Der professionelle JSON-Editor mit OSCAL-Integration

Dokumentenversion: 0.4.0
Stand: 27. April 2026
Lizenz: AGPL-3.0
OSCAL-Version: 1.2.1
MCP-Server: v2.6.0
Plattformen: Windows, Linux

Inhaltsverzeichnis

1. Zusammenfassung

mjEdit ist mehr als nur ein Editor – es ist die erste und derzeit einzige **Open Source GUI-Anwendung zur Bearbeitung von OSCAL-JSON-Dateien**, die eine professionelle und vollständig integrierte Arbeitsumgebung für Informationssicherheits-Managementsysteme (ISMS) und ganz allgemein für die Arbeit im Umfeld von Governance, Risk & Compliance bietet.

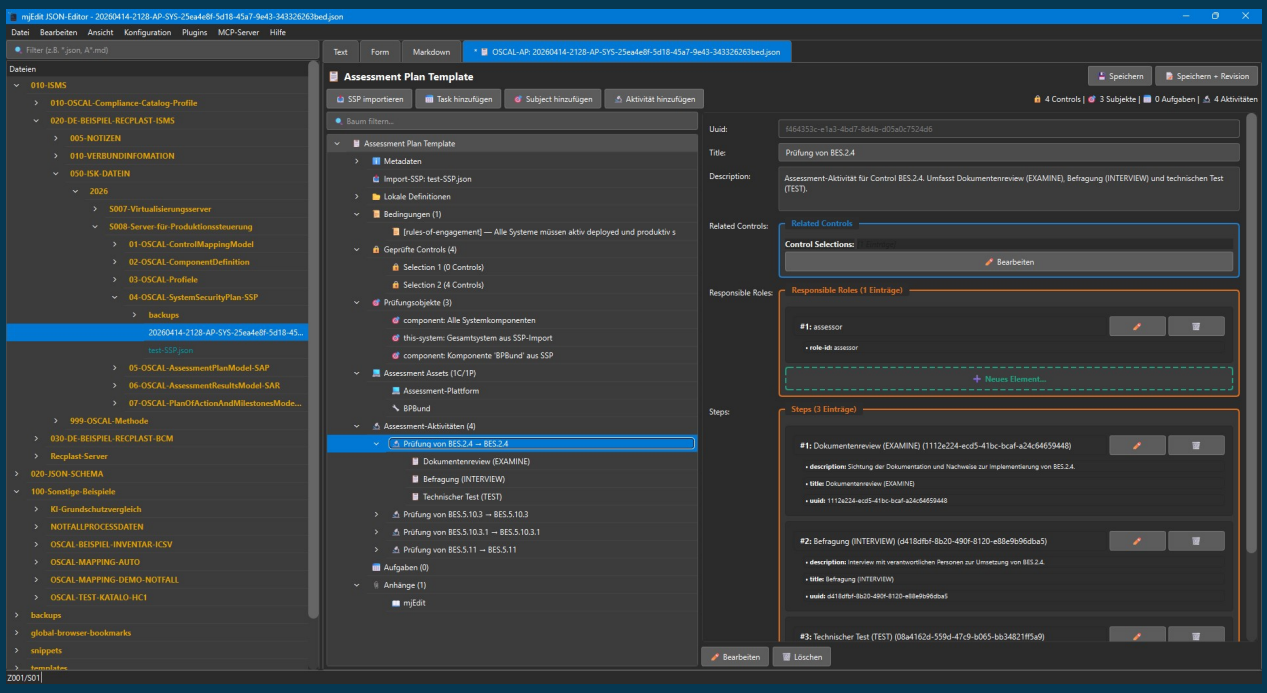

Abbildung 1: mjEdit - Bearbeitet ein OSCAL-SSP-Dokument

Für **ISMS-Bearbeiter** bedeutet das: endlich ein Werkzeug, das komplexe JSON-Strukturen nicht nur transparent darstellt, sondern auch mit dynamischen Formularen (qFORM) die tägliche Dokumentationsarbeit erleichtert. **Berater** profitieren von der Multi-Tab-Oberfläche, die es erlaubt, gleichzeitig Richtlinien in Markdown zu verfassen, Compliance-Dokumente zu prüfen und direkt im PDF-Viewer mit Annotationen oder Schwärzungen zu arbeiten. **Prüfer** wiederum erhalten eine zentrale Plattform, in der sie JSON-Daten, begleitende Dokumentationen und Webinhalte im integrierten Browser nebeneinander analysieren können – ohne Medienbrüche, ohne Tool-Wechsel.

mjEdit vereint damit die wichtigsten Werkzeuge für die tägliche Arbeit in einer Oberfläche:

- **JSON-Editor** für OSCAL und andere strukturierte Daten
- **qFORM-System** für schnelle, konsistente Datenerfassung
- **Markdown-Editor mit Live-Preview** für Richtlinien und Berichte

- **PDF-Viewer mit Annotationen und Schwärzung** für revisionssichere Dokumentation

- **Integrierter Webbrowser** für Recherche und direkte Verknüpfung von Quellen

Das Ergebnis: eine nahtlose, effiziente und professionelle Umgebung, die den gesamten Workflow von ISMS-Bearbeitung, Beratung und Prüfung abdeckt – und das als frei verfügbare Open Source Lösung.

Die Kernfunktionen auf einen Blick:

- **JSON-Editor** mit Syntax-Highlighting, Echtzeit-Validierung, Auto-Reparatur, Regex-Suche, Code-Faltung und intelligentem Performance-Management für Dateien bis 1 MB+
- **qFORM-Formularsystem** – JSON-Arrays automatisch als editierbare Formulare mit Textfeldern, Checkboxen, Dropdowns, Datumspickern und eingebetteten QC-Scripts (Python-Sandbox)
- **Markdown-Editor** mit Split-View, GitHub Flavored Markdown, erweitertem Task-System (Prioritäten, Zuweisung, Zeiterfassung) und PDF-/HTML-Export
- **PDF-Viewer** mit Annotationen (Highlighter, Notizen, Freihand), sicherer Text-Schwärzung (Redaction) und Zoom bis 400%
- **Browser-Tab** – Chromium-basierter Webbrowser mit Lesezeichen, Tabs und Lazy-Loading
- **Plugin-System** – erweiterbar über Hook-basierte Plugins mit Tab-Registrierung und Menü-Integration
- **Übergreifend:** Effizientes Tree-View mit Echtzeit-Filter, automatische Backups, Versions-Archiv, 4 Themes (Dark/Light/Blue/Green), vollständige Zweisprachigkeit (de/en)

OSCAL-Compliance – das Alleinstellungsmerkmal:

- **Alle 8 OSCAL-Dokumenttypen** (v1.2.1) vollständig unterstützt – vom Katalog bis zur Mapping-Collection, mit spezialisierten Editoren pro Dokumenttyp
- **KI-Integration mit 88 MCP-Tools** für KI-gestützte Automatisierung – OSCAL-Dokumente per KI-Agent (Claude, Cursor, VS Code Copilot, AnythingLLM) erstellen, validieren und automatisieren
- **Pydantic-basierte Validierung** – Schema-konforme Dokumente bereits bei der Erstellung
- **BSI IT-Grundschutz** und **NIST SP 800-53** vorinstalliert
- **Netzwerk-Discovery** – IP-Adressen, Hostnamen und MAC-Adressen automatisch ins SSP-Inventar übernehmen

1.1 Was mjEdit von anderen JSON-Editoren unterscheidet

Eigenschaft	Standard-JSON-Editoren	mjEdit
OSCAL-Erkennung	Manuell	Automatisch – erkennt Dokumenttyp und öffnet spezialisierten Tab
Schema-Validierung	Generisch	OSCAL v1.2.1 Schemas vorinstalliert, Echtzeit-Validierung
KI-Integration	Keine	88 MCP-Tools für vollautomatisierte OSCAL-Workflows
Compliance-Daten	Nicht vorhanden	BSI Grundschutz (2.128 Controls), NIST 800-53 (468 Controls)
Dokumentketten	Nicht möglich	Ein Klick: Profil → SSP → AP → AR → POA&M
Inventar-Management	Nicht vorhanden	In der Analogie zu OOP-Modell: Komponenten (Klassen) ↔ Inventar (Instanzen)

2. Zielgruppen und Anwendungsfälle

2.1 Informationssicherheitsbeauftragte (ISB / CISO)

Tägliche Arbeit: ISMS-Dokumentation pflegen, Controls implementieren, Audits vorbereiten.

Wie mjEdit hilft:

- Vorgefertigte BSI-Grundschutz- und NIST-Kataloge als Ausgangsbasis
- Profil-Tailoring: Nur die Controls auswählen, die Ihre Organisation betreffen
- SSP-Generierung: System-Sicherheitsplan direkt aus dem Profil ableiten
- Assessment-Planung: Prüfpläne aus dem SSP generieren
- POA&M-Tracking: Maßnahmen mit Fristen und Verantwortlichkeiten verfolgen

2.2 Compliance-Auditoren

Tägliche Arbeit: Kontrollen prüfen, Nachweise sammeln, Berichte erstellen.

Wie mjEdit hilft:

- Assessment Results direkt im Editor dokumentieren
- Findings mit Severity und Evidenz verknüpfen
- Markdown-Export für prüfbare Berichte
- Schema-Validierung garantiert normkonforme Dokumente
- Reverse-Lookup: Von einer Komponente alle zugehörigen Inventar-Items finden

2.3 IT-Architekten und Systemadministratoren

Tägliche Arbeit: Systeme dokumentieren, Netzwerk-Topologien pflegen, Patch-Management.

Wie mjEdit hilft:

- **Netzwerk-Discovery:** IP-Adressen, Hostnamen, MAC-Adressen automatisch ins SSP-Inventar
- **Komponenten-Bibliothek:** Software, Hardware, Services als wiederverwendbare Bausteine definieren
- **Inventar-Management:** Konkrete Server-Instanzen mit Komponenten verknüpfen
- **CSV-Import/Export:** Bestehendes Asset-Management-System anbinden
- **Netzwerk-Diagramme:** Automatische NWDiag-Generierung aus Inventar-Daten

2.4 DevSecOps-Teams und KI-Entwickler

Tägliche Arbeit: Security-as-Code, automatisierte Compliance-Pipelines, KI-gestützte Workflows.

Wie mjEdit hilft:

- **88 MCP-Tools** für programmatische OSCAL-Steuerung
- **Batch-Generierung:** 8 Server × komplette Dokumentkette in einem MCP-Aufruf
- **execute_steps:** Bis zu 20 Tool-Aufrufe in einem einzigen Request
- **Claude, Cursor, VS Code Copilot, AnythingLLM:** Direkte Integration in bestehende Entwicklungsumgebungen
- **AnythingLLM RAG:** Compliance-Wissensbasis mit eigenen Dokumenten aufbauen und per MCP mit mjEdit verknüpfen
- **Pydantic-Validierung:** Modell-basierte Datenintegrität für CI/CD-Pipelines

---

3. OSCAL – Warum mjEdit den Unterschied macht

3.1 Alle 8 OSCAL-Dokumenttypen

mjEdit unterstützt den vollständigen OSCAL v1.2.1 Standard:

Dokumenttyp	Kürzel	Funktion	mjEdit-Feature
Katalog	catalog	Kontrollrahmenwerk definieren	Gruppen/Controls browsen, bearbeiten, erstellen
Profil	profile	Baseline-Auswahl treffen	Controls auswählen, Tailoring, Parameter setzen
System Security Plan	ssp	System dokumentieren	Komponenten, Inventar, Control-Implementierung
Assessment Plan	ap	Prüfung planen	Prüfmethoden, Scope, Zeitrahmen definieren
Assessment Results	ar	Ergebnisse dokumentieren	Findings, Observationen, Evidenz erfassen
Plan of Action & Milestones	poam	Maßnahmen verfolgen	Remediation-Items, Fristen, Status-Tracking
Component Definition	component	Bausteine definieren	Wiederverwendbare Komponentenbibliotheken
Mapping Collection	mapping	Frameworks mappen	Bidirektionale Control-Zuordnung zwischen Standards

3.2 Das OOP-Prinzip: Komponenten als Klassen, Inventar als Instanzen

mjEdit bildet die OSCAL-Architektur intuitiv ab – angelehnt an objektorientierte Programmierung:

Komponente = Klasse (Blueprint)

```
"Apache HTTP Server" – Typ: Software, Beschreibung: Webserver
```

Inventar-Item = Instanz (konkretes Asset)

```
"Apache 2.4.58 auf Server web-01" – IP: 192.168.1.10, FQDN: web-01.example.com
```

Die Verknüpfung:

- Im SSP-Tab einen Inventar-Item-Dialog öffnen
- In Tab 3 „Komponenten" die zugehörigen Software-/Hardware-Bausteine per Multi-Select verknüpfen

- mjEdit erzeugt automatisch die `implemented-components`-Referenz mit `component-uuid`

Automatisierungs-Features:

- **Verwaiste Items erkennen:** Im SSP-Inventar-Tab werden Einträge ohne Komponentenverknüpfung automatisch markiert – so lassen sich unverknüpfte Assets per Filter anzeigen und direkt einer Komponente zuordnen
- **Reverse-Lookup:** Von einer Komponente alle konkreten Server-Instanzen anzeigen
- **Duplikatschutz:** Doppelte Verknüpfungen werden automatisch verhindert
- **CSV-Import:** Inventar-Listen aus Excel oder CMDB importieren

3.3 Automatische OSCAL-Dokumentketten

Ein zentraler Vorteil von mjEdit: Sie müssen Dokumente nicht einzeln erstellen. mjEdit kann die gesamte Compliance-Kette automatisch generieren – von einem Katalog über Profil und SSP bis hin zu Assessment Plan, Results und POA&M.

