Feedback wird nur teilweise oder nur unzureichend umgesetzt. Bei Unklarheiten wird kein Feedback eingeholt. Falsche Annahmen in der Kommunikation führen zu unklaren Erwartungen. Ich befinde mich über den gesamten Projektverlauf in Japan und habe damit eine Zeitverschiebung von +7 Stunden, dies kann zu Problemen in der Kommunikation führen.	Aus dem Feedback werden Backgolelemente generiert, die priorisiert bearbeitet werden. Kommunikation und Feedback wird niederschwellig gesucht. Kommunikation per E-Mail ist aufgrund der Zietverschiebung sinnvoll und zielführend.	Phasenweise	Absprache vor Beginn der Phase I
14	Prüfungsamt IU	Prüft formal ob die Prüfungsleistung erbracht ist. (Abgabe, Bewertung, Korrektheit der Prozesse)	wenig	neutral	mittel	hoch	Möchte das administrative, inhaltliche und formale Anforderungen nach ihren Vorgaben umgesetzt werden.	Administrative Fehler beim Hochladen und der Organisation.	Kein Ausreichendes Verständnis über Pebble Pad. Fehler bei der Bedienung.	Relevante Dokumentation und Voraussetzungen lesen.	Phasenweise	Checkliste zur Prüfung der Vollständigkeit

## 5.2 JSON Schema

### JSON Schema

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "title": "Threat_Intelligence_Data_Import_Schema",
4   "description": "JSON_Schema_for_validating_threat_intelligence_data_
   ↪ import_files",
5   "type": "object",
6   "required": ["metadata", "nodes", "relationships"],
7   "properties": {
8     "metadata": {
9       "type": "object",
10      "required": ["version", "created", "source"],
11      "properties": {
12        "version": {
13          "type": "string",
14          "pattern": "^1\\.0$",
15          "description": "Schema_version, currently 1.0"
16        },
17        "created": {
18          "type": "string",
19          "format": "date-time",
20          "description": "ISO_8601_timestamp_of_creation"
21        },
22        "source": {
23          "type": "string",
24          "minLength": 1,
25          "description": "Data_source_identifier"
26        },
27        "description": {
28          "type": "string",
29          "description": "Human-readable_description"
30        },
31        "node_count": {
32          "type": "integer",
33          "minimum": 0,
34          "description": "Number_of_nodes_in_file"
35        },
36        "relationship_count": {
37          "type": "integer",
38          "minimum": 0,
39          "description": "Number_of_relationships_in_file"
```

```

40     },
41     "author": {
42         "type": "string",
43         "description": "Data creator or organization"
44     },
45     "tags": {
46         "type": "array",
47         "items": {
48             "type": "string"
49         },
50         "description": "Classification tags"
51     }
52 }
53 },
54 "nodes": {
55     "type": "array",
56     "items": {
57         "type": "object",
58         "required": ["label", "properties"],
59         "properties": {
60             "label": {
61                 "type": "string",
62                 "enum": [
63                     "AttackPattern",
64                     "Campaign",
65                     "Identity",
66                     "Incident",
67                     "Indicator",
68                     "Malware",
69                     "Observable",
70                     "File",
71                     "DomainName",
72                     "URL",
73                     "EmailAddr",
74                     "IPv4Adress",
75                     "Organization",
76                     "Report",
77                     "ThreatActor",
78                     "Tool",
79                     "Vulnerability"
80                 ],
81                 "description": "Node label from allowed labels"
82             },

```

