DORA-konformer Szenariengenerator

Vollständige Dokumentation

Inhaltsverzeichnis

1. Projektübersicht

DORA-konformer Szenariengenerator für Krisenmanagement (MVP)

Ein Prototyp zur Generierung realistischer, logisch konsistenter Krisenszenarien (MSELs) für Finanzunternehmen, die den Anforderungen des **Digital Operational Resilience Act (DORA)** entsprechen.

🎯 Projektziel

Das System verwendet **Generative KI (LLMs)**, **Multi-Agenten-Systeme** und **Knowledge Graphs**, um: - Realistische Krisenszenarien zu generieren - Logische Konsistenz sicherzustellen - DORA-Konformität zu validieren - Second-Order Effects zu modellieren

🏗️ Architektur

High-Level Übersicht

> *[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*

graph TB subgraph "Frontend Layer" ST[Streamlit UI] end

```
subgraph "Orchestration Layer"
    LG[LangGraph Workflow]
end

subgraph "Agent Layer"
    MA[Manager Agent]
    GA[Generator Agent]
    CA[Critic Agent]
    IA[Intel Agent]
end

subgraph "Data Layer"
    NEO[Neo4j<br/>Knowledge Graph]
    CHROMA[ChromaDB<br/>Vector DB]
    LLM[OpenAI GPT-4o]
end

ST --> LG
LG --> MA
LG --> GA
LG --> CA
LG --> IA

MA --> LLM
GA --> LLM
CA --> LLM

IA --> CHROMA
LG --> NEO
GA --> NEO
CA --> NEO
```

```
**📊 Detaillierte Architektur-Diagramme**: Siehe [ARCHITECTURE.md](ARCHITECTURE.md)

### Tech Stack

- **Sprache:** Python 3.10+
- **Frontend:** Streamlit
- **Orchestrierung:** LangGraph (Multi-Agenten-System)
- **LLM:** OpenAI GPT-4o oder Llama 3
- **Knowledge Graph:** Neo4j
- **Vektor-Datenbank:** ChromaDB (für RAG)
- **Validierung:** Pydantic

### Komponenten

1. **State Models** (`state_models.py`): Pydantic-Modelle für Injects, State Manageme
2. **Neo4j Client** (`neo4j_client.py`): Verwaltung des Knowledge Graph States
3. **LangGraph Workflow**: Orchestrierung der Agenten (Manager, Generator, Critic, In
4. **Streamlit Frontend**: Parametereingabe und Visualisierung

## 🚀 Setup

### 1. Installation

```bash
# Virtual Environment erstellen
python -m venv venv
source venv/bin/activate  # Auf Windows: venv\Scripts\activate

# Dependencies installieren
pip install -r requirements.txt
```

2. Umgebungsvariablen

Kopiere `.env.example` zu `.env` und fülle die Werte aus:

```
cp .env.example .env
```
```

Bearbeite `.env`: - `NEO4J_URI`: Neo4j Verbindungs-URI (Standard: `bolt://localhost:7687`) - `NEO4J_USER`: Neo4j Benutzername - `NEO4J_PASSWORD`: Neo4j Passwort - `OPENAI_API_KEY`: OpenAI API Key

3. Neo4j Setup

Stelle sicher, dass Neo4j läuft:

```
# Mit Docker
docker run -d \
  --name neo4j \
  -p 7474:7474 -p 7687:7687 \
  -e NEO4J_AUTH=neo4j/password \
  neo4j:latest
```

4. Neo4j starten

```
# Mit dem bereitgestellten Skript
./start_neo4j.sh

# Oder manuell mit Docker
docker run -d \
  --name neo4j \
  -p 7474:7474 -p 7687:7687 \
  -e NEO4J_AUTH=neo4j/password \
  neo4j:latest
```

5. Setup testen

```
# Prüfe ob alles funktioniert
python check_setup.py

# Teste den Workflow
python test_workflow.py
```

📋 Verwendung

Frontend (Empfohlen)

Die einfachste Art, das System zu nutzen, ist über das Streamlit Frontend:

1. Starte die App: `streamlit run app.py`
2. Wähle Szenario-Typ und Anzahl Injects
3. Klicke auf "Szenario generieren"
4. Prüfe Ergebnisse im "Ergebnisse" Tab
5. Exportiere bei Bedarf (CSV/JSON)

Siehe FRONTEND.md für detaillierte Anleitung.

Programmgesteuerte Nutzung

State Models

Die Pydantic-Modelle in `state_models.py` definieren:

- **`Inject`**: MSEL-Inject Schema mit Validierung
- **`ScenarioState`**: Zustand eines laufenden Szenarios
- **`KnowledgeGraphEntity`**: Entität für den Neo4j Graph
- **`ValidationResult`**: Ergebnis der Critic-Agent Validierung

Neo4j Client

Der `Neo4jClient` verwaltet den Systemzustand:

```
from neo4j_client import Neo4jClient

with Neo4jClient() as client:
    # Aktuellen State abfragen
    entities = client.get_current_state()

    # Status einer Entität aktualisieren
    client.update_entity_status("SRV-001", "offline", inject_id="INJ-005")

    # Second-Order Effects abfragen
    affected = client.get_affected_entities("SRV-001")
```

Workflow

```
from workflows.scenario_workflow import ScenarioWorkflow
from state_models import ScenarioType

workflow = ScenarioWorkflow(neo4j_client=neo4j, max_iterations=10)
result = workflow.generate_scenario(ScenarioType.RANSOMWARE_DOUBLE_EXTORTION)
```

🔄 Workflow (implementiert)

Der LangGraph-basierte Workflow orchestriert folgende Schritte:

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

stateDiagram-v2 [*] --> StateCheck

```
StateCheck --> Manager: System State
Manager --> Intel: Storyline Plan
Intel --> ActionSelection: TTPs
ActionSelection --> Generator: Selected TTP
Generator --> Critic: Draft Inject
Critic --> StateUpdate: Valid
Critic --> Generator: Invalid (Refine)
StateUpdate --> StateCheck: Continue
StateUpdate --> [*]: End
```

```
**Detaillierte Schritte:**

1. **State Check**: Abfrage des aktuellen Systemzustands aus Neo4j
2. **Manager Agent**: Erstellt Storyline-Plan basierend auf Szenario-Typ und Phase
3. **Intel Agent**: Stellt relevante MITRE ATT&CK TTPs bereit
4. **Action Selection**: Auswahl des nächsten logischen Angriffsschritts
5. **Generator Agent**: Erstellt detaillierten Inject mit LLM
6. **Critic Agent**: Validiert Logik, DORA-Konformität und Causal Validity
7. **State Update**: Schreibt Auswirkungen in Neo4j (inkl. Second-Order Effects)
8. **Refine Loop**: Bei Validierungsfehlern zurück zum Generator (max. 2 Versuche)

**📊 Vollständige Workflow-Diagramme**: Siehe [ARCHITECTURE.md](ARCHITECTURE.md)

## 📊 Szenario-Typen

- Ransomware & Double Extortion
- DDoS auf kritische Funktionen
- Supply Chain Compromise
- Insider Threat / Datenmanipulation

## 🧪 Qualitäts-Metriken

- **Logical Consistency**: Widerspruchsfreiheit zur Historie
- **Causal Validity**: MITRE ATT&CK Graph Konformität
- **DORA Compliance**: Artikel 25 Anforderungen

## 🎯 Schnellstart

### Frontend starten

```bash
# Virtual Environment aktivieren
source venv/bin/activate

# Streamlit App starten
streamlit run app.py
```
```

Die App öffnet sich automatisch im Browser unter `http://localhost:8501`

Programmgesteuert verwenden

```
from neo4j_client import Neo4jClient
from workflows.scenario_workflow import ScenarioWorkflow
from state_models import ScenarioType

# Initialisiere
neo4j = Neo4jClient()
neo4j.connect()

workflow = ScenarioWorkflow(neo4j_client=neo4j, max_iterations=10)

# Generiere Szenario
result = workflow.generate_scenario(
    scenario_type=ScenarioType.RANSOMWARE_DOUBLE_EXTORTION
)

# Verarbeite Ergebnisse
for inject in result['injects']:
    print(f"{inject.inject_id}: {inject.content}")
```

✅ Implementierte Features

- ✅ LangGraph Workflow mit Multi-Agenten-System
- ✅ 4 Agenten: Manager, Generator, Critic, Intel
- ✅ Neo4j Knowledge Graph für State Management
- ✅ FSM-basierte Phasen-Übergänge
- ✅ Streamlit Frontend mit Visualisierungen
- ✅ Export-Funktionalität (CSV, JSON)
- ✅ DORA-Compliance Validierung
- ✅ MITRE ATT&CK Integration
- ✅ Second-Order Effects Tracking

📝 Nächste Schritte / Verbesserungspotenzial

Siehe STATUS.md für detaillierte Informationen zu: - Was das System jetzt kann - Was noch fehlt - Wie es eingesetzt werden kann - Roadmap für zukünftige Features

📄 Lizenz

[Zu definieren]

---

2. Status & Capabilities

📊 Projekt-Status & Capabilities

✅ Was das System jetzt kann

🎯 Kernfunktionalität

1. Szenario-Generierung

- ✅ Generierung realistischer Krisenszenarien für Finanzunternehmen
- ✅ Unterstützung für 4 Szenario-Typen:
- Ransomware & Double Extortion
- DDoS auf kritische Funktionen
- Supply Chain Compromise
- Insider Threat / Datenmanipulation
- ✅ Automatische Phasen-Übergänge (FSM-basiert)
- ✅ Konfigurierbare Anzahl von Injects (1-20)

2. Multi-Agenten-System (LangGraph)

- ✅ **Manager Agent**: Erstellt Storyline-Pläne basierend auf Szenario-Typ und Systemzustand
- ✅ **Generator Agent**: Generiert detaillierte, realistische Injects mit LLM
- ✅ **Critic Agent**: Validiert Injects auf:
- Logische Konsistenz
- DORA-Compliance (Artikel 25)
- Causal Validity (MITRE ATT&CK)
- ✅ **Intel Agent**: Stellt relevante TTPs (Taktiken, Techniken, Prozeduren) bereit

3. State Management

- ✅ Neo4j Knowledge Graph für Systemzustand

- ✅ Tracking von Assets (Server, Applikationen, Abteilungen)
- ✅ Second-Order Effects (indirekte Auswirkungen)
- ✅ Status-Updates basierend auf Injects

4. Validierung & Qualitätssicherung

- ✅ Pydantic-basierte Schema-Validierung
- ✅ FSM-Validierung für Phasen-Übergänge
- ✅ LLM-basierte Konsistenz-Prüfung
- ✅ Refine-Loop bei Validierungsfehlern (max. 2 Versuche)

5. Frontend (Streamlit)

- ✅ Benutzerfreundliche Web-UI
- ✅ Parametereingabe (Szenario-Typ, Anzahl Injects)
- ✅ Detaillierte Inject-Anzeige
- ✅ Visualisierungen (Phasen-Verteilung, Timeline)
- ✅ Export-Funktionen (CSV, JSON)

6. Datenmodell

- ✅ Vollständiges Inject-Schema (Pydantic)
- ✅ Technical Metadata (MITRE IDs, IOCs, Assets)
- ✅ DORA Compliance Tags
- ✅ Business Impact Tracking

🔧 Technische Features

- ✅ LangGraph Workflow-Orchestrierung
- ✅ OpenAI GPT-4o Integration
- ✅ Neo4j Knowledge Graph
- ✅ ChromaDB für TTP-Vektor-Datenbank (Grundstruktur)
- ✅ Automatische Fehlerbehandlung
- ✅ Session Management (Streamlit)

⚠️ Was noch fehlt / Verbesserungspotenzial

🔴 Kritische Features (für Produktion)