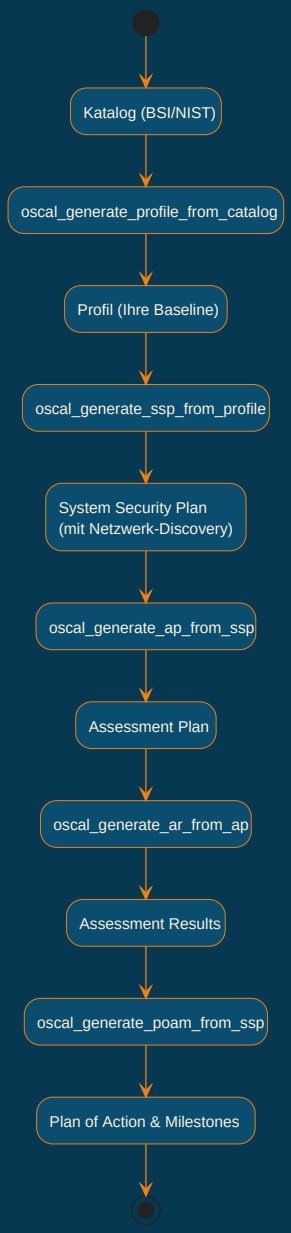

Abbildung 2: Diagramm

Jeder Schritt:

- Übernimmt UUID-Referenzen automatisch
- Setzt Metadata (Title, Version, last-modified) korrekt
- Validiert gegen OSCAL v1.2.1 Schema
- Unterstützt Custom-Prose und Assessment-Actions

3.4 Batch-Generierung für Server-Landschaften

Für Organisationen mit mehreren Systemen bietet mjEdit die **Batch-Server-Chain-Generierung**:

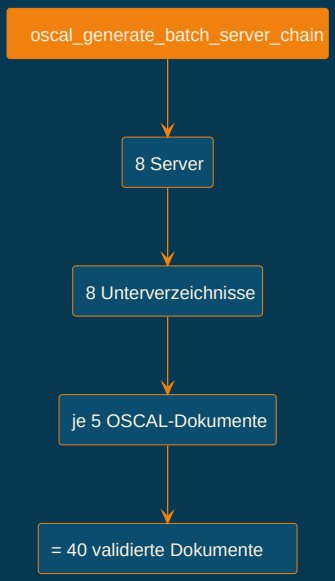

Abbildung 3: Diagramm

Jeder Server erhält:

- Eigenes Profil mit system-spezifischen Controls
- SSP mit Netzwerk-Inventar (IP, Hostname, MAC)
- Assessment Plan mit angepassten Prüfmethoden
- Assessment Results als Vorlage
- POA&M für Maßnahmen-Tracking

3.5 Evidenz-Workflow mit Status-Tracking (NEU 05/2026)

Seit dem 16.05.2026 visualisiert mjEdit den vollständigen 3-stufigen Evidenz-Workflow **SSP → AP → AR** direkt in der TreeView. Auditoren und ISMS-Verantwortliche erkennen auf einen Blick, welche Belege bereits geprüft sind und welche noch ausstehen.

Prominente Tree-Knoten in jedem Dokument:

- **SSP:** `Evidenz-Verlinkungen` pro `implemented-requirement` – welche Belege werden für diese Kontrolle gesammelt?
- **AP:** `Geplante Evidenzen` (Top-Level) – welche Belege müssen geprüft werden? Filterung über die OSCAL-Property `validation-status`.
- **AR:** `Aus AP zu prüfende Evidenzen (X/Y geprüft)` – mit Live-Status ☑ (geprüft) oder ⏳ (ausstehend) je AP-Evidenz.

Ein-Klick-Observation:

Rechtsklick auf eine ⏳-Evidenz im AR-Tab → **„📋 Observation für diese Evidenz erstellen"**. mjEdit erzeugt automatisch eine OSCAL v1.2.1 konforme Observation mit `methods=["EXAMINE"]` und `relevant-evidence[0].href = "#<uuid>"` als schemakonformer Anker auf die AP-Back-Matter-Resource. Optional kann `props.name=evidence-source` mit Wert `"ap"` als Quellangabe gesetzt werden.

Cross-Document-Linking:

Beim Öffnen eines AR-Dokuments lädt mjEdit den verknüpften Assessment Plan automatisch über `import-ap.href` und gleicht die geplanten Evidenzen gegen referenzierte Observations ab.

Nutzen:

- Lückenlose Nachvollziehbarkeit Plan → Prüfung → Beleg
- Sofortige Statusübersicht ohne manuelle Listenpflege
- 100 % OSCAL-v1.2.1-konform – exportierbar an externe GRC-Tools
- Reduziert Audit-Vorbereitungsaufwand signifikant

3.6 Profil-Tailoring und Mapping

Profil-Tailoring:

- Controls per Katalog-Baum auswählen (mit Kaskadenauswahl für ganze Gruppen)
- Parameter-Werte setzen (z.B. Passwortlänge, Audit-Intervall)
- Alterations: Controls ergänzen, einschränken oder ersetzen
- Profil-Auflösung: Flachen Katalog aus Profil + Basis-Katalog generieren

Mapping-Collection:

- Controls zwischen verschiedenen Frameworks per automatischer KI-Zuordnung Mappen (z.B. BSI ↔ NIST)
- Bidirektionaler Format-Adapter (OSCAL-NIST ↔ internes mjEdit Format mit automatischer Validierung)
- Schneller Überblick durch visuelles Highlighting verbundener Controls
- Markdown-Export der vollständigen Zuordnungstabelle

3.7 Schema-Validierung und Datenqualität

mjEdit validiert auf drei Ebenen:

Ebene	Mechanismus	Zeitpunkt
1. Pydantic-Modelle	Python-basierte Datenvalidierung	Bei Erstellung und Bearbeitung
2. JSON-Schema	NIST OSCAL v1.2.1 Schemas	Auf Wunsch oder vor dem Speichern
3. Referenz-Integrität	UUID/Href/Control-ID Prüfung	Bei Validierungs-Aufforderung

Vorinstallierte Schemas:

- oscal_catalog_schema.json
- oscal_profile_schema.json
- oscal_ssp_schema.json
- oscal_component_schema.json
- oscal_assessment-plan_schema.json
- oscal_assessment-results_schema.json
- oscal_poam_schema.json

3.8 Netzwerk-Discovery – Automatisches IT-Inventar für Ihren SSP

Das **Netzwerk-Discovery-Plugin** scannt das lokale Netzwerk per ARP-Sweep und überträgt die gefundenen Hosts als OSCAL v1.2.1-konforme `inventory-item`-Einträge direkt in einen offenen SSP-Tab. Es schließt die zeitintensivste Lücke in jedem SSP-Projekt: die manuelle Erfassung des tatsächlich vorhandenen IT-Inventars.

> **Hinweis:** Das Plugin ist standardmäßig deaktiviert und muss im Plugin-Manager bewusst aktiviert werden. Unter Windows ist zusätzlich Npcap als Treiber erforderlich.

Mehrwert nach Rolle

Rolle	Mehrwert
ISB / CISO	Aktuelles, vollständiges Inventar ohne manuelle Erfassungsaufwände; Asset-Drift wird bei jedem Scan sofort sichtbar

Rolle	Mehrwert
Compliance-Auditor	Reproduzierbarer Scan mit Zeitstempel als revisionssicherer Audit-Nachweis; direkter Vergleich zwischen gescanntem Ist-Zustand und dokumentiertem Soll-Inventar
IT-Architekt	Netzwerk-Topologie automatisch als OSCAL-Inventar – Grundlage für Netzwerk-Diagramme und Komponenten-Verknüpfungen
DevSecOps	MCP-gestützte Automatisierung: `oscal_generate_ssp_from_profile` startet optional einen Discovery-Scan und befüllt das Inventar ohne manuelle Eingaben

Wie setzt man es ein?

1. Plugin im Plugin-Manager aktivieren (*Extras → Plugin-Manager*)
2. Unter Windows: Npcap installieren (Raw-Socket-Zugriff)
3. Ein SSP-Dokument in mjEdit öffnen (der SSP-Tab muss aktiv sein)
4. Menü **Plugins** → 🔎 **Netzwerk-Discovery…** aufrufen
5. Subnet in CIDR-Notation eingeben (z. B. `192.168.1.0/24`)
6. Timeout festlegen und **Scan starten** (Dauer: ~30 Sekunden für 256 Hosts)
7. Hosts in der Ergebnistabelle aus-/abwählen; optional „Duplikate überspringen" aktivieren
8. Ziel-SSP aus dem Dropdown wählen, dann **„In SSP übernehmen"** klicken

Ergebnisse und OSCAL-Zieldokument

Die Scan-Ergebnisse landen als `inventory-item`-Einträge im Abschnitt `system-implementation.inventory-items` des aktiven **System Security Plan (SSP)**:

Scan-Ergebnis	OSCAL-Feld
IP-Adresse	`props[].name="ipv4-address"`
MAC-Adresse	`props[].name="mac-address"`
FQDN (vollständiger DNS-Name)	`props[].name="fqdn"`
Hostname (Kurzform)	`props[].name="hostname"`
Scan-Zeitstempel	`props[].name="discovered-at"`

Jeder Eintrag erhält automatisch eine RFC-4122-konforme UUID, ein leeres `description`-Feld für die manuelle Ergänzung der Asset-Funktion und eine leere `responsible-parties`-Liste für die spätere

Verantwortlichkeits-Zuordnung. Das befüllte SSP-Dokument wird als geändert markiert (`*` im Tab-Titel) und kann mit **Strg+S** gespeichert werden.

Sicherheit:

- Kein externer Dienst, keine Cloud-Anbindung – alle Daten bleiben im lokalen Netzwerk
- Subnet auf max. 4.096 IP-Adressen begrenzt (Anti-DoS-Schutz)
- Bestätigungsdialog mit Hinweis „Nur in eigenen Netzwerken scannen" vor jedem Scan

Praxis-Beispiel: Ein Berater öffnet einen vorbereiteten SSP-Tab, löst einen Scan auf `192.168.50.0/24` aus (Dauer ca. 30 Sekunden), erhält eine Liste von 47 Hosts inklusive Hostnamen aus dem internen DNS, wählt 41 produktive Systeme aus (6 Test-VMs werden ausgeschlossen) und übernimmt sie mit einem Klick in den SSP. Anschließend werden im SSP-Tab die `description`- und `responsible-parties`-Felder pro Host ergänzt – das Asset-Inventar ist in **unter 15 Minuten** aufgebaut, vollständig OSCAL-konform und auditfest.

3.9 Automatische Standard-Netzwerkarchitektur aus dem IT-Inventar

Sobald ein SSP ein befülltes `inventory-items`-Array enthält – sei es durch manuellen Import, CSV-Import oder den Netzwerk-Discovery-Scan – kann mjEdit automatisch ein **NWDiag-Netzwerkdiagramm** generieren. Das Diagramm zeigt die erfassten Assets in einem standardisierten, dokumentationsreifen Layer-3-Topologie-Format und wird direkt als OSCAL-konformes Artefakt in das SSP-Dokument eingebettet.

