

# AI ENTERPRISE CONFIGURATION & CODE AUDITOR

Author: © Thorsten Bylicki

Company: © BYLICKILABS

App ID / Name: ai-enterprise-configuration-code-auditor

Version: 1.0.0

---

## 1. Zweck und Mehrwert

---

Der AI Enterprise Configuration & Code Auditor ist eine serverseitige Analyse-Anwendung zur Sicherheits-, Risiko- und Konfigurationsprüfung von Software-Projekten, die als ZIP-Archiv hochgeladen werden.

Die Anwendung ist für den Einsatz in Enterprise-Workflows konzipiert und bietet:

- nachvollziehbare Analyseergebnisse (Findings und Score)
- persistente Scan-Historie (SQLite via SQLAlchemy)
- klare Trennung der Anwendungsschichten (UI, Scanner, Storage)
- vollständige Mehrsprachigkeit (Deutsch und Englisch)

---

## 2. Funktionsumfang (Ist-Stand)

---

- ZIP-Projekt-Scan über Web-UI
- Berechnung eines Risk Scores und Risk Levels  
(LOW, MEDIUM, HIGH, CRITICAL)

- Detaillierte Findings pro Datei (Dateiname, Severity, Message)
- Persistente Speicherung aller Scans in SQLite
- Mehrsprachigkeit über i18n
- Server-Side Rendering mit FastAPI und Jinja2  
(keine fragile SPA-HTML-Parsing-Logik)

---

### 3. Architektur (High-Level)

---

Die Anwendung ist vollständig serverseitig umgesetzt  
(Server Side Rendering).

Komponenten:

- FastAPI:

Webserver und API-Endpunkte

- Jinja2:

Serverseitig gerenderte UI-Templates

- SQLAlchemy:

ORM für SQLite (später optional PostgreSQL)

- Scanner-Modul:

Entpackt ZIP-Archive, analysiert Dateien und erzeugt strukturierte  
Findings

- Repository-Layer:

Zentrale Persistenzschicht für alle Datenbankzugriffe

Wichtig:

Die Benutzeroberfläche wird vollständig vom Server gerendert.

JavaScript wird ausschließlich für UI-Interaktionen eingesetzt

(z. B. Sprache, Modals).

---

## 4. Projektstruktur

---

Beispielhafte Projektstruktur:

ai-enterprise-auditor/

app/

\_\_init\_\_.py

main.py     FastAPI-Einstiegspunkt

config.py    Anwendungs-Metadaten und Standardkonfiguration

i18n.py     Serverseitiger Übersetzungs-Helper

analysis/

\_\_init\_\_.py

scanner.py   ZIP-Analyse und Sicherheits-Scans

storage/

\_\_init\_\_.py

db.py       SQLAlchemy Engine, Session und Base

models.py   ORM-Modelle (Scans, Findings, Audits)

repository.py Persistenzschicht

ui/

templates/

index.html   Haupt-Dashboard (SSR)

static/

css/

style.css

js/

i18n.js

app.js

data/

app.db      SQLite-Datenbank (automatisch erzeugt)

logs/      optionale Logdateien

requirements.txt

README.txt

---

## 5. Requirements / Abhängigkeiten

---

Die Anwendung benötigt folgende Python-Pakete:

- fastapi
- uvicorn[standard]
- jinja2
- sqlalchemy
- python-multipart (für ZIP-Uploads)

---

## 6. Installation (Windows – empfohlen)

---

### 1. Projektverzeichnis öffnen

```
cd C:\Users\...\Desktop\ai-enterprise-auditor
```

### 2. Virtuelle Umgebung erstellen

```
python -m venv .venv
```

### 3. Virtuelle Umgebung aktivieren

```
.venv\Scripts\activate
```

### 4. Abhängigkeiten installieren

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

---

### 7. Installation (Linux / macOS)

---

```
cd ai-enterprise-auditor
```

```
python3 -m venv .venv
```

```
source .venv/bin/activate
```

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

---

### 8. Start / Ausführung

---

Die Anwendung muss immer aus dem Projekt-Root gestartet werden:

```
python -m app.main
```

Nach dem Start erscheint typischerweise:

Uvicorn running on http://0.0.0.0:8000

Im Browser öffnen:

http://127.0.0.1:8000

---

## 9. Erster Scan (Web UI)

---

1. ZIP-Datei auswählen
2. Scan-Profil wählen (BASE, OWASP, CIS)
3. API Token im Dev-Mode verwenden:  
    admin-dev-token
4. Scan starten

Das Ergebnis wird serverseitig gerendert angezeigt.

---

## 10. Datenbank (SQLite)

---

Die SQLite-Datenbank wird automatisch beim ersten Start erzeugt.

Standardpfad:

data/app.db

Reset in der Entwicklungsphase:

1. Anwendung beenden (STRG + C)
2. Datenbankdatei löschen

data/app.db

### 3. Anwendung neu starten

```
python -m app.main
```

---

## 11. Konfiguration

---

Die Datenbank-URL kann über eine Umgebungsvariable gesetzt werden.

Beispiel:

```
DB_URL=sqlite:///./data/app.db
```

Optional später:

```
DB_URL=postgresql+psycopg2://user:pass@localhost:5432/auditor
```

---

## 12. Troubleshooting

---

Problem:

```
ModuleNotFoundError: fastapi
```

Lösung:

- Virtuelle Umgebung aktivieren
- pip install -r requirements.txt

Problem:

```
No module named 'app'
```

Lösung:

- Anwendung korrekt starten:

```
python -m app.main
```

Problem:

CSS oder JavaScript wird nicht geladen

Lösung:

- Requests im Terminal prüfen

- Browser Hard Reload:

STRG + SHIFT + R

Problem:

"File is not a zip file"

Lösung:

- Sicherstellen, dass eine echte ZIP-Datei hochgeladen wird

---

### 13. Roadmap (empfohlene nächste Schritte)

---

- Scan-Historie im UI (Listen- und Detailansicht)

- Severity-Filter und Suchfunktion

- JSON-Export und PDF-Reporting

- Produktive Aktivierung des Rollenmodells

(Admin, Auditor, Viewer)

---

### 14. Hinweis zur Hash-Stabilität bei GitHub-Repositories



---

Hashwerte bleiben ausschließlich dann identisch, wenn das Repository per git clone bezogen wird.

Beim Download über "Download ZIP" erstellt GitHub automatisch ein neues Root-Verzeichnis, z. B.:

<repository-name>-main

<repository-name>\_main

Der Zusatz "main" wird automatisch vergeben und verändert den Root-Verzeichnisnamen.

Technischer Hintergrund:

- Der Root-Verzeichnisname ist Teil des Dateipfads
- Dateipfade fließen in rekursive Hash-Berechnungen ein
- Eine Änderung des Root-Verzeichnisses führt zwangsläufig zu abweichenden Hashwerten

Empfehlung:

- Repository per git clone beziehen
- Hashes ausschließlich über definierte Datei-Inhalte oder Manifeste prüfen

ZIP-Downloads sollten nicht als Grundlage für Integritätsvergleiche verwendet werden.

---

15. License

---

BYLICKILABS – Internal / Enterprise Usage