1. **ChromaDB TTP-Datenbank**
2. ❌ Vollständige MITRE ATT&CK TTP-Datenbank noch nicht geladen
3. ⚠️ Aktuell: Fallback-TTPs werden verwendet
4. 📝 **Nächster Schritt**: MITRE ATT&CK Daten importieren
5. **Erweiterte Validierung**
6. ❌ NLI-Modelle für tiefere Konsistenz-Prüfung
7. ❌ Automatische Widerspruchserkennung zwischen Injects
8. 📝 **Nächster Schritt**: NLI-Modell Integration
9. **Fehlerbehandlung**
10. ⚠️ Teilweise: Bessere Fehlerbehandlung bei LLM-Aufrufen
11. ⚠️ Retry-Logik für API-Calls
12. 📝 **Nächster Schritt**: Robustere Error Handling

🟡 Wichtige Features (für erweiterte Nutzung)

1. **TIBER-EU Konformität**
2. ❌ "Flags" (Ziele) Generierung
3. ❌ "Leg-ups" (Hilfestellungen) Generierung
4. 📝 **Nächster Schritt**: TIBER-spezifische Features
5. **Komplexitäts-Parameter**
6. ⚠️ Teilweise: Proportionalitätsprinzip noch nicht vollständig implementiert
7. ❌ Parametrisierung für verschiedene Unternehmensgrößen
8. 📝 **Nächster Schritt**: Komplexitäts-Slider im Frontend
9. **Export-Formate**
10. ✅ CSV, JSON

11. ❌ Excel (.xlsx)
12. ❌ MSEL-Format (Standard für Übungen)

13. 📝 **Nächster Schritt**: Excel & MSEL Export

14. **Historische Szenarien**

15. ❌ Speicherung von generierten Szenarien
16. ❌ Vergleich zwischen Szenarien
17. ❌ Wiederverwendung von erfolgreichen Szenarien
18. 📝 **Nächster Schritt**: Datenbank für Szenarien

🟢 Nice-to-Have Features

1. **Erweiterte Visualisierungen**
2. ⚠️ Basis: Phasen-Verteilung, Timeline
3. ❌ Interaktive Graphen (Neo4j Visualisierung)
4. ❌ Attack-Kill-Chain Visualisierung

5. 📝 **Nächster Schritt**: Graph-Visualisierung

6. **Templates & Vorlagen**

7. ❌ Vordefinierte Szenario-Templates
8. ❌ Wiederverwendbare Inject-Patterns

9. 📝 **Nächster Schritt**: Template-System

10. **Multi-User Support**
    - ❌ Benutzer-Authentifizierung
    - ❌ Projekt-Management
    - ❌ Kollaboration
    - 📝 **Nächster Schritt**: User Management

11. **API-Endpoints**
    - ❌ REST API für externe Integration
    - ❌ Webhook-Support
    - 📝 **Nächster Schritt**: FastAPI Integration

12. **Testing & Qualitätssicherung**
    - ⚠️ Teilweise: Basis-Tests vorhanden
    - ❌ Unit Tests für alle Agenten
    - ❌ Integration Tests
    - ❌ End-to-End Tests
    - 📝 **Nächster Schritt**: Test-Suite erweitern

🚀 Wie das System eingesetzt werden kann

📋 Aktuelle Anwendungsfälle

1. Krisenübungen vorbereiten

```
Zweck: Realistische MSELs (Master Scenario Event Lists) für Übungen generieren

Workflow:
1. Frontend öffnen (streamlit run app.py)
2. Szenario-Typ wählen (z.B. Ransomware)
3. Anzahl Injects konfigurieren (z.B. 10)
4. Szenario generieren
5. Injects prüfen und anpassen
6. Als CSV/JSON exportieren
7. In Übungs-Tool importieren
```

2. DORA-Compliance prüfen

```
Zweck: Prüfen ob Szenarien DORA Artikel 25 Anforderungen erfüllen

Workflow:
1. Szenario generieren
2. DORA Tags in Ergebnissen prüfen
3. Validierungsdetails anzeigen
4. Bei Bedarf anpassen und neu generieren
```

3. Threat-Led Penetration Testing (TLPT)

```
Zweck: Szenarien für TIBER-EU konforme Tests erstellen

Workflow:
1. Szenario generieren
2. MITRE ATT&CK TTPs analysieren
3. Attack-Kill-Chain nachvollziehen
4. Für Red Team Übungen verwenden
```

4. Business Continuity Planung

```
Zweck: Geschäftliche Auswirkungen von Cyber-Angriffen simulieren

Workflow:
1. Szenario mit Business Impact generieren
2. Betroffene Assets analysieren
3. Second-Order Effects prüfen
4. Business Continuity Pläne anpassen
```

🔧 Technische Integration

Als Standalone-Tool

```
# Direkte Nutzung über Streamlit
streamlit run app.py
```

Als Python-Modul

```
from neo4j_client import Neo4jClient
from workflows.scenario_workflow import ScenarioWorkflow
from state_models import ScenarioType

# Initialisiere
neo4j = Neo4jClient()
neo4j.connect()

workflow = ScenarioWorkflow(neo4j_client=neo4j, max_iterations=10)

# Generiere Szenario
result = workflow.generate_scenario(
    scenario_type=ScenarioType.RANSOMWARE_DOUBLE_EXTORTION
)

# Verarbeite Ergebnisse
for inject in result['injects']:
    print(f"{inject.inject_id}: {inject.content}")
```

Export & Weiterverarbeitung

```
# CSV Export
import pandas as pd
from app import export_to_csv

csv_data = export_to_csv(result['injects'])
# Weiterverarbeitung in Excel, etc.

# JSON Export
from app import export_to_json
json_data = export_to_json(result['injects'])
# API-Integration, etc.
```

📊 Empfohlene Workflows

Schneller Test (3-5 Injects)

- Für erste Tests und Konzept-Validierung
- Dauer: ~2-5 Minuten
- Ideal für: Schnelle Prototypen