Rolle	Mehrwert

Mehrwert nach Rolle

Rolle	Mehrwert
ISB / CISO	Visuelle Systemgrenzen-Übersicht ohne Zusatz-Tools; immer synchron mit dem aktuellen SSP-Inventar – keine veralteten Visio-Dokumente mehr
Compliance-Auditor	Netzwerkdiagramm als OSCAL-konformer Nachweis der erfassten Systemgrenzen (Authorization Boundary) im SSP; direkt aus den Primärdaten abgeleitet, nicht manuell erstellt
IT-Architekt	Automatisch aus OSCAL-Daten abgeleitete Basisdokumentation als Ausgangspunkt für detailliertere Architekturdiagramme; Subnetz-Gruppierung und Komponenten-Typen werden automatisch erkannt
DevSecOps	Diagramm-Generierung über MCP-Tool `oscal_generate_ssp_from_profile` oder direkt per MCP-Aufruf für Pipeline-Integration; SVG-Export für CI/CD-Dokumentations-Pipelines

Wie setzt man es ein?

Die Netzwerkarchitektur wird direkt aus dem geöffneten SSP-Tab erzeugt:

1. SSP mit befülltem Inventar im SSP-Tab öffnen
2. Im SSP-Tab → Kontextmenü oder Toolbar-Schaltfläche **„Netzwerkdiagramm generieren"** auswählen
3. mjEdit liest alle `inventory-items` mit IP-Adressen und gruppiert sie automatisch nach Subnetz
4. Ein NWDiag-Skript wird erzeugt und als Diagramm gerendert
5. Ausgabeformat wählen: SVG, PNG oder direkte Einbettung in OSCAL Back-Matter

Ergebnisse und OSCAL-Zieldokument

Das generierte Diagramm wird in den **System Security Plan (SSP)** eingebettet:

Artefakt	OSCAL-Ziel
NWDiag-Quellcode	`back-matter.resources[]` mit `media-type: "text/plain"`
Diagramm-Bild (SVG)	`back-matter.resources[]` mit `media-type: "image/svg+xml"`
Diagramm-Bild (PNG)	`back-matter.resources[]` mit `media-type: "image/png"`
Diagramm-Referenz	`system-characteristics.network-architecture.diagrams[]`

Was wird automatisch erkannt und dargestellt?

Erkanntes Element	Datenquelle im OSCAL-Inventar
Subnetze / Netzwerksegmente	IP-Adressen der `inventory-items` (Eigenschaft `ipv4-address`)
Hosts	Alle `inventory-items` mit gesetzter IP-Adresse
Hostnamen / Beschriftungen	Eigenschaft `hostname` oder `fqdn`
Komponenten-Typ	Mit dem `inventory-item` verknüpfte Komponente (Server, Switch, Firewall etc.)
Netzwerk-Segmente	Automatische Gruppierung nach /24-Subnetz

Praxis-Beispiel

Ein SSP enthält nach dem Netzwerk-Discovery-Scan 47 Hosts in zwei Subnetzen (`10.0.1.0/24` Server-Segment, `10.0.2.0/24` Client-Segment). Mit einem Klick generiert mjEdit ein Diagramm, das:

- beide Segmente als NWDiag-Netzwerkgruppen darstellt,
- Hosts mit ihren Hostnamen auflistet,
- die zugeordneten Komponenten-Typen (z. B. „Ubuntu 24.04", „Windows 11") annotiert.

Das Ergebnis: Eine vollständige, aktuelle und OSCAL-konforme Netzwerkarchitektur-Dokumentation – automatisch aus den Daten, die ohnehin im SSP stehen.

> **Hinweis:** Das Diagramm zeigt eine vereinfachte Layer-3-Topologie (Subnetz-Sicht). Für vollständige Netzwerkdiagramme mit VLAN-Konfiguration, Firewall-Regeln und Protokoll-Details empfehlen wir dedizierte Netzwerkmanagement-Tools als Ergänzung.

4. KI-Integration per MCP – Ihre Compliance auf Autopilot

> **WICHTIGER HINWEIS**: Die MCP-Integration befindet sich in einem frühen Entwicklungsstadium. Aktuell sind 88 Tools implementiert, die eine breite Palette von Funktionen abdecken – von der Dokumentenerstellung über Validierung bis hin zur GUI-Steuerung. In den kommenden Monaten werden weitere Tools und Ressourcen hinzugefügt oder können auch einfach entfernt werden, um die Automatisierungsmöglichkeiten kontinuierlich zu verbessern. Zur Zeit ist es nicht empfehlenswert auf diesen Entwicklungsstand eine produktive Integration vorzunehmen. Die MCP-Tools sind bereits voll funktionsfähig und können in Testumgebungen mit unterstützten KI-Agenten (Claude Desktop, Cursor, VS Code Copilot, AnythingLLM) ausprobiert werden. Bitte beachten Sie die Dokumentation und die Tool-Beschreibungen für Details zu den verfügbaren Funktionen und deren Anwendungsfällen.

4.1 Was ist MCP?

Das **Model Context Protocol (MCP)** ist ein offener Standard, der es KI-Systemen ermöglicht, mit externen Tools zu kommunizieren. mjEdit implementiert einen vollständigen MCP-Server mit zwei Transport-Optionen:

- **STDIO:** Direkte Prozess-Kommunikation (für Claude Desktop, Cursor, VS Code Copilot)
- **SSE:** HTTP Server-Sent Events auf localhost (für AnythingLLM Docker, eigene Integrationen)

4.2 88 MCP-Tools in 15 Kategorien

mjEdit hebt sich als integrierte Arbeitsumgebung deutlich von klassischen Editoren ab – und das nicht nur durch seine einzigartige Spezialisierung auf **OSCAL-JSON-Dokumente**. Mit der Unterstützung von **88 MCP-Tools in 15 Kategorien** stellt mjEdit eine Funktionsvielfalt bereit, die weit über das reine Bearbeiten von Dateien hinausgeht.

Diese Tools sind nicht nur technische Helfer, sondern bilden die Grundlage für eine **KI-gestützte Arbeitsweise**, die ISMS-Bearbeiter, Berater und Prüfer direkt in ihrem täglichen Workflow unterstützt. Ob es um das schnelle Erstellen und Validieren von OSCAL-Dokumenten geht, die Transformation von JSON-Strukturen, die interaktive Bearbeitung von Markdown-Richtlinien oder die revisionssichere Kommentierung von PDF-Dokumenten – mjEdit vereint alle Schritte in einer Oberfläche.

Besonders hervorzuheben sind die **geführten Workflows mit 15 MCP-Prompts**, die komplexe Aufgaben wie die Erstellung kompletter OSCAL-Dokumentketten, die Durchführung von Compliance-Checks oder die interaktive Tailoring-Bearbeitung von Profilen vereinfachen. Damit wird mjEdit zu einem ech-

ten **Produktivitäts-Hub**, der nicht nur Zeit spart, sondern auch die Qualität und Konsistenz der Arbeit sicherstellt.

Mit dieser Kombination aus **88 spezialisierten Tools** und **15 intelligenten Prompts** bietet mjEdit eine einzigartige Plattform, die den gesamten Lebenszyklus von Compliance-Dokumenten abdeckt – von der ersten Erstellung über die Validierung bis hin zur finalen Prüfung:

Datei-Operationen (12 Tools)

Tool	Beschreibung
`file_create`	Datei erstellen (mit Batch-Support für große Dateien)
`file_open`	Datei öffnen mit automatischem Tab-Routing
`file_read`	Datei lesen ohne Editor-Öffnung
`file_write`	Datei direkt auf Disk schreiben
`file_save`	Aktive Datei speichern mit Backup
`file_save_as`	Unter neuem Pfad speichern
`file_close`	Aktive Datei schließen
`file_append`	Inkrementelles Schreiben für große Dateien
`file_list`	Dateien mit Glob-Pattern auflisten
`file_get_active`	Pfad + Inhalt der aktuellen Datei
`dir_create`	Verzeichnis erstellen
`dir_rename`	Verzeichnis umbenennen

JSON-Operationen (6 Tools)

Tool	Beschreibung
`json_get_content`	JSON-Inhalt des Editors abrufen
`json_set_content`	JSON-Inhalt setzen (mit Validierung)
`json_format`	Pretty-Print mit konfigurierbarem Indent
`json_validate`	Syntax-Validierung mit Zeilen-/Spaltenangabe
`json_export`	Export als XML oder CSV
`json_query`	Werte per Punkt-Pfad abfragen (z.B. `metadata.title`)

Content-Manipulation (3 Tools)

Tool	Beschreibung
`content_search`	Text/Regex-Suche mit Kontext-Zeilen
`content_replace`	Text-Ersetzung mit Regex und Capture-Groups

Tool	Beschreibung
`content_insert_at`	Inhalt an bestimmter Zeile einfügen

Markdown-Operationen (8 Tools)

Tool	Beschreibung
`markdown_get_content`	Markdown-Inhalt abrufen
`markdown_set_content`	Markdown-Inhalt setzen
`markdown_insert_snippet`	Snippet an Cursor-Position einfügen
`markdown_replace_section`	Abschnitt nach Überschrift ersetzen
`markdown_get_toc`	Inhaltsverzeichnis generieren
`markdown_export_to_html`	HTML-Export
`markdown_export_to_pdf`	PDF-Export
`markdown_format`	Markdown formatieren

qFORM-Formulare (9 Tools)

Tool	Beschreibung
`qform_create`	Neues Formular mit Feldtypen erstellen
`qform_get_data`	Strukturierte Formulardaten abrufen
`qform_set_field`	Feldwert setzen (per Label oder Index)
`qform_add_field`	Feld mit Positionskontrolle hinzufügen
`qform_remove_field`	Feld entfernen
`qform_render`	Formular im GUI rendern
`qform_execute_qc_script`	QC-Script in Sandbox ausführen
`qform_create_qc_script`	QC-Script im CRC32:Base64-Format erstellen
`qform_list_fields`	Alle Felder mit Typen und Werten auflisten

Validierung (5 Tools)

Tool	Beschreibung
`validate_json_schema`	JSON gegen Schema validieren
`validate_oscal_document`	Gegen offizielles OSCAL-Schema prüfen
`validate_oscal_references`	UUID/Href/Control-ID-Referenzen prüfen
`validate_qform`	qFORM-Struktur und Konsistenz prüfen
`validate_file_integrity`	JSON-Syntax, Encoding, Dateigröße prüfen

OSCAL-Erstellung (8 Tools)

Tool	Beschreibung
`oscal_create_catalog`	Katalog mit Auto-UUID/Metadata erstellen
`oscal_create_profile`	Profil mit Import-Href erstellen
`oscal_create_ssp`	SSP mit System-Charakteristiken erstellen
`oscal_create_poam`	POA&M mit Platzhalter-Items erstellen
`oscal_create_assessment_plan`	Assessment Plan mit Include-All erstellen
`oscal_create_assessment_results`	Assessment Results erstellen
`oscal_create_component_definition`	Komponenten-Definition erstellen
`oscal_detect_type`	OSCAL-Dokumenttyp automatisch erkennen

OSCAL-Abfragen (8 Tools)

Tool	Beschreibung
`oscal_list_controls`	Alle Controls mit IDs, Titeln, Klassen auflisten
`oscal_get_control`	Einzelnes Control mit allen Details abrufen
`oscal_search`	OSCAL-Dokument durchsuchen (Text + Regex)
`oscal_get_metadata`	Metadata eines Dokuments abrufen
`oscal_list_groups`	Katalog-Gruppen auflisten
`oscal_get_statistics`	Statistiken (Control-Anzahl etc.) abrufen
`oscal_select_controls`	Controls im GUI-Tab per ID selektieren
`oscal_select_group`	Ganze Gruppe im GUI selektieren

OSCAL-Bearbeitung (7 Tools)

Tool	Beschreibung
`oscal_add_control`	Control zu Katalog hinzufügen
`oscal_update_metadata`	Metadata-Felder aktualisieren
`oscal_add_property`	Property via JSON-Pfad hinzufügen
`oscal_add_parameter`	Set-Parameter hinzufügen
`oscal_add_back_matter_resource`	Back-Matter-Resource hinzufügen
`oscal_update_implementation_status`	Control-Status im SSP setzen
`oscal_add_role`	Rolle zum Dokument hinzufügen

OSCAL-Control-Verwaltung (9 Tools)