```

83         "properties": {
84             "type": "object",
85             "minProperties": 1,
86             "description": "Node_properties"
87         }
88     },
89     },
90     "description": "Array_of_node_objects"
91 },
92 "relationships": {
93     "type": "array",
94     "items": {
95         "type": "object",
96         "required": ["type", "from", "to"],
97         "properties": {
98             "type": {
99                 "type": "string",
100                 "enum": [
101                     "BASED_ON",
102                     "DETECTS",
103                     "DESCRIBES",
104                     "EMPLOYES",
105                     "HAS_IDENTITY",
106                     "INDICATED_BY",
107                     "INVOLVES",
108                     "LAUNCHED",
109                     "RELATED_TO",
110                     "TARGETS",
111                     "USES"
112                 ],
113                 "description": "Relationship_type_from_allowed_types"
114             },
115             "from": {
116                 "type": "object",
117                 "required": ["label", "property", "value"],
118                 "properties": {
119                     "label": {
120                         "type": "string",
121                         "description": "Source_node_label"
122                     },
123                     "property": {
124                         "type": "string",
125                         "description": "Property_to_match_on_(usually_'name')"
```

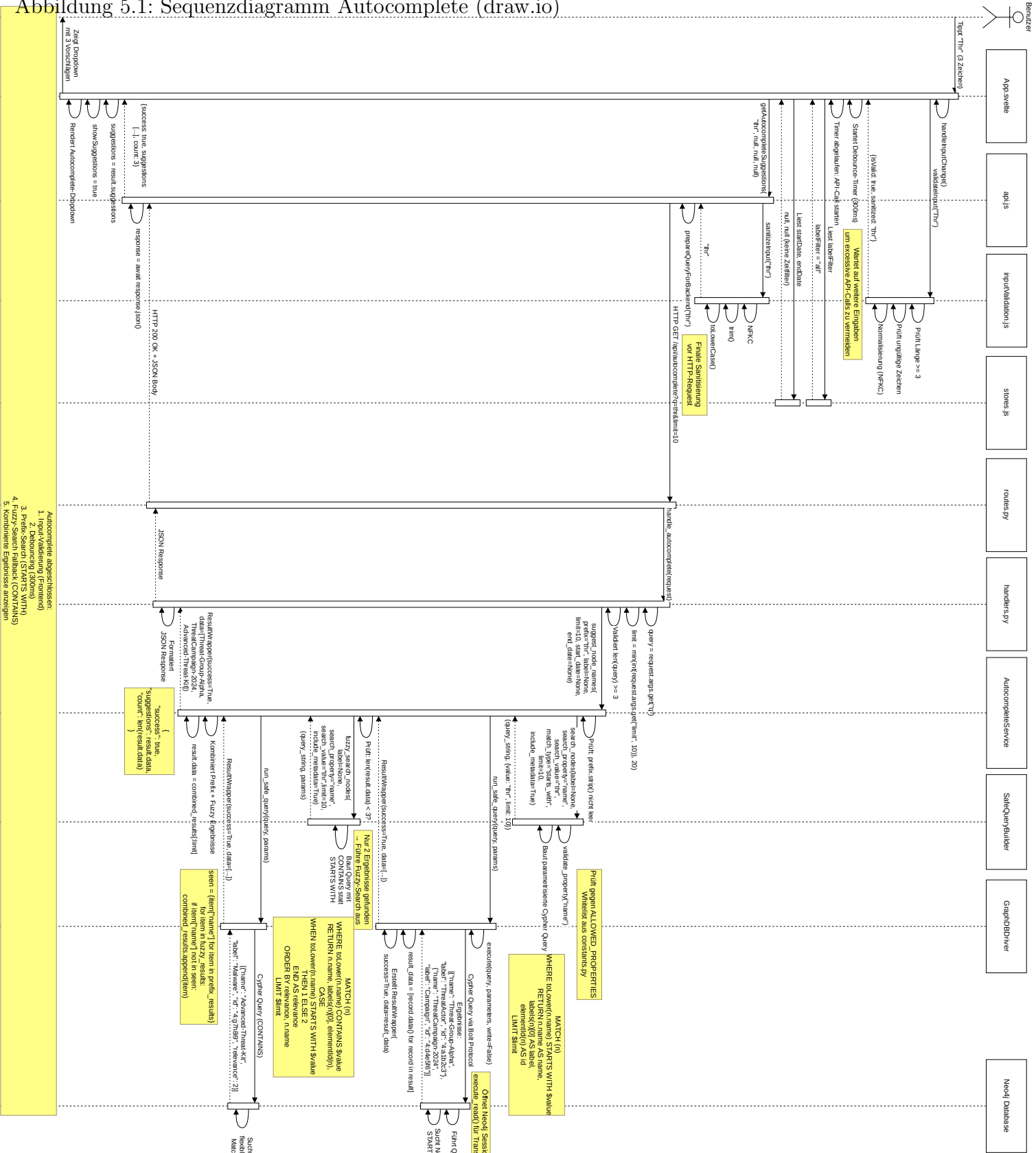
```