Standard-Szenario (10-15 Injects)

- Für vollständige Übungen
- Dauer: ~10-15 Minuten
- Ideal für: Reguläre Krisenübungen

Komplexes Szenario (15-20 Injects)

- Für umfassende Tests
- Dauer: ~20-30 Minuten
- Ideal für: Große Übungen, Audits

⚙️ Konfiguration

Umgebungsvariablen (.env)

```
# Neo4j
NEO4J_URI=bolt://localhost:7687
NEO4J_USER=neo4j
NEO4J_PASSWORD=your_password

# OpenAI
OPENAI_API_KEY=your_api_key

# ChromaDB (optional)
CHROMA_DB_PATH=./chroma_db
```

Workflow-Parameter

- `max_iterations`: Anzahl Injects (1-20)
- `scenario_type`: Szenario-Typ

- `auto_phase_transition`: Automatische Phasen-Übergänge

📈 Roadmap

Phase 1: MVP (✅ Abgeschlossen)

- ✅ Grundstruktur
- ✅ Agenten-Implementierung
- ✅ Frontend
- ✅ Basis-Validierung

Phase 2: Erweiterte Features (🔄 In Arbeit)

- 🔄 ChromaDB TTP-Datenbank
- 🔄 Erweiterte Validierung
- 🔄 Excel Export
- 🔄 TIBER-EU Features

Phase 3: Produktionsreife (📅 Geplant)

- 📅 Vollständige Test-Suite
- 📅 API-Endpoints
- 📅 Multi-User Support
- 📅 Performance-Optimierung

🎓 Best Practices

1. **Erste Nutzung**: Starte mit 3-5 Injects zum Testen
2. **Neo4j**: Stelle sicher, dass Neo4j läuft vor der Generierung
3. **Validierung**: Prüfe Validierungswarnungen in den Ergebnissen
4. **Export**: Exportiere regelmäßig für Backup
5. **Anpassungen**: Passe Injects manuell an, wenn nötig

📞 Support & Weiterentwicklung

- **Dokumentation**: Siehe README.md, SETUP.md, FRONTEND.md
- **Tests**: `python test_workflow.py`
- **Setup-Prüfung**: `python check_setup.py`

Letzte Aktualisierung: 2025-01-XX **Version**: MVP 1.0

---

3. Schnellstart

🚀 Quick Start Guide

Schnellstart-Anleitung für den DORA-Szenariengenerator.

⚡ In 5 Minuten zum ersten Szenario

Schritt 1: Voraussetzungen prüfen

```
# Python 3.10+ installiert?
python3 --version

# Docker installiert? (für Neo4j)
docker --version
```

Schritt 2: Projekt einrichten

```
# Repository klonen/öffnen
cd BA

# Virtual Environment erstellen
python3 -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# Dependencies installieren
pip install -r requirements.txt
```

Schritt 3: Konfiguration

```
# .env Datei erstellen
cp .env.example .env

# .env bearbeiten und eintragen:
# - NEO4J_PASSWORD=dein_passwort
# - OPENAI_API_KEY=dein_api_key
```

Schritt 4: Neo4j starten

```
# Docker starten (falls nicht läuft)
# Dann Neo4j Container starten
./start_neo4j.sh
```

Schritt 5: Setup testen

```
# Prüfe ob alles funktioniert
python check_setup.py
```

Schritt 6: Frontend starten

```
streamlit run app.py
```

Schritt 7: Erstes Szenario generieren

1. Öffne Browser: `http://localhost:8501`
2. Wähle Szenario-Typ (z.B. "Ransomware Double Extortion")
3. Setze Anzahl Injects auf 3 (für schnellen Test)
4. Klicke auf "🎯 Szenario generieren"
5. Warte ~2-5 Minuten
6. Prüfe Ergebnisse im "Ergebnisse" Tab

📚 Weitere Dokumentation

- **README.md**: Vollständige Projekt-Dokumentation
- **STATUS.md**: Was kann das System, was fehlt, wie einsetzen
- **SETUP.md**: Detaillierte Setup-Anleitung
- **FRONTEND.md**: Frontend-Bedienungsanleitung

🆘 Häufige Probleme

Neo4j-Verbindungsfehler

```
# Prüfe ob Neo4j läuft
docker ps | grep neo4j

# Starte Neo4j falls nicht
./start_neo4j.sh
```

OpenAI API Fehler

- Prüfe `.env` Datei
- Stelle sicher, dass `OPENAI_API_KEY` gesetzt ist
- Prüfe API Key Gültigkeit

Import-Fehler

```
# Stelle sicher, dass venv aktiviert ist
source venv/bin/activate

# Reinstalliere Dependencies
pip install -r requirements.txt
```

✅ Erfolg!

Wenn du dein erstes Szenario generiert hast, kannst du: - Injects im Frontend ansehen - Als CSV/JSON exportieren - Visualisierungen prüfen - Mit verschiedenen Szenario-Typen experimentieren

Viel Erfolg! 🎯

---

4. Setup-Anleitung

🚀 Schnellstart-Anleitung

Schritt 1: Python Virtual Environment erstellen

```
# Virtual Environment erstellen
python3 -m venv venv

# Aktivieren (macOS/Linux)
source venv/bin/activate

# Aktivieren (Windows)
venv\Scripts\activate
```

Schritt 2: Dependencies installieren

```
pip install -r requirements.txt
```

Schritt 3: Umgebungsvariablen konfigurieren

```
# .env Datei erstellen
cp .env.example .env
```

Dann öffne `.env` und trage deine Werte ein:

```
# Neo4j (optional für jetzt - kann später konfiguriert werden)
NEO4J_URI=bolt://localhost:7687
NEO4J_USER=neo4j
NEO4J_PASSWORD=dein_passwort_hier

# OpenAI (wird später für LLM benötigt)
OPENAI_API_KEY=dein_api_key_hier
```

Schritt 4: Setup testen

```
python test_setup.py
```

Dieser Test prüft: - ✅ Pydantic-Modelle funktionieren - ✅ Neo4j-Verbindung (falls konfiguriert)

Schritt 5: Neo4j starten (optional)

Falls du Neo4j testen möchtest:

```
# Mit Docker
docker run -d \
  --name neo4j \
  -p 7474:7474 -p 7687:7687 \
  -e NEO4J_AUTH=neo4j/password \
  neo4j:latest
```

Dann in `.env` eintragen:

```
NEO4J_PASSWORD=password
```

✅ Fertig!