Tool	Beschreibung
`oscal_update_control`	Control aktualisieren (Titel, Klasse, Parts, Props)
`oscal_remove_control`	Control aus Katalog entfernen
`oscal_add_group`	Gruppe zu Katalog hinzufügen
`oscal_remove_group`	Gruppe mit allen Controls entfernen
`oscal_profile_list_imports`	Profil-Imports mit Include/Exclude auflisten
`oscal_profile_add_import`	Import zu Profil hinzufügen
`oscal_profile_remove_import`	Import nach Href entfernen
`oscal_profile_add_alter`	Alteration zu Control hinzufügen (Tailoring)
`oscal_profile_set_parameter`	Parameter-Wert im Modify-Abschnitt setzen

OSCAL-Generierung (8 Tools)

Tool	Beschreibung
`oscal_generate_profile_from_catalog`	Profil aus Katalog mit ausgewählten Controls
`oscal_generate_ssp_from_profile`	SSP mit Netzwerk-Discovery-Support
`oscal_generate_ap_from_ssp`	Assessment Plan aus SSP generieren
`oscal_generate_poam_from_ssp`	POA&M aus SSP generieren
`oscal_generate_ar_from_ap`	Assessment Results aus Plan generieren
`oscal_generate_full_chain`	Komplette Kette ohne Tailoring
`oscal_generate_tailored_chain`	Kette mit Assessment-Actions/Custom-Prose
`oscal_generate_batch_server_chain`	Batch-Generierung für mehrere Server

OSCAL-Modell-API (7 Tools – Pydantic-basiert)

Tool	Beschreibung
`oscal_model_create`	OSCAL über Pydantic-Modell mit Template-Support
`oscal_model_open`	OSCAL-Datei im passenden Tab öffnen
`oscal_model_delete`	OSCAL löschen (mit .bak-Backup)
`oscal_model_modify`	Batch-Operationen (set/append/remove/replace)
`oscal_model_save`	Modell speichern + GUI-Refresh
`oscal_model_query`	Abfragen aus Modell (paginiert)
`oscal_model_list_templates`	Verfügbare OSCAL-Templates auflisten

Editor-Operationen (8 Tools)

Tool	Beschreibung
`editor_undo`	Rückgängig
`editor_redo`	Wiederherstellen
`editor_cut`	Ausschneiden
`editor_copy`	Kopieren
`editor_paste`	Einfügen
`editor_select_all`	Alles auswählen
`editor_find_replace`	Suchen & Ersetzen Dialog
`editor_go_to_line`	Zu Zeile navigieren

Template-Verwaltung (5 Tools)

Tool	Beschreibung
`template_list`	Templates auflisten
`template_create`	Template erstellen
`template_delete`	Template löschen
`template_export`	Template exportieren
`template_import`	Template importieren

GUI-Steuerung (8 Tools)

Tool	Beschreibung
`execute_steps`	**Batch-Tool:** Bis zu 20 Tool-Aufrufe in EINEM Request
`get_help`	Vollständige Tool-Liste + Workflows (zweisprachig)
`gui_show_message`	Nachricht anzeigen (info/warning/error)
`gui_show_tab`	Tab wechseln (Index oder Name)
`gui_get_status`	Editor-Status + Tool-Liste mit Beispielen
`gui_set_theme`	Theme setzen (dark/light/blue/green)
`gui_list_tabs`	Alle offenen Tabs mit Typ/Path auflisten
`gui_get_tab_content`	Inhalt eines spezifischen Tabs lesen

4.3 22 MCP-Ressourcen

KI-Agenten können kontextuelle Informationen direkt abfragen:

Ressource	Beschreibung
`mjedit://config/app`	Aktuelle Anwendungs-Konfiguration
`mjedit://config/keybindings`	Konfigurierte Tastenkürzel
`mjedit://config/security`	Sicherheitseinstellungen
`mjedit://config/mcp`	MCP-Server-Konfiguration
`mjedit://schema/oscal/catalog`	OSCAL Catalog Schema v1.2.1
`mjedit://schema/oscal/profile`	OSCAL Profile Schema v1.2.1
`mjedit://schema/oscal/ssp`	OSCAL SSP Schema v1.2.1
`mjedit://schema/oscal/component`	OSCAL Component Schema v1.2.1
`mjedit://schema/oscal/assessment-plan`	OSCAL Assessment Plan Schema v1.2.1
`mjedit://schema/oscal/assessment-results`	OSCAL Assessment Results Schema v1.2.1
`mjedit://schema/oscal/poam`	OSCAL POA&M Schema v1.2.1
`mjedit://snippets/json`	JSON-Code-Snippets
`mjedit://snippets/markdown`	Markdown-Snippets
`mjedit://snippets/oscal`	OSCAL-Snippets
`mjedit://snippets/qform`	qFORM-Snippets
`mjedit://snippets/qscript`	QC-Script-Vorlagen
`mjedit://templates/list`	Alle verfügbaren Templates
`mjedit://editor/active-file`	Pfad + Inhalt der aktuellen Datei
`mjedit://editor/active-tab`	Aktiver Tab (Typ, Index, Datei)
`mjedit://plugins/list`	Aktive Plugins
`mjedit://oscal/controls`	Controls des aktuellen Dokuments
`mjedit://oscal/required-fields`	Pflicht-/Optionalfelder pro OSCAL-Typ

4.4 15 MCP-Prompts für geführte Workflows

Prompt	Beschreibung
`oscal_create_catalog_prompt`	Katalog-Erstellung mit Assistenz
`oscal_create_ssp_prompt`	SSP-Erstellung mit System-Definition
`oscal_validate_prompt`	Interaktive OSCAL-Validierung
`oscal_compliance_check_prompt`	Compliance-Prüfung gegen Framework
`oscal_edit_controls_prompt`	Interaktive Control-Bearbeitung
`oscal_profile_tailoring_prompt`	Geführtes Profil-Tailoring
`oscal_batch_edit_prompt`	Batch-Bearbeitung von Controls
`oscal_network_discovery_prompt`	SSP Netzwerk-Discovery
`oscal_batch_server_chain_prompt`	Batch-Dokumentketten für Server
`oscal_migration_prompt`	OSCAL-Versions-Migration
`qform_design_prompt`	Formular-Design per KI

Prompt	Beschreibung
`qform_qc_script_prompt`	QC-Script-Erstellung per KI
`json_transform_prompt`	JSON-Transformation
`markdown_document_prompt`	Markdown-Dokument-Erstellung
`security_audit_prompt`	Sicherheitsanalyse

4.5 AnythingLLM – Der KI-Agent mit RAG-Wissensbasis für Compliance-Profis

Während Claude Desktop, Cursor und VS Code Copilot als Cloud-basierte KI-Agenten leistungsfähige MCP-Integration bieten, nimmt **AnythingLLM** eine besondere Rolle ein: Als selbst gehostete, Open-Source-Plattform kombiniert AnythingLLM die Fähigkeiten eines KI-Assistenten mit einer **Retrieval-Augmented Generation (RAG)**-Wissensbasis – und das vollständig unter Ihrer Kontrolle, wahlweise lokal oder im eigenen Docker-Container.

Was ist AnythingLLM?

AnythingLLM ist eine All-in-One KI-Plattform, die folgende Kernfähigkeiten vereint:

Fähigkeit	Beschreibung
Multi-LLM-Support	OpenAI GPT-4o, Anthropic Claude, lokale Modelle (Ollama, LM Studio), Azure OpenAI und weitere
RAG-Engine	Eigene Dokumente (PDF, DOCX, TXT, JSON, Markdown) als durchsuchbare Wissensbasis einbetten
Vektor-Datenbank	LanceDB (eingebettet), ChromaDB, Pinecone, Weaviate, Qdrant und weitere
MCP-Client	Vollständige MCP-Unterstützung via SSE – direkte Anbindung an mjEdit
Workspace-Konzept	Isolierte Arbeitsbereiche mit eigenen Dokumenten, Modellen und Agenten
Agenten-System	KI-Agenten mit Tool-Zugriff, Web-Suche und Datei-Operationen
Self-Hosted	Lokal, Docker oder eigener Server – keine Cloud-Abhängigkeit, volle Datensouveränität

RAG – Ihre eigene Compliance-Wissensbasis

Retrieval-Augmented Generation (RAG) ist der entscheidende Vorteil von AnythingLLM gegenüber reinen Chat-Assistenten. Statt die KI nur mit allgemeinem Wissen arbeiten zu lassen, füttern Sie sie mit **Ihren eigenen Compliance-Dokumenten**:

Was Sie einbetten können:

- BSI IT-Grundschutz-Kompendium (PDF, 800+ Seiten)
- NIST SP 800-53 Kontrollen-Katalog
- Ihre bestehenden ISMS-Richtlinien und Verfahrensanweisungen
- Audit-Berichte vergangener Jahre
- Interne Sicherheitsrichtlinien und Betriebshandbücher
- OSCAL-Dokumente (JSON) aus mjEdit
- Technische Dokumentation Ihrer IT-Infrastruktur

Wie RAG funktioniert:

1. **Einbetten (Embedding):** Sie laden Ihre Dokumente in einen AnythingLLM-Workspace. Die Texte werden in semantische Vektoren zerlegt und in einer lokalen Vektor-Datenbank gespeichert.
2. **Abfragen (Retrieval):** Wenn Sie eine Frage stellen, sucht AnythingLLM die relevantesten Textpassagen aus Ihren Dokumenten.
3. **Generieren (Generation):** Das LLM beantwortet Ihre Frage auf Basis der gefundenen Passagen – mit **Quellenverweisen** zu den Originaldokumenten.

Ergebnis: Die KI kennt Ihre Organisation, Ihre Richtlinien und Ihre Infrastruktur – und kann diese Informationen direkt in mjEdit-Workflows einbringen.

MCP-Integration: AnythingLLM + mjEdit

Die Verbindung zwischen AnythingLLM und mjEdit erfolgt über den **SSE-Transport** des MCP-Servers:

Konfiguration in AnythingLLM:

```
{
  "mcpServers": {
    "mjEdit": {
      "url": "http://localhost:8765/sse",
      "transport": "sse"
    }
  }
}
```

Nach der Verbindung stehen dem AnythingLLM-Agenten alle **88 MCP-Tools** von mjEdit zur Verfügung –

kombiniert mit dem RAG-Wissen aus Ihren eigenen Dokumenten.

Die Synergie:

Komponente	Beitrag
AnythingLLM RAG	Kennt Ihre Richtlinien, Standards, Audit-Berichte und Infrastruktur
AnythingLLM Agent	Interpretiert Anfragen, plant Schritte, ruft Tools auf
mjEdit MCP-Tools	Erstellt, validiert und bearbeitet OSCAL-Dokumente
mjEdit GUI	Zeigt Ergebnisse in spezialisierten Tabs an

> **Hinweis:** Die nachfolgenden MCP-Anwendungsfälle sind nur im Ansatz getestet und stellen ein Zielbild dar, durch die aktive Entwicklung der MCP-API können sich die Anwendungsfälle noch änderen oder im Workflow erheblich abweichen bzw. erweitern. Die Anwendungsfälle bilden legendlich das Potential von KI in der aktiven Interaktion mit mjEdit ab.

Interaktion zwischen MCP-HOST und MCP-SERVER (mjEdit): OSCAL-Dokumentkette per KI erstellen

Szenario: Sie betreiben 8 Linux-Webserver und möchten für jeden eine vollständige Compliance-Dokumentation erstellen.

Schritt 1: In Claude Desktop, VS Code Copilot oder AnythingLLM:

> „Erstelle für meine 8 Webserver eine vollständige OSCAL-Dokumentkette auf Basis des BSI-IT-Grundschutz-Katalogs. Jeder Server hat folgende Netzwerk-Daten: …"

Was passiert:

1. KI ruft per MCP das TOOL: `oscal_generate_batch_server_chain` auf und übergibt an den MCP-SERVER die nötigen Daten
2. Der MCP-SERVER erstellt für jeden Server ein Unterverzeichnis
3. Der MCP-Server generiert die OSCAl-Dokumente: Profil → SSP (mit Inventar) → AP → AR → POA&M
4. Netzwerk-Discovery befüllt Inventar automatisch, sofern mjEdit Zugriff auf die Server hat (alternativ können die Daten auch per RAG aus Dokumenten extrahiert werden)
5. Alle Dokumente werden gegen OSCAL v1.2.1 Schema validiert

Ergebnis: 40 validierte OSCAL-Dokumente in einem Aufruf.

Use Case 1 – RAG-gestützte SSP-Erstellung aus bestehender Dokumentation

Szenario: Ihre Organisation hat bereits umfangreiche Sicherheitsdokumentation in Word und PDF – aber noch keine OSCAL-Dokumente. Sie möchten einen System Security Plan (SSP) erstellen, der auf Ihren bestehenden Richtlinien basiert.