126         },
127         "value": {
128             "description": "Value to identify source node"
129         }
130     },
131 },
132 "to": {
133     "type": "object",
134     "required": ["label", "property", "value"],
135     "properties": {
136         "label": {
137             "type": "string",
138             "description": "Target node label"
139         },
140         "property": {
141             "type": "string",
142             "description": "Property to match on (usually 'name')"
143         },
144         "value": {
145             "description": "Value to identify target node"
146         }
147     }
148 },
149 "properties": {
150     "type": "object",
151     "description": "Optional relationship properties"
152 }
153 },
154 },
155 "description": "Array of relationship objects"
156 }
157 }
158 }

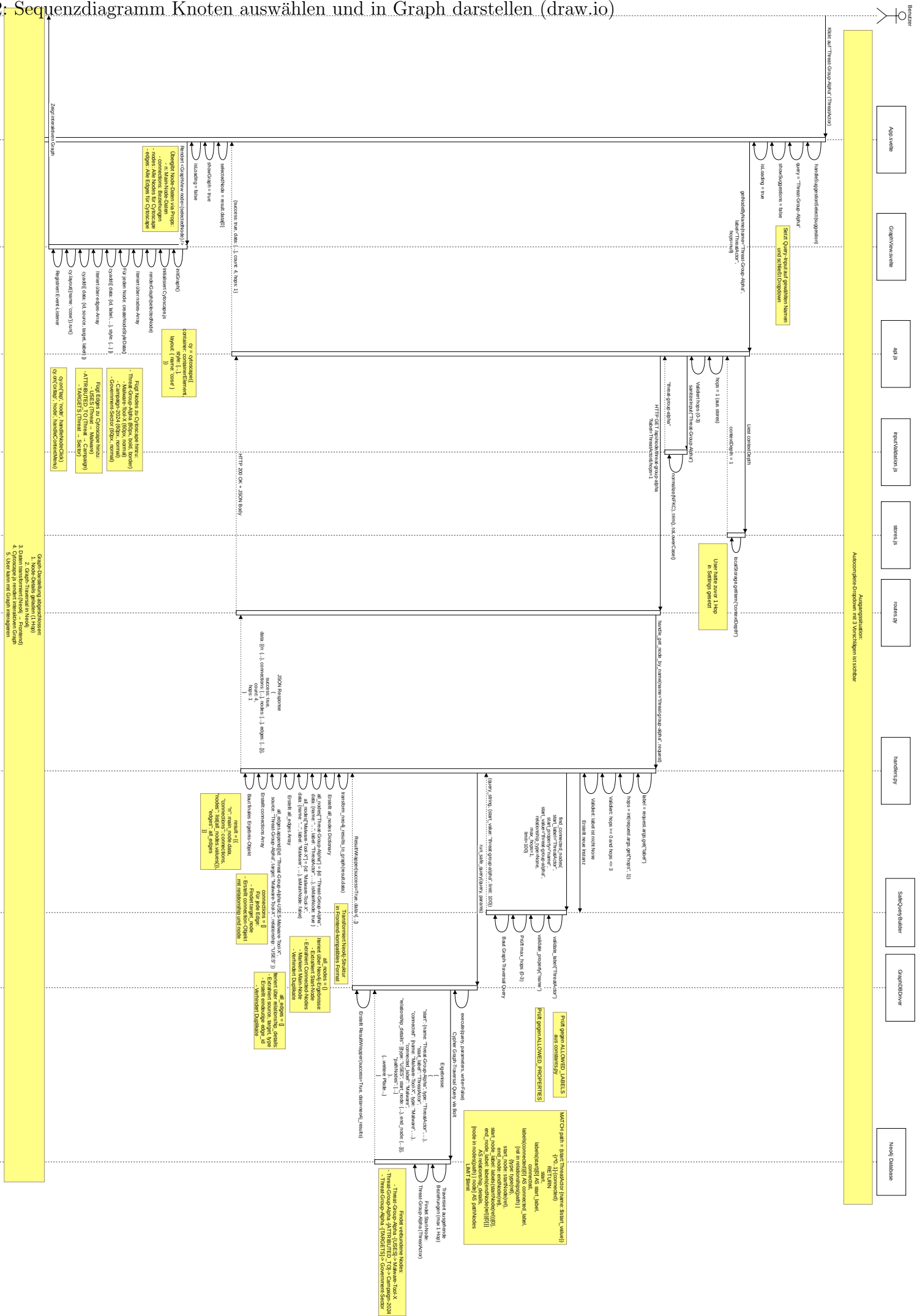
```

### 5.3 Sequenzdiagramme Querformat

Abbildung 5.1: Sequenzdiagramm Autocomplete (draw.io)