Wenn `test_setup.py` erfolgreich durchläuft, ist die Grundstruktur bereit.

Nächste Entwicklungsschritte

1. **LangGraph Workflow** implementieren
2. **Agenten** entwickeln (Manager, Generator, Critic, Intel)
3. **ChromaDB** für RAG einrichten
4. **Streamlit Frontend** erstellen

---

5. Frontend-Anleitung

🎨 Streamlit Frontend

🚀 Starten

```
# Virtual Environment aktivieren
source venv/bin/activate

# Streamlit App starten
streamlit run app.py
```

Die App öffnet sich automatisch im Browser unter `http://localhost:8501`

📋 Features

1. Generierung Tab

- Szenario-Typ auswählen (Ransomware, DDoS, Supply Chain, Insider Threat)
- Anzahl Injects konfigurieren (1-20)
- Erweiterte Optionen:
- Automatische Phasen-Übergänge
- Validierungsdetails anzeigen

2. Ergebnisse Tab

- Übersicht aller generierten Injects
- Detaillierte Anzeige pro Inject:
- Inject ID & Zeitversatz
- Phase (mit farblicher Markierung)
- Quelle & Ziel

- Modalität
- Inhalt
- MITRE ID
- Betroffene Assets
- DORA Compliance Tag
- Business Impact
- Export-Funktionen:
- CSV Export
- JSON Export

3. Visualisierung Tab

- Phasen-Verteilung (Balkendiagramm)
- Timeline-Übersicht
- Betroffene Assets-Liste

🎯 Verwendung

1. **Konfiguration** (Sidebar):
2. Wähle Szenario-Typ
3. Setze Anzahl Injects
4. Aktiviere/deaktiviere erweiterte Optionen
5. **Generierung**:
6. Klicke auf "🎯 Szenario generieren"
7. Warte auf Abschluss (kann einige Minuten dauern)
8. Erfolgsmeldung erscheint
9. **Ergebnisse ansehen**:
10. Wechsle zum "📊 Ergebnisse" Tab
11. Scrolle durch alle Injects
12. Exportiere bei Bedarf
13. **Visualisierung**:
14. Wechsle zum "📈 Visualisierung" Tab
15. Analysiere Phasen-Verteilung und Timeline

⚠️ Wichtige Hinweise

- **Neo4j muss laufen**: Stelle sicher, dass Neo4j läuft (`./start_neo4j.sh`)
- **OpenAI API Key**: Muss in `.env` konfiguriert sein
- **Erste Generierung**: Kann länger dauern (LLM-Aufrufe)
- **Session State**: Ergebnisse bleiben während der Session erhalten

🔧 Troubleshooting

App startet nicht

```
# Prüfe ob Streamlit installiert ist
pip install streamlit

# Prüfe Python-Version
python --version  # Sollte 3.10+ sein
```

Neo4j-Verbindungsfehler

```
# Starte Neo4j
./start_neo4j.sh

# Prüfe Verbindung
python check_setup.py
```

OpenAI API Fehler

- Prüfe `.env` Datei
- Stelle sicher, dass `OPENAI_API_KEY` gesetzt ist
- Prüfe API Key Gültigkeit

---

6. Architektur

🏗️ Architektur-Dokumentation

Detaillierte Architektur-Diagramme und Beschreibungen des DORA-Szenariengenerators.

📊 Übersicht

High-Level Architektur

> *[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*

graph TB subgraph "Frontend Layer" ST[Streamlit UI] end

```
subgraph "Orchestration Layer"
    LG[LangGraph Workflow]
end

subgraph "Agent Layer"
    MA[Manager Agent]
    GA[Generator Agent]
    CA[Critic Agent]
    IA[Intel Agent]
end

subgraph "Data Layer"
    NEO[Neo4j<br/>Knowledge Graph]
    CHROMA[ChromaDB<br/>Vector DB]
    LLM[OpenAI GPT-4o]
end

ST --> LG
LG --> MA
LG --> GA
LG --> CA
LG --> IA

MA --> LLM
GA --> LLM
CA --> LLM

IA --> CHROMA
LG --> NEO
GA --> NEO
CA --> NEO

style ST fill:#1f77b4
style LG fill:#ff7f0e
style MA fill:#2ca02c
style GA fill:#2ca02c
style CA fill:#2ca02c
style IA fill:#2ca02c
```

```
style NEO fill:#d62728
style CHROMA fill:#9467bd
style LLM fill:#8c564b
```

```
## 🔄 Workflow-Architektur

### LangGraph Workflow Flow
```

[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]

```
stateDiagram-v2
    [*] --> StateCheck

    StateCheck --> Manager: System State
    Manager --> Intel: Storyline Plan
    Intel --> ActionSelection: TTPs
    ActionSelection --> Generator: Selected TTP
    Generator --> Critic: Draft Inject
    Critic --> StateUpdate: Valid
    Critic --> Generator: Invalid (Refine)
    StateUpdate --> StateCheck: Continue
    StateUpdate --> [*]: End

    note right of Critic
        Max 2 Refine Attempts
        per Inject
    end note

    note right of StateUpdate
        Updates Neo4j
        Tracks Second-Order Effects
    end note
```