Vorbereitung:

- Laden Sie Ihre Sicherheitsrichtlinien, Netzwerk-Dokumentation und Betriebshandbücher in einen AnythingLLM-Workspace
- Verbinden Sie AnythingLLM per SSE mit mjEdit

Prompt in AnythingLLM:

> „Erstelle einen OSCAL System Security Plan für unseren Webserver ‚web-prod-01'. Nutze die Informationen aus unserer Sicherheitsrichtlinie und dem Netzwerkplan, um die System-Charakteristiken, Komponenten und das Inventar zu befüllen. Verwende den BSI-Grundschutz als Baseline."

Was passiert:

1. AnythingLLM durchsucht per RAG Ihre Dokumente nach relevanten Informationen (IP-Adressen, installierte Software, Verantwortliche, Sicherheitsmaßnahmen)
2. Der Agent ruft `oscal_generate_ssp_from_profile` auf und befüllt die System-Charakteristiken mit den gefundenen Daten
3. Komponenten und Inventar-Items werden aus Ihrer technischen Dokumentation extrahiert
4. Die Control-Implementierungen werden mit Textpassagen aus Ihren Richtlinien vorausgefüllt
5. mjEdit öffnet den fertigen SSP im spezialisierten SSP-Tab zur Überprüfung

Ergebnis: Ein SSP, der nicht generisch ist, sondern Ihre tatsächliche Infrastruktur und Ihre realen Sicherheitsmaßnahmen widerspiegelt – mit nachvollziehbaren Quellenverweisen.

Use Case 2 – Intelligente Audit-Vorbereitung mit Wissensbasis

Szenario: Ein BSI-Audit steht an. Sie müssen nachweisen, dass alle relevanten Bausteine des IT-Grundschutz-Kompendiums umgesetzt sind. Bisherige Audit-Berichte und Maßnahmen-Dokumentationen liegen in verschiedenen Formaten vor.

Vorbereitung:

- Laden Sie das BSI-Grundschutz-Kompendium, vergangene Audit-Berichte und Ihre Maßnahmen-Dokumentation in AnythingLLM
- Bestehende OSCAL-Dokumente aus mjEdit als JSON-Dateien einbetten

Prompt in AnythingLLM:

> „Prüfe unseren aktuellen SSP gegen den BSI-Grundschutz-Katalog. Identifiziere Controls, die laut unserem letzten Audit-Bericht als ‚teilweise umgesetzt' markiert waren. Erstelle einen Assessment Plan für diese Controls und generiere ein POA&M mit den offenen Maßnahmen aus dem Audit-Bericht."

Was passiert:

1. RAG findet die relevanten Passagen aus dem letzten Audit-Bericht (Findings, Empfehlungen, Fristen)
2. Agent ruft `oscal_list_controls` auf, um den aktuellen SSP-Status abzugleichen
3. `oscal_generate_ap_from_ssp` erstellt einen fokussierten Assessment Plan für die identifizierten Controls
4. `oscal_generate_poam_from_ssp` erzeugt ein POA&M mit konkreten Maßnahmen, Fristen und Verantwortlichen – extrahiert aus dem Audit-Bericht
5. Die Dokumente werden in mjEdit geöffnet, wo Sie Details nachbearbeiten können

Ergebnis: Statt Wochen manueller Arbeit entsteht in Minuten ein konsistenter Audit-Vorbereitungs-Satz – basierend auf Ihren realen Daten.

Use Case 3 – Compliance-Chatbot für das gesamte Team

Szenario: Ihr ISMS-Team hat 12 Mitarbeiter, die regelmäßig Fragen zu Richtlinien, Controls und Zuständigkeiten haben. Bisher müssen sie in verschiedenen Dokumenten suchen oder den ISB fragen.

Vorbereitung:

- Richten Sie AnythingLLM als Docker-Container im internen Netzwerk ein
- Laden Sie alle ISMS-Dokumente, OSCAL-Kataloge, Richtlinien und Verfahrensanweisungen ein
- Konfigurieren Sie die MCP-Verbindung zu mjEdit

Typische Fragen des Teams:

> „Welche Controls aus dem BSI-Grundschutz betreffen unsere Datenbank-Server?"

→ RAG durchsucht den Katalog und Ihren SSP, listet relevante Bausteine mit Implementierungsstatus auf.

> „Wer ist verantwortlich für die Umsetzung von OPS.1.1.3 und bis wann muss die Maßnahme abgeschlossen sein?"

→ RAG findet die Zuordnung im SSP und die Frist im POA&M, der Agent ruft `oscal_get_control` und `oscal_get_metadata` auf, um aktuelle Daten aus mjEdit zu liefern.

> „Erstelle eine Zusammenfassung aller offenen Maßnahmen mit Fälligkeit in den nächsten 30 Tagen."

→ Agent nutzt `oscal_search` im POA&M, RAG ergänzt Kontext aus vergangenen Berichten, Ergebnis wird als Markdown-Bericht in mjEdit erstellt.

Ergebnis: Ein internes Compliance-Wissensportal, das Fragen in Sekunden beantwortet – mit Quellenverweisen zu den Originaldokumenten und direktem Zugriff auf aktuelle OSCAL-Daten in mjEdit.

Vergleich: AnythingLLM vs. Cloud-KI-Agenten

Kriterium	Cloud-Agenten (Claude, Copilot)	AnythingLLM (Self-Hosted)
Datensouveränität	Daten verlassen das Unternehmen	Vollständig lokal / on-premise
RAG-Wissensbasis	Begrenzt (Kontext-Fenster)	Unbegrenzt (Vektor-Datenbank)
Eigene Dokumente	Upload pro Konversation	Persistent eingebettet
MCP-Transport	STDIO	SSE (HTTP)
Kosten	API-Gebühren pro Token	Einmalig (Hardware) + optionale API
Offline-Fähigkeit	Nein (Internet erforderlich)	Ja (mit lokalen Modellen via Ollama)
Multi-User	Einzelnutzer-Kontext	Team-Workspaces mit Rechteverwaltung
Modellauswahl	Anbieter-gebunden	Frei wählbar (OpenAI, Anthropic, lokal)

Empfehlung: Für Organisationen mit sensiblen Compliance-Daten, die keine Informationen an Cloud-Dienste übermitteln dürfen, ist AnythingLLM die ideale Wahl. Die RAG-Wissensbasis hebt die Qualität der KI-Ausgaben auf ein neues Niveau, da die KI mit Ihrem spezifischen Organisationswissen arbeitet – nicht nur mit allgemeinem Training.

5. Die Tabs im Detail – Alle Ansichten, Funktionen und Anwendungsfälle

mjEdit organisiert die Arbeit in einem Multi-Tab-System. Jeder Tab ist auf einen bestimmten Arbeitsbereich spezialisiert und bietet eigene Werkzeuge, Kontextmenüs und Tastenkürzel. Die Tabs arbeiten nahtlos zusammen: Änderungen im Text-Tab spiegeln sich im Formular-Tab, OSCAL-Tabs generieren Markdown-Berichte, und der PDF-Tab zeigt exportierte Dokumente direkt an.

5.1 TEXT-TAB (Tab 0) – Der JSON-Editor

Der Text-Tab ist das Herzstück von mjEdit: ein vollwertiger JSON-Editor mit Syntax-Highlighting, Echtzeit-Validierung, Klammer-Matching und intelligenter Auto-Reparatur.

Funktionsübersicht

Funktion	Detail
Syntax-Highlighting	Farbcodierung für Keys (lila), Strings (grün), Zahlen (blau), Klammern (orange)
Echtzeit-Validierung	JSON-Syntax wird bei jeder Änderung geprüft, Fehlerposition (Zeile/Spalte) in Statusleiste
Klammer-Matching	Passende `{}` und `[]` werden beim Cursor-Setzen farblich hervorgehoben
Auto-Reparatur	Fehlende Kommas, unbeendete Strings und ungültige Escape-Sequenzen automatisch korrigieren
Rechtschreibprüfung	Wellenlinie unter falsch geschriebenen Wörtern, anpassbares Wörterbuch
Zeilennummern	Klickbare Zeilennummern mit Navigation
Code-Faltung	Verschachtelte JSON-Strukturen ein-/ausklappen
Auto-Einrückung	Intelligente Einrückung bei verschachtelten Strukturen
Undo/Redo	Unbegrenzter Verlauf für alle Änderungen
Suchen & Ersetzen	Regex-Unterstützung, Live-Highlighting aller Treffer, Vorwärts/Rückwärts-Navigation
Schriftart-Persistierung	Gewählte Schriftart und -größe wird zwischen Sitzungen gespeichert
Encoding-Erkennung	Automatische Erkennung von UTF-8, Latin-1, BOM-Markern
JSON-Formatierung	Pretty-Print mit Strg+Alt+J, konfigurierbare Einrückung
Schema-Validierung	Gegen OSCAL v1.2.1 oder benutzerdefinierte JSON-Schemas

Performance-Modi für große Dateien

mjEdit passt den Editor automatisch an die Dateigröße an:

Dateigröße	Verhalten
< 100 KB	Vollständiges Highlighting + Rechtschreibprüfung + alle Features
100 KB – 200 KB	Vereinfachtes Highlighting (nur Keys/Klammern), keine Rechtschreibprüfung
> 1 MB	Kein Highlighting, Lazy-Loading (erste 10.000 Zeilen)

Tastenkürzel im Text-Tab

Kürzel	Aktion
Strg+Alt+J	JSON formatieren (Pretty-Print)
Strg+F	Suchen
Strg+H	Suchen & Ersetzen
F3 / Umschalt+F3	Nächster / Vorheriger Treffer
Strg+Umschalt+H	Alle ersetzen
Strg+Z / Strg+Y	Rückgängig / Wiederherstellen
Strg+A	Alles auswählen
Strg+<	Snippet einfügen

Kontextmenü (Rechtsklick)

- Ausschneiden, Kopieren, Einfügen
- Snippet einfügen (aus Snippet-Bibliothek)
- Template einfügen
- Drucken

Use Cases

Use Case 1 – Große Konfigurationsdateien bearbeiten: Sie öffnen eine umfangreiche Infrastruktur-Konfiguration (2 MB JSON). mjEdit aktiviert automatisch den Performance-Modus, zeigt das JSON strukturiert an und ermöglicht die Suche nach spezifischen Schlüsseln. Die Echtzeit-Validierung stellt sicher, dass Ihre Änderungen die JSON-Struktur nicht zerstören.

> **Hinweis:** OSCAL-Dateien (Katalog, Profil, SSP etc.) werden von mjEdit automatisch erkannt und im spezialisierten OSCAL-Tab geöffnet – nicht im Text-Tab. Für die Bearbeitung von OSCAL-Dokumenten siehe OSCAL-TABS.

Use Case 2 – Fehlerhafte JSON-Datei reparieren: Sie erhalten eine JSON-Datei von einem Kollegen, die

Syntaxfehler enthält. Beim Öffnen erkennt mjEdit die Fehler und zeigt die Position an. Per „JSON reparieren"-Dialog werden fehlende Kommas, unbeendete Strings und ungültige Escapes automatisch korrigiert.

Use Case 3 – Konfigurationsdatei bearbeiten: Sie pflegen eine `config.json` für Ihr Projekt. Das Klammer-Matching zeigt Ihnen immer den korrekten Kontext, die Rechtschreibprüfung fängt Tippfehler in Strings ab, und das Snippet-System fügt häufig verwendete Strukturen per Strg+< ein.

Mehrwert: Im Vergleich zu einem normalen Texteditor bietet der Text-Tab eine JSON-spezifische Arbeitsumgebung mit aktiver Fehlervermeidung. Die Auto-Reparatur und Echtzeit-Validierung sparen besonders bei großen Compliance-Dateien erhebliche Zeit bei der Fehlersuche.