### Fig 5.2: Sequenzdiagramm Knoten auswählen und in Graph darstellen (draw.io)



## 5.4 Zertifikate



Abbildung 5.3: Zertifikat Neo4j Fundamentals erhalten am 14.05.25

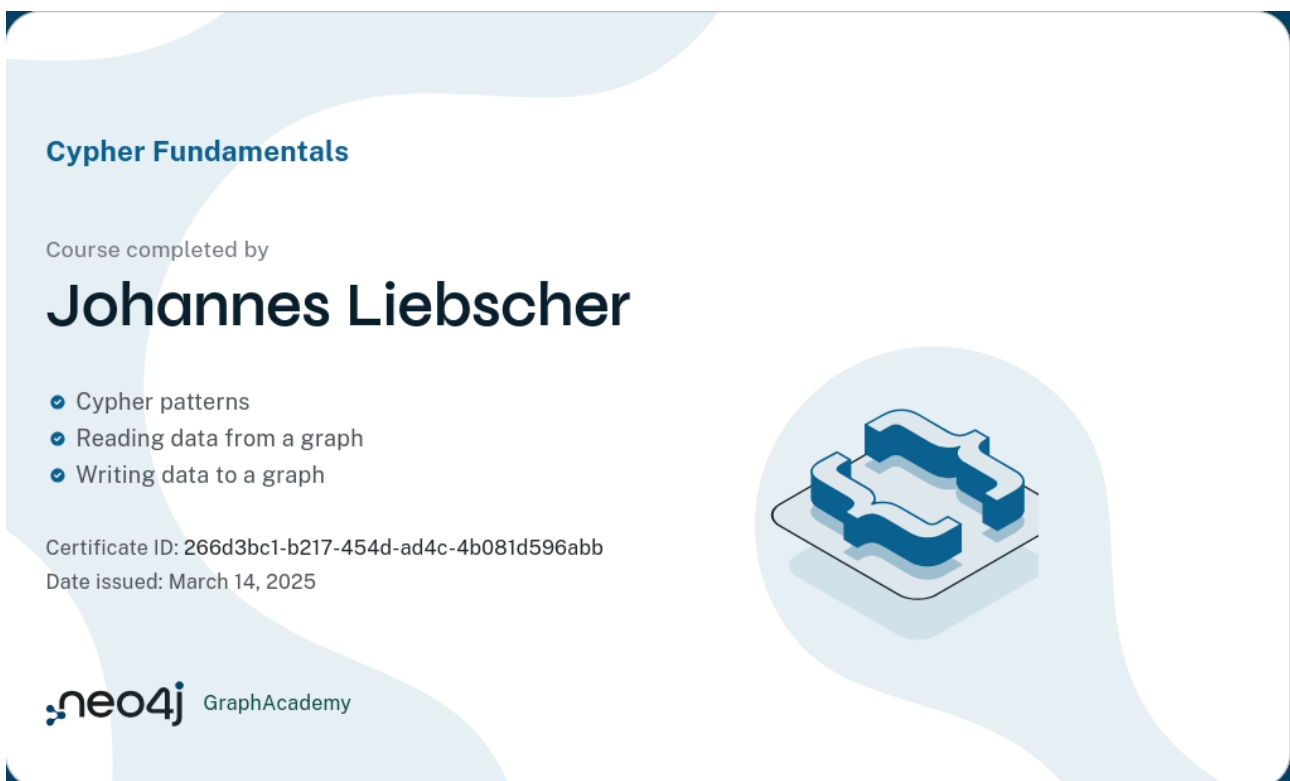


Abbildung 5.4: Zertifikat Cypher Fundamentals erhalten am 14.05.25



Abbildung 5.5: Zertifikat Graph Data Modeling Fundamentals erhalten am 16.05.25

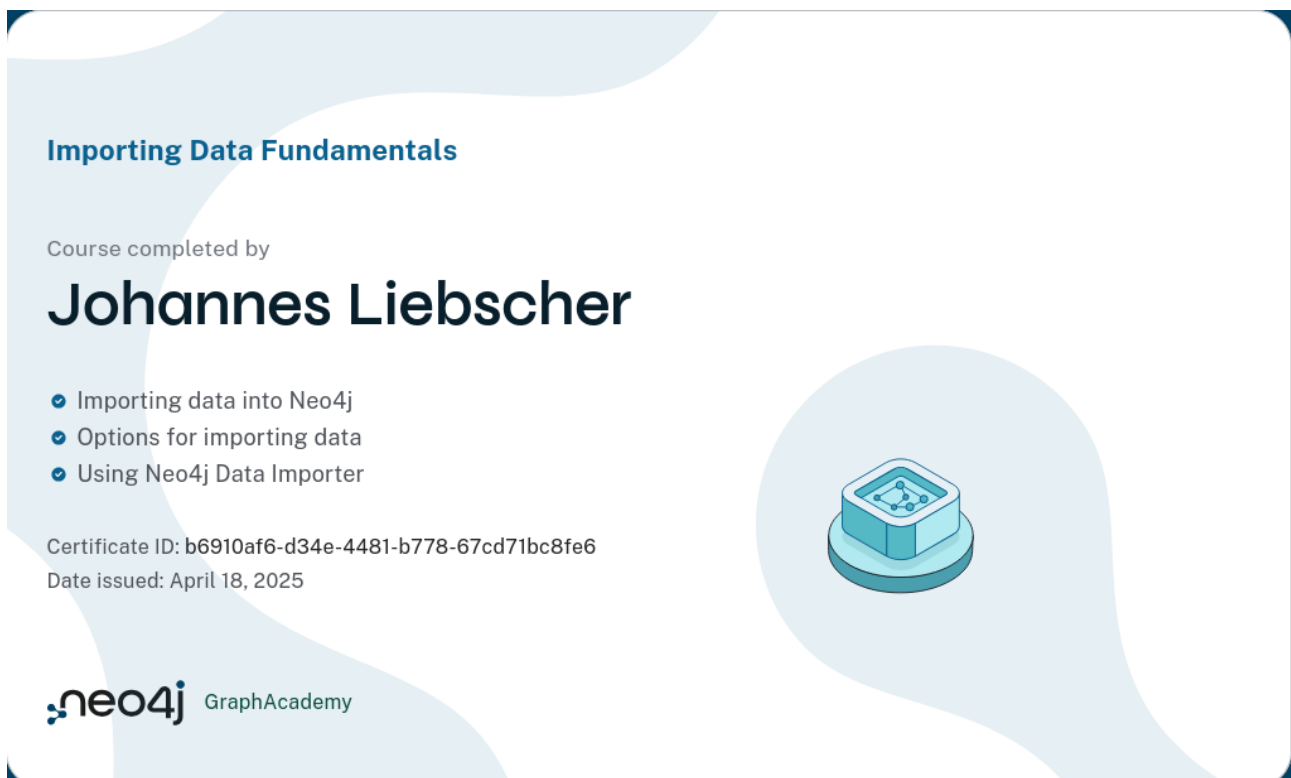


Abbildung 5.6: Zertifikat Importing Data Fundamentals erhalten am 18.05.25



Abbildung 5.7: Zertifikat Neo4j with Python erhalten am 24.05.25