Detaillierter Workflow

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

sequenceDiagram participant User participant Frontend as Streamlit UI participant Workflow as LangGraph Workflow participant Manager as Manager Agent participant Intel as Intel Agent participant Generator as Generator Agent participant Critic as Critic Agent participant Neo4j participant ChromaDB participant OpenAI

```
User->>Frontend: Configure Scenario
Frontend->>Workflow: Start Generation

loop For each Inject
    Workflow->>Neo4j: Get Current State
    Neo4j-->>Workflow: System Entities

    Workflow->>Manager: Create Storyline
    Manager->>OpenAI: Generate Plan
    OpenAI-->>Manager: Storyline Plan
    Manager-->>Workflow: Plan

    Workflow->>Intel: Get Relevant TTPs
    Intel->>ChromaDB: Query TTPs
    ChromaDB-->>Intel: TTP List
    Intel-->>Workflow: TTPs

    Workflow->>Workflow: Select Action

    Workflow->>Generator: Generate Inject
    Generator->>OpenAI: Create Inject
    OpenAI-->>Generator: Draft Inject
    Generator-->>Workflow: Inject

    Workflow->>Critic: Validate Inject
    Critic->>OpenAI: Validate
    OpenAI-->>Critic: Validation Result
    Critic-->>Workflow: Result

    alt Valid
        Workflow->>Neo4j: Update State
        Neo4j-->>Workflow: Updated
    else Invalid
        Workflow->>Generator: Refine (max 2x)
    end
end
```

```
Workflow-->>Frontend: Scenario Result
Frontend-->>User: Display Results
```

```
## 🧩 Komponenten-Architektur

### Agent-Architektur
```

[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]

```
graph LR
    subgraph "Manager Agent"
        M1[Storyline Planning]
        M2[Phase Transition Logic]
        M3[LLM Integration]
    end

    subgraph "Generator Agent"
        G1[Inject Creation]
        G2[Content Generation]
        G3[Metadata Assignment]
    end

    subgraph "Critic Agent"
        C1[Logical Consistency]
        C2[DORA Compliance]
        C3[Causal Validity]
    end

    subgraph "Intel Agent"
        I1[TTP Retrieval]
        I2[Vector Search]
        I3[Phase Filtering]
    end

    M1 --> M2
    M2 --> M3

    G1 --> G2
    G2 --> G3

    C1 --> C2
    C2 --> C3

    I1 --> I2
    I2 --> I3

    style M1 fill:#2ca02c
```

```
    style G1 fill:#2ca02c
    style C1 fill:#2ca02c
    style I1 fill:#2ca02c
```

State Management Architektur

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

graph TB subgraph "State Models" SM[Pydantic Models] INJ[Inject Schema] SCEN[Scenario State] ENT[Graph Entities] end

```
subgraph "Neo4j Knowledge Graph"
    N1[Entities<br/>Server, Apps, Depts]
    N2[Relationships<br/>RUNS_ON, USES]
    N3[Status Tracking<br/>online, offline, compromised]
end

subgraph "FSM"
    F1[Phase States]
    F2[Transition Rules]
    F3[Validation Logic]
end

SM --> INJ
SM --> SCEN
SM --> ENT

ENT --> N1
ENT --> N2
ENT --> N3

SCEN --> F1
F1 --> F2
F2 --> F3

style SM fill:#1f77b4
style N1 fill:#d62728
style F1 fill:#ff7f0e
```

```
## 📦 Datenfluss

### Inject-Generierungs-Pipeline
```

[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]

```
flowchart TD
    START([User Request]) --> CONFIG[Configuration<br/>Type, Count]

    CONFIG --> LOOP{More Injects?}

    LOOP -->|Yes| STATE[State Check<br/>Neo4j Query]
    LOOP -->|No| EXPORT[Export Results]

    STATE --> PLAN[Manager: Storyline Plan]
    PLAN --> TTP[Intel: Get TTPs]
    TTP --> SELECT[Action Selection]
    SELECT --> GEN[Generator: Create Inject]

    GEN --> VALID[Critic: Validate]

    VALID -->|Valid| UPDATE[Update Neo4j State]
    VALID -->|Invalid| REFINE{Refine Count < 2?}

    REFINE -->|Yes| GEN
    REFINE -->|No| UPDATE

    UPDATE --> LOOP

    EXPORT --> END([Complete])

    style START fill:#2ca02c
    style END fill:#d62728
    style VALID fill:#ff7f0e
    style UPDATE fill:#9467bd
```

🔐 Sicherheits-Architektur

Datenfluss und Sicherheit

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

graph TB subgraph "Secure Storage" ENV[.env File
NOT in Git] NEO_PASS[Neo4j Password] API_KEY[OpenAI API Key] end

```
subgraph "Application"
    APP[Streamlit App]
    WORKFLOW[Workflow]
end

subgraph "External Services"
    OPENAI[OpenAI API]
    NEO4J[Neo4j Database]
end

ENV --> APP
NEO_PASS --> NEO4J
API_KEY --> OPENAI

APP --> WORKFLOW
WORKFLOW --> OPENAI
WORKFLOW --> NEO4J

style ENV fill:#d62728
style NEO_PASS fill:#d62728
style API_KEY fill:#d62728
```

```
## 🗄️ Datenmodell

### Entity-Relationship Diagram
```

[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]

```
erDiagram
    SCENARIO ||--o{ INJECT : contains
    SCENARIO {
        string scenario_id
        enum scenario_type
        enum current_phase
        datetime start_time
    }

    INJECT ||--|| TECHNICAL_METADATA : has
    INJECT {
        string inject_id
        string time_offset
        enum phase
        string source
        string target
        string content
        string dora_compliance_tag
    }

    TECHNICAL_METADATA {
        string mitre_id
        array affected_assets
        string ioc_hash
        string severity
    }

    ENTITY ||--o{ RELATIONSHIP : has
    ENTITY {
        string entity_id
        string entity_type
        string name
        string status
    }

    RELATIONSHIP {
        string source_id
        string target_id
```

```
        string relationship_type
    }

    INJECT ||--o{ ENTITY : affects
```

🔄 Phasen-Übergänge (FSM)