5.2 FORM-TAB (Tab 1) – qFORM Formularsystem & QC-Scripts

Der Form-Tab verwandelt JSON-Daten automatisch in bearbeitbare Formulare. Das proprietäre mjEdit qFORM-Format ermöglicht es, strukturierte Dateneingabe mit eingebetteten Python-Scripts (QC-Scripts) zu kombinieren – ideal für Prüfprotokolle, Checklisten, dynamische Berichte oder zur strukturierten Datenerfassung.

qFORM-Erkennung

JSON-Arrays mit einem speziellen Header werden automatisch als Formulare dargestellt:

```
[
  {"type": "qFORM", "version": "1.0", "last_scroll_position": 0.0},
  {"label": "Projektname", "value": "ISMS-2026"},
  {"label": "Datum", "value": "2026-04-18"},
  {"label": "prn_qc_berechnung", "value": "@(len(json_data.get('controls',[])))@"}
]
```

Feldtypen im Formular – Automatische Widget-Erkennung

mjEdit erkennt den passenden Widget-Typ automatisch anhand des JSON-Werts. Die Erkennung erfolgt in einer festen Prioritätsreihenfolge:

1. Verschachteltes Objekt → Gruppierung (QGroupBox)

```
{"kontakt": {"name": "Max Mustermann", "email": "max@example.com", "telefon": "+49 123 456"}}
```

→ Erzeugt eine **eingebettete Feldgruppe** mit Rahmen. Tiefe Verschachtelungen erhalten abgestufte Rahmenfarben.

2. Array → Listencontainer (QGroupBox mit orangem Rahmen)

```
{"notizen": ["Erste Notiz", "Zweite Notiz", "Dritte Notiz"]}
```

→ Erzeugt eine **Liste mit orangem Rahmen** und einem „Neues Element…"-Button zum Hinzufügen.

3. QC-Script → Code-Feld (QTextEdit, readonly, lila Hintergrund)

```
{"prn_qc_berechnung": "QC<Base64-kodierter-Python-Code-mit-CRC32>"}
```

→ Der Wert beginnt mit `"QC"` und ist länger als 10 Zeichen. Wird als **lila hinterlegtes, nicht editierbares Feld** mit dem Hinweis „F12-Code-Feld!" dargestellt. Per F12 wird der eingebettete Python-Code in der Sandbox ausgeführt.

4. Boolean → Checkbox (QCheckBox)

```
{"benachrichtigungen": true}
{"auto_save": false}
```

→ Python-`bool`-Werte (`true` / `false`) werden als **Checkbox** dargestellt. Kein String – der JSON-Typ muss `boolean` sein.

5. Dropdown → Auswahlliste (QComboBox) – Pipe-Syntax

```
{"status": "|Offen|In Bearbeitung|Abgeschlossen|"}
{"prioritaet": "|Niedrig|Mittel|Hoch|Kritisch|"}
{"risiko_level": "|Gering|Mittel|Hoch|Sehr Hoch|Kritisch|"}
```

→ Erkennung: String **beginnt mit** `|` und enthält **mindestens 2** Pipe-Zeichen. Die Optionen zwischen den Pipes werden als **Dropdown-Menü** dargestellt. Die erste Option ist vorausgewählt. Beim Speichern wird nur der gewählte Wert (ohne Pipes) zurückgeschrieben.

6. Dateipfad → Pfadfeld (QLineEdit + Browse-Button)

```
{"projekt_pfad": "C:/Users/dev/project"}
{"backup_ordner": "./backups/daily"}
{"netzwerk_share": "\\\\server\\freigabe\\daten"}
```

→ Erkennung: Windows-Laufwerk (`C:/...`), UNC-Pfad (`\\server\...`), Unix-absolut (`/path/...`) oder relativ (`./...`, `../...`). Erzeugt ein **Textfeld mit** 🗀 **Browse-Button** und einer „Relativer Pfad"-Checkbox.

7. Datum/Zeit → Datumsfeld (QLineEdit + DATUM-Button mit Kalender)

```
{"erstelldatum": "2025-11-29"}
{"ablaufdatum": "15.01.2024"}
{"created": "2024-01-15T10:30:00"}
{"geburtstag": "01/15/2024"}
```

→ **Zwei Erkennungswege:**

a) Schlüssel-basiert – Einer dieser Keywords im Feldnamen: `date`, `datum`, `day`, `tag`, `time`, `zeit`, `created`, `modified`, `updated`, `born`, `geburtstag`, `ablauf`, `expiry`, `gueltig`, `valid`

b) Wert-basiert – Eines dieser Datumsformate im Wert:

- `YYYY-MM-DD` (ISO 8601)
- `DD.MM.YYYY` (deutsch)
- `MM/DD/YYYY` (US-Format)
- `YYYY-MM-DDTHH:MM:SS` (ISO mit Zeitkomponente)

→ Erzeugt ein **Textfeld mit „DATUM"-Button**, der ein Kalender-Widget (QCalendarWidget) als Popup öffnet. Das gewählte Datum wird im erkannten Format zurückgeschrieben.

8. Standard → Auto-Resize-Textfeld (AutoResizeTextEdit)

```
{"titel": "Beispiel-Formular"}
{"beschreibung": "Langer Text mit mehreren Zeilen..."}
{"speicher_intervall_minuten": 5}
```

→ Alle übrigen Werte (Strings, Zahlen) werden als **automatisch wachsendes Textfeld** dargestellt. Der Original-Datentyp (`int`, `float`, `bool`, `null`) wird beim Speichern korrekt zurück-konvertiert.

Zusammenfassung der Feldtypen

Feldtyp	Widget	JSON-Erkennung	GUI-Element
Verschachteltes Objekt	QGroupBox	Wert ist `{}` (Dict)	Feldgruppe mit Rahmen
Array/Liste	QGroupBox (orange)	Wert ist `[]` (Array)	Liste + „Neues Element"-Button
QC-Script	QTextEdit (readonly)	Wert beginnt mit `"QC"`, >10 Zeichen	Lila Feld, F12 zum Ausführen
Checkbox	QCheckBox	Wert ist `true` oder `false`	Anklickbare Checkbox
Dropdown	QComboBox	Wert beginnt mit `|`, ≥2 Pipes	Aufklappbare Auswahlliste
Dateipfad	QLineEdit + 🗀	Wert enthält Pfad-Muster	Textfeld + Browse-Button
Datum	QLineEdit + DATUM	Key enthält Datums-Keyword oder Wert ist Datumsformat	Textfeld + Kalender-Popup
Text (Standard)	AutoResizeTextEdit	Alles andere	Automatisch wachsendes Textfeld

Feld-Präfix-System

Das Präfix-System steuert, wie Felder behandelt und gedruckt werden:

Präfix	Typ	Funktion
`prn_`	Druckbar	Feld erscheint im Druckausgabe
`prn_qc_*`	QC-Script	Python-Code mit Text-Ausgabe, ausführbar per F12
`prn_qr_*`	QR-Code	Feldwert wird als QR-Code dargestellt
`prn_qr_qc_*`	QR + Script	Python-Script mit QR-Code-Ausgabe
`_comment_`	Kommentar	Nur-Lese-Hinweistext im Formular
(kein Präfix)	Standard	Normales Feld, nicht in Druckausgabe

QC-Scripts – Eingebettete Python-Berechnungen

QC-Scripts sind Python-Codeblöcke, die sicher in einer RestrictedPython-Sandbox ausgeführt werden. Sie ermöglichen dynamische Berechnungen direkt im Formular.

Syntax: `@(python_code)@`

Verfügbare Variablen:

- `json_data` – Dict aller Formularfelder (Key-Value)
- `json_data_list` – Original-qFORM-Liste mit allen Metadaten

Erlaubte Operationen:

- String-Manipulation (`str`, `len`, `split`, `join`)
- Mathematik (`int`, `float`, `abs`, `round`, `sum`)
- Listen/Dict-Operationen (`list`, `dict`, `sorted`, `enumerate`)
- Datumsformatierung (`datetime`, `date`, `time`, `timedelta`)
- Reguläre Ausdrücke (`re`)
- JSON-Verarbeitung (`json`)
- **REST-API / Netzwerkzugriffe** (`urllib.request`, `requests`)

Erlaubte Module (via `import` in der Sandbox):

Modul	Beschreibung
`datetime`	Datum/Zeit-Operationen
`json`	JSON serialisieren/deserialisieren
`math`	Mathematische Funktionen
`re`	Reguläre Ausdrücke
`time`	Zeitfunktionen
`urllib.request`	HTTP-Anfragen (GET, POST)
`urllib.parse`	URL-Encoding/Decoding
`requests`	HTTP-Client (wenn installiert)
`socket`	Netzwerk-Sockets

Blockierte Operationen (Sicherheit):

- Dateisystem-Zugriff (`os`, `sys`, `shutil`, `subprocess`)
- Dynamisches Code-Laden (`importlib`, `ctypes`, `pickle`)
- Zugriff auf private Attribute (`__dunder__`-Methoden)
- Nicht explizit freigegebene Module

Beispiel 1 – Lokale Berechnung:

```
# Zählt implementierte Controls in einem SSP
@(
count = 0
for comp in json_data.get('implemented-requirements', []):
    if comp.get('status') == 'implemented':
        count += 1
f"Implementiert: {count} von {len(json_data.get('implemented-requirements', []))}"
)@
```

Beispiel 2 – REST-API-Abruf (Daten aus externer Quelle):

```
# Ruft aktuelle CVE-Daten für eine Komponente ab
@(
import urllib.request
import json

url = f"https://api.example.com/cve?product={json_data.get('product_name','')}"
response = urllib.request.urlopen(url)
data = json.loads(response.read().decode())
f"Offene CVEs: {len(data.get('vulnerabilities', []))}"
)@
```

Platzhalter-System (F11)

Platzhalter in der Form `<feldname>` werden dynamisch mit Werten aus anderen Feldern befüllt:

- **Einfache Platzhalter:** `<project_name>` → Wert des Feldes „project_name"
- **Verschachtelte Pfade:** `<metadata.title>` → Verschachtelte JSON-Pfade
- **Array-Zugriff:** `<controls[0].id>` → Erstes Element eines Arrays
- **Sichere Auswertung:** Keine Code-Injection möglich

Formular-Besonderheiten

Feature	Vorteil
Bidirektionale Synchronisation	Änderungen im Formular aktualisieren den JSON-Text und umgekehrt
Inkrementelles Rendering	Große Formulare (100+ Felder) werden schrittweise aufgebaut
Scroll-Erhaltung	Die letzte Scroll-Position wird bei Tab-Wechsel beibehalten
ISO 8601 Lokalisierung	Datumsfelder zeigen das Format der Systemsprache
Auto-Fokus	Tab-Reihenfolge folgt der JSON-Struktur

Tastenkürzel im Form-Tab

Kürzel	Aktion
F11	Platzhalter mit Werten aus anderen Feldern ersetzen
F12	QC-Script im aktuellen Feld ausführen
Tab / Umschalt+Tab	Zwischen Formularfeldern navigieren
Strg+S	Formular speichern (synchronisiert zurück zu JSON)

Kontextmenü (Rechtsklick im Formular)

- JSON-Element löschen (mit Bestätigungsdialog)
- Kopieren / Einfügen ganzer Strukturen

Use Cases

Use Case 1 – Prüfprotokoll für Audit erstellen: Sie erstellen ein qFORM-Formular als Prüfprotokoll mit Feldern für Prüfer, Datum, System-Name und Checklisten-Einträgen. Per QC-Script berechnet F12 automatisch die Erfüllungsquote: „43 von 50 Controls implementiert (86%)". Das Formular wird als PDF exportiert – fertig ist der Prüfbericht.

Use Case 2 – Dynamischer Compliance-Bericht: Ein qFORM-Formular enthält Platzhalter wie `<system_name>` und `<responsible_person>`. Per F11 werden diese aus einer JSON-Datenbasis befüllt. QC-Scripts berechnen KPIs (Compliance-Rate, offene Maßnahmen, Risiko-Score) und generieren QR-Codes mit Links zu Detailberichten.

Use Case 3 – Inventar-Erfassung: Techniker erfassen Server-Daten in einem qFORM-Formular: Hostname, IP-Adresse, Software-Versionen, Verantwortlicher. Die Daten werden als JSON gespeichert und können direkt als OSCAL-Inventar-Items in einen SSP importiert werden.

Use Case 4 – Checkliste mit automatischer Auswertung: Eine BSI-Grundschutz-Checkliste als qFORM: Jeder Baustein hat ein Checkbox-Feld (umgesetzt/nicht umgesetzt). Ein QC-Script am Ende zählt automatisch die erledigten Bausteine, berechnet den Reifegrad und erzeugt einen Ampel-Status (rot/gelb/grün).

Mehrwert: Das qFORM-System verbindet strukturierte Dateneingabe mit programmierbarer Logik. Statt separate Tools für Formulare, Berechnungen und Berichterstellung zu verwenden, erledigt ein einziges qFORM-Dokument alles – von der Eingabe über die Berechnung bis zum fertigen Bericht.

5.3 MARKDOWN-TAB (Tab 2) – Dokumentation und Berichte

Der Markdown-Tab bietet einen vollwertigen Markdown-Editor mit Live-Preview, erweitertem Task-Listen-System und direkter Integration in die OSCAL-Dokumentation.

Funktionsübersicht

Funktion	Detail
Split-View	Editor links, Live-Preview rechts – Änderungen sofort sichtbar
GitHub Flavored Markdown	Tabellen, Task-Listen, Fußnoten, Code-Blöcke mit Syntax-Highlighting
Formatierungs-Shortcuts	Fett, Kursiv, Code, Durchgestrichen, Listen, Links, Bilder per Tastenkürzel
Rechtschreibprüfung	Dasselbe System wie im Text-Tab mit Wellenlinien-Markierung
Auto-Formatierung	mdformat-Integration für konsistente Markdown-Formatierung
HTML-Export	Markdown in HTML-Datei konvertieren
PDF-Export	Markdown als PDF mit konfigurierbarem Layout (Titel, Autor, Header/Footer, Font, Farb-Theme)
Separate Dateiverwaltung	Markdown-Dateien werden unabhängig von JSON verwaltet
Auto-Backup	Automatisches Backup beim Speichern

Erweitertes Task-Listen-System

mjEdit erweitert Standard-Markdown-Task-Listen um ein professionelles Projektmanagement-System:

Element	Syntax	Beispiel
Checkbox	`- [ ]` / `- [x]`	`- [x] Server konfiguriert`
Priorität	Schlüsselwörter	`kritisch`, `wichtig`, `hoch`, `normal`, `niedrig`
Status	Schlüsselwörter	`in arbeit`, `pausiert`, `entscheidung`, `bewertung`, `vollständig`
Zuweisung	`@Person`	`@Mueller`
Datum	`YYYY-MM-DD`	`2026-04-30`
Tags	`#Tag`	`#sicherheit #patch`
Interne Links	`[[Dokument]]`	`[[SSP-Server-01]]`
Zeiterfassung	`Est:` / `Actual:`	`Est: 4h`, `Actual: 3h`
Bug-Tracking	`Bug:` / `Fixed:`	`Bug: SEC-2024-001`, `Fixed: 2026-04-15`
OSCAL-Links	`&OSCAL|...|...`	Direktlink zu OSCAL-Controls
POA&M-Referenzen	`&POAM|...|...`	Direktlink zu Maßnahmen

Tastenkürzel im Markdown-Tab

Kürzel	Aktion
Strg+B	**Fett**
Strg+I	*Kursiv*
Strg+Umschalt+D	Code-Block (“‘)
Strg+Umschalt+X	~~Durchgestrichen~~
Strg+Umschalt+U	Ungeordnete Liste
Strg+Umschalt+L	Geordnete Liste
Strg+Umschalt+I	Bild einfügen
Strg+Umschalt+K	Link einfügen
Strg+H	Suchen & Ersetzen
Strg+S	Speichern

Use Cases

Use Case 1 – Audit-Bericht erstellen: Nach einem Assessment exportieren Sie die Ergebnisse aus dem OSCAL Assessment Results-Tab als Markdown. Im Markdown-Tab ergänzen Sie Bewertungen, fügen Tabellen mit Risiko-Scores ein und erstellen Task-Listen für offene Maßnahmen. Der fertige Bericht wird als PDF mit Firmen-Header und -Footer exportiert.

Use Case 2 – Projekt-Tracking mit OSCAL-Integration: Sie führen ein Compliance-Projekt und nutzen die erweiterten Task-Listen: Jede Aufgabe hat eine Priorität, einen Verantwortlichen (@Mueller), ein Fälligkeitsdatum (📅 2026-05-01) und einen direkten Link zum OSCAL-Control (&OSCAL). Die Zeiterfassung (`Est: 4h`, `Actual: 3h`) ermöglicht Aufwands-Controlling.

Use Case 3 – Technische Dokumentation: Sie dokumentieren die System-Architektur in Markdown: Code-Blöcke für Konfigurationsbeispiele, Tabellen für Netzwerk-Übersichten, und verschachtelte Listen für Abhängigkeiten. Die Live-Preview zeigt das formatierte Ergebnis sofort, und per HTML-Export entsteht eine browserfähige Dokumentation.

Mehrwert: Der Markdown-Tab ist nicht nur ein Texteditor – er ist ein Dokumentations-Hub mit direkter OSCAL-Integration. Die erweiterten Task-Listen mit Zuweisung, Zeiterfassung und OSCAL/POA&M-Referenzen machen ihn zur zentralen Stelle für Projekt-Tracking und Berichtserstellung.

5.4 PDF-TAB (Tab 3) – Viewer, Annotator und Schwärzungswerkzeug

Der PDF-Tab zeigt PDF-Dokumente direkt in mjEdit an – ohne externes Programm. Er unterstützt Multi-Tab-Anzeige, Annotationen, Text-Suche und sichere Schwärzung sensibler Inhalte.

Funktionsübersicht

Funktion	Detail
Multi-Tab PDF-Anzeige	Mehrere PDF-Dokumente gleichzeitig in separaten Unter-Tabs
Lazy-Loading	PDF-Engine wird erst beim Öffnen des ersten PDFs geladen – spart Startup-Zeit
Zoom-Steuerung	25% – 400%, Fit-to-Width, Fit-to-Height
Seiten-Navigation	Seitenweise Blättern, Direkt-Sprung zu Seitennummer
Text-Suche	Volltextsuche innerhalb des PDFs mit Treffer-Highlighting
Highlighter	Farbige Textmarkierungen (mehrere Farben)
Notizen/Kommentare	Anmerkungen an beliebigen Stellen im Dokument
Freihand-Zeichnung	Markierungen direkt auf dem PDF
Formen	Rechtecke und Kreise zum Hervorheben
Text einfügen	Zusätzlichen Text auf das PDF setzen
Schwärzung (Redaction)	Sichere Schwärzung mit Unicode ■ – Inhalte werden tatsächlich entfernt, nicht nur überdeckt
PDF speichern	PDF mit allen Annotationen als neue Datei speichern
Drucken	PDF direkt aus dem Tab drucken
Tab-Schließen	Einzelne PDF-Tabs per Kontextmenü oder Kürzel schließen

Tastenkürzel im PDF-Tab

Kürzel	Aktion
Strg+F	Suche im PDF
Strg+P	PDF drucken
Strg+S	PDF mit Annotationen speichern
+ / -	Hineinzoomen / Herauszoomen
Bild↑ / Bild↓	Seite vor / zurück
Alt+Q	PDF-Tab schließen

Use Cases

Use Case 1 – Audit-Nachweise prüfen: Während eines Compliance-Audits erhalten Sie Nachweisdokumente als PDF. Sie öffnen mehrere PDFs gleichzeitig in separaten Tabs, markieren relevante Passagen mit dem Highlighter und fügen Notizen mit Bewertungen hinzu. Die annotierten PDFs werden gespeichert und als Evidenz im OSCAL Assessment Results referenziert.

Use Case 2 – Vertrauliche Dokumente schwärzen: Vor der Weitergabe eines SSP-Berichts müssen IP-Adressen und Zugangsdaten geschwärzt werden. Die Schwärzungsfunktion nutzt Unicode-Blockzeichen (■),

die den Originaltext tatsächlich ersetzen – im Gegensatz zu vielen PDF-Editoren, die nur schwarze Rechtecke überlagern und deren Inhalte extrahiert werden können.

Use Case 3 – Compliance-Berichte reviewen: Sie generieren einen OSCAL-Compliance-Bericht als PDF und öffnen ihn direkt im PDF-Tab. Mit Freihand-Zeichnungen markieren Sie Abschnitte, die überarbeitet werden müssen, und fügen Textnotizen mit Korrekturhinweisen ein.

Mehrwert: Der PDF-Tab spart den ständigen Wechsel zwischen mjEdit und einem externen PDF-Viewer. Besonders die sichere Schwärzungsfunktion ist für Compliance-Arbeiten essenziell: Sie garantiert, dass vertrauliche Daten nicht nur visuell überdeckt, sondern tatsächlich aus dem Dokument entfernt werden.

5.5 BROWSER-TAB (Plugin) – Integrierter Webbrowser

Der Browser-Tab integriert einen vollwertigen Chromium-basierten Webbrowser direkt in mjEdit. Damit entfällt der Kontextwechsel beim Nachschlagen von OSCAL-Referenzen, NIST-Dokumentationen oder BSI-Grundschutz-Kompendien.

Funktionsübersicht

Funktion	Detail
Chromium-Engine	Vollständiger Webbrowser basierend auf QWebEngineView
Lazy-Loading	Browser-Engine wird erst beim ersten Aufruf geladen – spart 200+ MB RAM beim Start
URL-Navigation	Adressleiste mit Eingabe und Enter-Navigation
Vor/Zurück/Reload/Home	Standard-Browser-Navigation
Mehrere Browser-Tabs	Innerhalb des Browser-Tabs weitere Unter-Tabs öffnen
HTTPS-Status	Grünes/rotes Schloss-Symbol zeigt Verbindungssicherheit
Zoom-Steuerung	75% – 300% Zoom
Lesezeichen	Zweistufiges System: Globale (Team) + private (persönliche) Lesezeichen mit Tag-basierter Organisation
Konfigurierbare Startseite	Standard-Homepage in Config einstellbar
Cookie-Consent	Automatische Akzeptierung (konfigurierbar)
Externe URL-Integration	Links aus Markdown-Dokumenten öffnen automatisch im Browser-Tab
HTTPS-Warnung	Warnung bei unsicheren HTTP-Verbindungen

GPU-Stabilitäts-Optimierungen

Der Browser-Tab verwendet spezielle Flags für maximale Stabilität:

- Deaktivierte GPU-Sandbox für Kompatibilität mit virtualisierten Umgebungen

- Angepasstes Rendering für Terminal-Server und Remote-Desktop

Tastenkürzel im Browser-Tab

Kürzel	Aktion
Strg+Tab	Nächster Browser-Unter-Tab
Strg+Umschalt+Tab	Vorheriger Browser-Unter-Tab
Strg+W	Browser-Unter-Tab schließen

Lesezeichen-System: Globale und private Lesezeichen

Der Browser-Tab verfügt über ein **zweistufiges Lesezeichen-System**, das zwischen Team-weiten und persönlichen Bookmarks unterscheidet – ein Konzept, das es in Standard-Browsern nicht gibt.

Globale Lesezeichen sind für das gesamte Team gedacht: Eine gemeinsame Sammlung von Compliance-Referenzen, OSCAL-Dokumentationen und Audit-Ressourcen, die alle Anwender nutzen. Die Datei `global-browser-bookmarks.json` liegt im `data/`-Verzeichnis und kann über Versionskontrolle (Git) im Team geteilt werden.

Private Lesezeichen sind persönliche Bookmarks eines einzelnen Nutzers. Sie werden in `config/browser-privat-bookmarks.j` gespeichert und automatisch mit dem Tag `privat` markiert. Über die „Quick Lesezeichen"-Funktion lassen sie sich mit einem einzigen Klick anlegen – ohne Dialog, ohne Eingabe.

Eigenschaft	Globale Lesezeichen	Private Lesezeichen
Zweck	Team-Ressourcen, geteilte Referenzen	Persönliche Sammlung
Speicherort	`data/global-browser-bookmarks/`	`config/`
Hinzufügen	Dialog mit Titel, Bemerkung, Tags, Kürzel	Ein Klick – kein Dialog nötig
Typischer Inhalt	BSI-Kompendium, NIST-Referenz, ISO-Standards	Eigene Recherche-Links, Notizen
Team-Sharing	Über Git teilbar	Nur lokal

Lesezeichen hinzufügen – drei Wege:

Alle Lesezeichen-Funktionen sind über das Kontextmenü erreichbar (Rechtsklick auf die Webseite oder die URL-Leiste):

Kontextmenü-Option	Funktion
⭐ **Zu globalen Lesezeichen hinzufügen**	Öffnet einen Dialog mit Titel (vorausgefüllt), URL, Bemerkungsfeld, Tag-Eingabe und Benutzer-Kürzel
📌 **Quick Lesezeichen**	Speichert die aktuelle Seite **sofort** als privates Lesezeichen – Tags werden automatisch aus dem Seitentitel extrahiert
📑 **Webseite aus Lesezeichen öffnen**	Öffnet den Lesezeichen-Manager mit allen gespeicherten Bookmarks

Tag-basierte Organisation:

Statt starrer Ordner-Hierarchien setzt mjEdit auf ein flexibles **Tag-System**. Jedes Lesezeichen kann beliebig viele Tags erhalten – dargestellt als farbige Chips mit Lösch-Button. Die Vorteile:

- Ein Lesezeichen kann mehreren Kategorien gleichzeitig zugeordnet werden (z. B. `oscal`, `nist`, `ssp`)
- Tags werden beim Quick-Lesezeichen automatisch aus dem Seitentitel generiert (Stoppwörter in Deutsch und Englisch werden gefiltert, mindestens 4 Zeichen, maximal 5 Tags)
- Im Lesezeichen-Manager filtert die Live-Suche über Titel, Tags, Bemerkung, Ersteller und Datum

Lesezeichen-Manager:

Der Lesezeichen-Manager (📑 „Webseite aus Lesezeichen öffnen") zeigt alle globalen und privaten Bookmarks in einer Tabelle:

Spalte	Inhalt
Titel	Seitentitel (Doppelklick öffnet die URL im Browser)
Bemerkung	Optionale Beschreibung des Lesezeichens
Tags	Farbige Tag-Chips zur Kategorisierung
Erstellt	Datum, Uhrzeit und Benutzer-Kürzel
Aktionen	✏ Bearbeiten, 🗑 Löschen

Eine Live-Filter-Eingabe über der Tabelle durchsucht alle Felder in Echtzeit – ideal, um bei einer wachsenden Sammlung schnell die richtige Referenz zu finden.