Finite State Machine

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

stateDiagram-v2 [*] --> NORMAL_OPERATION

```
NORMAL_OPERATION --> SUSPICIOUS_ACTIVITY : Detection
NORMAL_OPERATION --> INITIAL_INCIDENT : Direct Attack

SUSPICIOUS_ACTIVITY --> INITIAL_INCIDENT : Confirmed
SUSPICIOUS_ACTIVITY --> NORMAL_OPERATION : False Positive

INITIAL_INCIDENT --> ESCALATION_CRISIS : Severe Impact
INITIAL_INCIDENT --> CONTAINMENT : Quick Response

ESCALATION_CRISIS --> CONTAINMENT : Response Actions

CONTAINMENT --> RECOVERY : Systems Restored
CONTAINMENT --> ESCALATION_CRISIS : Re-Escalation

RECOVERY --> NORMAL_OPERATION : Full Recovery

note right of NORMAL_OPERATION
    Baseline State
    All Systems Operational
end note

note right of ESCALATION_CRISIS
    Critical State
    Business Impact
end note
```

```
## 📊 Deployment-Architektur

### Lokale Entwicklung
```

[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]

```
graph TB
    subgraph "Development Machine"
        DEV[Developer]
        IDE[IDE/Editor]
        VENV[Python venv]
        STREAMLIT[Streamlit App]
    end

    subgraph "Local Services"
        DOCKER[Docker]
        NEO4J_LOCAL[Neo4j Container]
    end

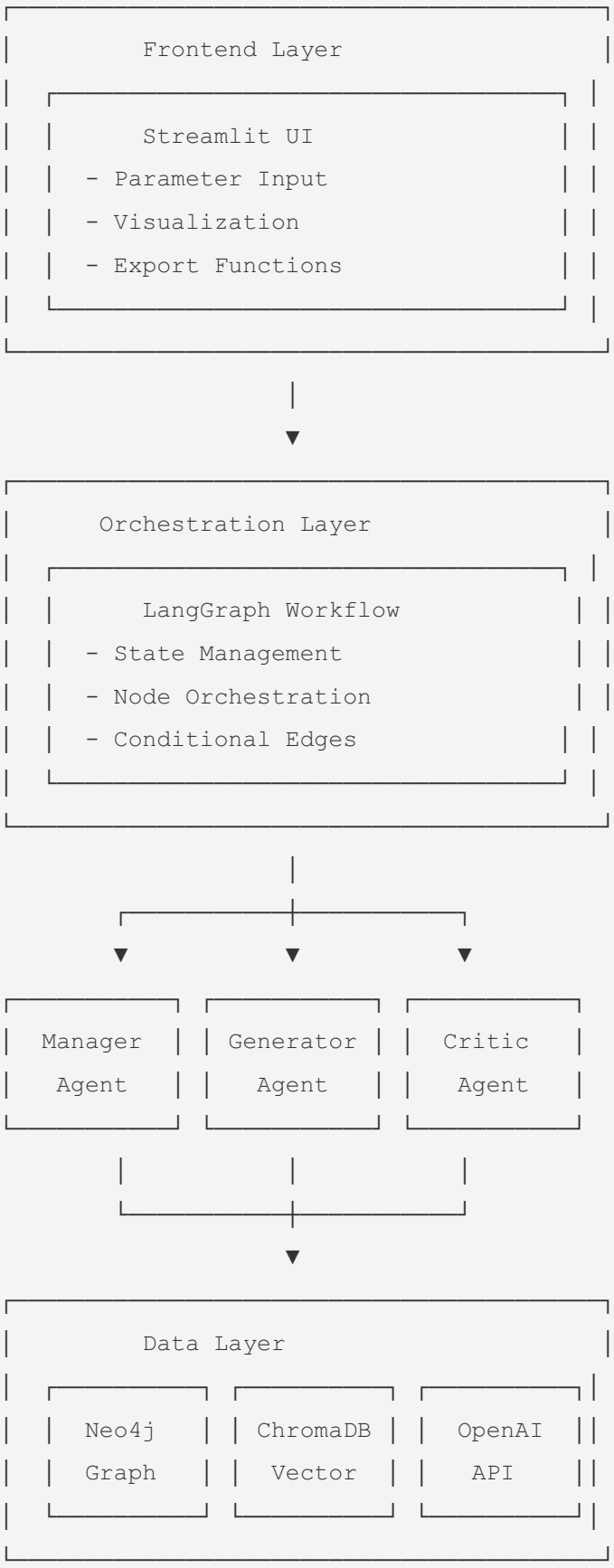
    subgraph "External Services"
        OPENAI_API[OpenAI API]
    end

    DEV --> IDE
    IDE --> VENV
    VENV --> STREAMLIT
    STREAMLIT --> NEO4J_LOCAL
    STREAMLIT --> OPENAI_API
    DOCKER --> NEO4J_LOCAL

    style DEV fill:#2ca02c
    style NEO4J_LOCAL fill:#d62728
    style OPENAI_API fill:#8c564b
```

🔧 Technologie-Stack

Technologie-Layers


Frontend Layer
Streamlit UI
- Parameter Input
- Visualization
- Export Functions
Orchestration Layer
LangGraph Workflow
- State Management
- Node Orchestration
- Conditional Edges
Manager Agent
Generator Agent
Critic Agent
Data Layer
Neo4j Graph
ChromaDB Vector
OpenAI API

📈 Skalierungs-Architektur

Zukünftige Erweiterungen

```
*[Mermaid-Diagramm - siehe Online-Dokumentation für interaktive Version]*
```

graph TB subgraph "Current MVP" C1[Single User] C2[Local Neo4j] C3[Streamlit Frontend] end

```
subgraph "Future: Multi-User"
    F1[User Management]
    F2[Project Sharing]
    F3[Collaboration]
end

subgraph "Future: Cloud"
    F4[Neo4j Cloud]
    F5[API Gateway]
    F6[Load Balancer]
end

C1 --> F1
C2 --> F4
C3 --> F5

style C1 fill:#2ca02c
style C2 fill:#2ca02c
style C3 fill:#2ca02c
style F1 fill:#ff7f0e
style F4 fill:#ff7f0e
style F5 fill:#ff7f0e
```

```
```

📝 Legende

Farb-Codierung

- 🔵 **Blau**: Frontend/UI Komponenten
- 🟠 **Orange**: Orchestration/Workflow
- 🟢 **Grün**: Agenten
- 🔴 **Rot**: Datenbanken/Storage
- 🟣 **Lila**: Externe Services
- 🟤 **Braun**: LLM/API Services

Diagramm-Typen

- **Mermaid**: Wird von GitHub und vielen Markdown-Viewern unterstützt
- **ASCII**: Fallback für einfache Text-Editoren
- **Flowcharts**: Für Prozess-Flows
- **State Diagrams**: Für FSM und Zustandsübergänge
- **Sequence Diagrams**: Für Interaktionen zwischen Komponenten

🔗 Verwandte Dokumentation

- README.md - Hauptdokumentation
- STATUS.md - Feature-Status
- SETUP.md - Setup-Anleitung

---

7. Dokumentations-Übersicht

📚 Dokumentations-Übersicht

Übersicht aller verfügbaren Dokumentationen für den DORA-Szenariengenerator.

🚀 Schnelleinstieg

Für neue Nutzer

1. **QUICKSTART.md** - In 5 Minuten zum ersten Szenario
2. **SETUP.md** - Detaillierte Setup-Anleitung
3. **FRONTEND.md** - Frontend-Bedienungsanleitung

📖 Hauptdokumentation

README.md

Hauptdokumentation des Projekts - Projektziel und Architektur - Tech Stack Übersicht - Setup-Anleitung - Verwendungsbeispiele - Komponenten-Übersicht

ARCHITECTURE.md

Detaillierte Architektur-Dokumentation - High-Level Architektur-Diagramme - Workflow-Diagramme (Mermaid) - Komponenten-Architektur - Datenfluss-Diagramme - FSM (Finite State Machine) Diagramme - Entity-Relationship Diagramme

📊 Status & Capabilities

STATUS.md

Detaillierte Status-Übersicht - ✅ Was das System jetzt kann - ⚠️ Was noch fehlt / Verbesserungspotenzial - 🚀 Wie das System eingesetzt werden kann - 📈 Roadmap - 🎓 Best Practices

Empfohlen für: - Projekt-Manager - Entwickler, die Features hinzufügen wollen - Stakeholder, die den aktuellen Stand verstehen wollen

🔧 Setup & Installation

SETUP.md

Detaillierte Setup-Anleitung - Schritt-für-Schritt Installation - Umgebungsvariablen-Konfiguration - Neo4j Setup - Troubleshooting

QUICKSTART.md

Schnellstart in 5 Minuten - Minimales Setup - Erste Schritte - Häufige Probleme

🎨 Frontend

FRONTEND.md

Streamlit Frontend Anleitung - Features-Übersicht - Verwendungsanleitung - Tabs-Erklärung - Export-Funktionen - Troubleshooting

💻 Code-Dokumentation

Python-Dateien

Alle Python-Module enthalten Docstrings:

- `state_models.py`: Pydantic-Modelle mit vollständiger Dokumentation

- `neo4j_client.py` : Neo4j Client mit Methoden-Dokumentation
- `workflows/scenario_workflow.py` : LangGraph Workflow
- `agents/` : Alle Agenten mit Funktions-Dokumentation

Test-Dateien

- `test_setup.py` : Setup-Tests
- `test_workflow.py` : Workflow-Tests
- `check_setup.py` : Erweiterte Setup-Prüfung

📋 Verwendungsbeispiele

Frontend (Empfohlen)

```
streamlit run app.py
```

Siehe FRONTEND.md

Programmgesteuert

```
from neo4j_client import Neo4jClient
from workflows.scenario_workflow import ScenarioWorkflow
from state_models import ScenarioType

neo4j = Neo4jClient()
neo4j.connect()

workflow = ScenarioWorkflow(neo4j_client=neo4j, max_iterations=10)
result = workflow.generate_scenario(ScenarioType.RANSOMWARE_DOUBLE_EXTORTION)
```

Siehe README.md für weitere Beispiele.

📦 Dokumentations-Struktur

```
BA/
├── README.md           # Hauptdokumentation
├── ARCHITECTURE.md     # Architektur-Diagramme
├── STATUS.md           # Status & Capabilities
├── QUICKSTART.md       # Schnellstart
├── SETUP.md            # Setup-Anleitung
├── FRONTEND.md         # Frontend-Anleitung
├── DOCUMENTATION.md     # Diese Datei
│
├── state_models.py      # Code-Dokumentation (Docstrings)
├── neo4j_client.py     # Code-Dokumentation (Docstrings)
├── workflows/          # Workflow-Dokumentation
└── agents/             # Agenten-Dokumentation
```

🎯 Nach Anwendungsfall

Ich möchte...

- **...schnell starten**: QUICKSTART.md
- **...alles verstehen**: README.md
- **...den aktuellen Stand wissen**: STATUS.md
- **...das Frontend nutzen**: FRONTEND.md
- **...Setup-Probleme lösen**: SETUP.md
- **...Code verstehen**: Siehe Docstrings in den Python-Dateien

📞 Support

Bei Fragen oder Problemen: 1. Prüfe die entsprechende Dokumentation 2. Siehe Troubleshooting-Abschnitte 3. Prüfe `check_setup.py` für System-Status

🔄 Dokumentation aktualisieren

Diese Dokumentationen werden regelmäßig aktualisiert: - **README.md**: Bei größeren Änderungen - **STATUS.md**: Bei neuen Features oder Änderungen - **Code-Dokumentation**: Bei Code-Änderungen

Letzte Aktualisierung: 2025-01-XX

DORA-konformer Szenariengenerator MVP 1.0

Vollständige Projektdokumentation