Praxis-Beispiel:

Ein ISB-Team arbeitet gemeinsam an der OSCAL-Dokumentation. Die globalen Lesezeichen enthalten:

- BSI IT-Grundschutz-Kompendium (Tags: `bsi`, `grundschutz`)
- NIST OSCAL Reference (Tags: `nist`, `oscal`, `referenz`)
- NIST SP 800-53 Controls (Tags: `nist`, `controls`, `800-53`)
- ISO 27001:2022 Annex A (Tags: `iso`, `27001`, `annex-a`)

Jedes Teammitglied hat zusätzlich private Lesezeichen für eigene Recherche – z. B. CVE-Einträge, Herstellerdokumentationen oder interne Wiki-Seiten. Die private Sammlung bleibt auf dem eigenen Rechner, die globale wird über Git synchronisiert.

Use Cases

Use Case 1 – OSCAL-Referenz nachschlagen: Während Sie einen SSP bearbeiten, müssen Sie die genaue Struktur eines OSCAL-Elements prüfen. Statt den Browser zu wechseln, öffnen Sie die NIST OSCAL-Referenz (pages.nist.gov/OSCAL-Reference/) direkt im Browser-Tab. Die Lesezeichen-Funktion speichert häufig genutzte Referenzseiten.

Use Case 2 – BSI-Grundschutz-Kompendium einsehen: Sie implementieren BSI-Controls in einem Katalog und benötigen die Originaltexte aus dem BSI-Kompendium. Der Browser-Tab zeigt die BSI-Website neben Ihrer Arbeit – kein Kontextwechsel, kein Fensterwechsel.

Use Case 3 – Schulungsmaterialien während der Arbeit: Neue Mitarbeitende können OSCAL-Tutorials und Dokumentationen im Browser-Tab lesen, während sie gleichzeitig in den anderen Tabs die ersten eigenen OSCAL-Dokumente erstellen – alles in einer Anwendung.

Use Case 4 – Markdown-Links direkt öffnen: Ihre Markdown-Dokumentation enthält Links zu externen Ressourcen (z.B. CVE-Datenbank, NIST NVD). Ein Klick auf den Link öffnet die Seite automatisch im Browser-Tab, statt ein externes Programm zu starten.

Webseiten als OSCAL Back-Matter-Resource speichern

Eine Besonderheit des Browser-Tabs ist die direkte Integration mit dem OSCAL-System: Jede angezeigte Webseite kann per Rechtsklick als **Back-Matter-Resource** in ein geöffnetes OSCAL-Dokument eingefügt werden. OSCAL verwendet Back-Matter als standardisierten Anhang für Referenzen, Nachweise und externe Quellen – und mjEdit macht das Anlegen dieser Referenzen so einfach wie einen Rechtsklick.

Zwei Varianten im Kontextmenü:

Menüpunkt	Beschreibung	Voraussetzung
Als OSCAL-Anhang erstellen	Erstellt eine Back-Matter-Resource mit URL-Verweis, Titel und Beschreibung	OSCAL-Plugin ist aktiv
Neue OSCAL-Resource erstellen	Erweiterte Variante mit Anmerkungen (Remarks), Media-Type und Quell-URL	Mindestens ein OSCAL-Tab ist geöffnet

So funktioniert es:

1. **Webseite öffnen** – Navigieren Sie im Browser-Tab zur gewünschten Quelle (z. B. BSI-Grundschutz-Kompendium, CVE-Datenbank, NIST-Referenz)
2. **Rechtsklick → Resource erstellen** – Der Dialog öffnet sich mit vorausgefüllter URL und Seitentitel
3. **Titel und Beschreibung ergänzen** – Optional: Anmerkungen, warum diese Quelle relevant ist
4. **OSCAL-Tab auswählen** – Bei mehreren geöffneten OSCAL-Dokumenten wählen Sie das Ziel-Dokument
5. **Einfügen** – Die Resource wird automatisch in die `back-matter.resources`-Liste des OSCAL-Dokuments eingefügt

Was wird in der OSCAL-Datei gespeichert:

```
{
  "uuid": "auto-generierte-uuid-v4",
  "title": "BSI IT-Grundschutz-Kompendium 2024",
  "description": "Referenz für Baustein SYS.1.1",
  "props": [
    {"name": "created-by", "value": "mjEdit"},
    {"name": "created-at", "value": "2026-04-19T14:30:00+02:00"},
    {"name": "source-url", "value": "https://www.bsi.bund.de/..."}
  ],
  "rlinks": [
    {"href": "https://www.bsi.bund.de/...", "media-type": "text/html"}
  ],
  "remarks": "Grundlage für Control-Implementierung SYS.1.1"
}
```

Unterstützte OSCAL-Dokumenttypen: Die Einfügung funktioniert in **allen 7 OSCAL-Tabs** – Katalog, Profil, SSP, Assessment Plan, Assessment Results, POAM und Component-Definition. Die Einfüge-Logik erkennt automatisch, ob der Tab ein Pydantic-Modell (Katalog, Profil) oder ein Dictionary-basiertes Datenmodell (SSP, AP, AR, POAM, Component) verwendet, und wählt die passende Strategie.

Praxis-Szenarien:

- **Audit-Trail:** Während eines Audits speichern Sie Links zu CVE-Einträgen, BSI-Warnmeldungen oder Herstellerdokumentationen direkt im Assessment-Results-Dokument
- **Compliance-Nachweis:** Verlinken Sie Richtlinien, Zertifikate oder Compliance-Reports als Back-Matter-Resource im SSP
- **Katalog-Referenz:** Beim Erstellen eines firmenspezifischen Katalogs verlinken Sie die Originalquellen (NIST, BSI, ISO) direkt im Dokument

Mehrwert: Der integrierte Browser eliminiert den Kontextwechsel zwischen Editor und Webbrowser. Besonders bei der Arbeit mit OSCAL-Spezifikationen, BSI-Kompendien oder NIST-Referenzen spart die direkte Integration viel Zeit und reduziert Fehler durch Copy-Paste zwischen Anwendungen.

5.6 OSCAL-TABS (Plugin) – Spezialisierte Editoren für Compliance-Dokumente

Die OSCAL-Tabs sind dynamische, spezialisierte Editoren, die für jeden der 8 OSCAL-Dokumenttypen eine maßgeschneiderte Oberfläche bieten. Sie werden automatisch geöffnet, wenn mjEdit eine OSCAL-Datei erkennt.

Gemeinsame Features aller OSCAL-Tabs

Funktion	Detail
Automatische Erkennung	OSCAL-Dateien werden beim Öffnen erkannt und im passenden Tab geladen
Baumstruktur (TreeView)	Hierarchische Navigation durch alle Dokumentelemente
Kontextmenüs	Pro Dokumenttyp spezialisierte Aktionen per Rechtsklick
Schema-Validierung	Echtzeit-Validierung gegen OSCAL v1.2.1
UUID-Auto-Generierung	RFC 4122-konforme UUIDs für neue Elemente
Metadata-Editor	Dialog für Titel, Version, last-modified, Rollen, Parteien
Markdown-Export	Jeder OSCAL-Tab kann als Markdown-Bericht exportiert werden
Tab-Registry	Zentrale Verwaltung aller geöffneten OSCAL-Tabs
Bidirektionale Synchronisation	Änderungen im Tab aktualisieren die JSON-Daten und umgekehrt

6.6.1 Katalog-Tab

Bearbeitung von Kontrollrahmenwerken (z.B. BSI IT-Grundschutz, NIST SP 800-53).

Funktion	Detail
Gruppen-Navigation	Hierarchische Baumstruktur mit Gruppen und Controls
Control-Details	Titel, Beschreibung, Properties, Parts, Parameter
Control-Suche	Filter nach ID, Titel oder Freitext
Kaskadenauswahl	Ganze Gruppen per Klick auswählen
Verschachtelte Gruppen	Bis zu 3 Ebenen tief (Gruppe → Untergruppe → Control)
Controls importieren	Aus anderen Katalogen übernehmen
Back-Matter	Base64-kodierte Ressourcen (PDFs, Bilder) anhängen

Use Case: Sie erstellen einen firmenspezifischen Katalog, indem Sie aus dem NIST SP 800-53 und dem BSI-Grundschutz relevante Controls auswählen, in Gruppen organisieren und mit eigenen Beschreibungen ergänzen.

6.6.2 Profil-Tab

Erstellung von Baselines durch Auswahl und Anpassung von Controls aus Katalogen.

Funktion	Detail
Baseline-Auswahl	Controls aus Import-Katalogen auswählen
Profil-Tailoring	Controls ergänzen, einschränken oder ersetzen
Parameter-Tuning	Organisationsspezifische Werte setzen (z.B. Passwortlänge)
Profil-Auflösung	Flachen Katalog aus Profil + Basis-Katalog generieren
Profil-Merge	Mehrere Profile zu einem kombinieren
Exclusion-Listen	Nicht relevante Controls explizit ausschließen
Speichern-unter	Dialog mit Zielverzeichnis-Auswahl

Use Case: Sie erstellen ein „Virtueller Windows Server 2019"-Profil, indem Sie aus dem NIST-Katalog die relevanten Controls auswählen, Parameter wie Audit-Retention-Period anpassen und nicht anwendbare Controls (physische Sicherheit) ausschließen.

6.6.3 SSP-Tab (System Security Plan)

Das umfangreichste OSCAL-Tab – dokumentiert ein komplettes System mit Komponenten, Inventar und Control-Implementierung.

Funktion	Detail
System-Definition	Metadata, Authorization-Boundary, Systemtyp
Komponenten-Registry	Software, Hardware, Services, Policies als Bausteine
Inventar-Management	Konkrete Assets mit Netzwerk-Properties (IP, MAC, FQDN)
Komponenten-Inventar-Verknüpfung	OOP-Modell: Komponente = Klasse, Inventar = Instanz
Control-Implementierung	By-Component-Zuordnung: Welche Komponente erfüllt welches Control
Rollen & Parteien	Verantwortlichkeiten zuordnen
Netzwerk-Discovery	IP-Adressen, Hostnamen automatisch ins Inventar
CSV-Import/Export	Inventar aus Excel/CMDB importieren
Netzwerk-Diagramm	Automatische NWDiag-Generierung aus Inventar-Daten
Compliance-Check	Control-Abdeckung analysieren (implementiert/teilweise/geplant)
Impact-Analyse	CVE → betroffene Komponenten ermitteln
Dynamische Dokumentation	Auto-Update bei Inventar-Änderungen

Use Case: Sie dokumentieren Ihre Server-Landschaft: Definieren „Apache HTTP Server" und „MySQL" als Komponenten (Klassen), erstellen Inventar-Items für jeden konkreten Server (Instanzen), verknüpfen beides

und dokumentieren pro Komponente, welche Controls implementiert sind.

6.6.4 Assessment Plan-Tab

Planung von Sicherheitsprüfungen und Audits.

Funktion	Detail
Assessment-Aktivitäten	Prüfmethoden definieren (Review, Interview, Test, Examine)
Assessment-Subjects	Zu prüfende Systeme/Komponenten auswählen
Scope-Definition	Grenzen und Zeitrahmen der Prüfung
Assessment-Team	Prüfer und Rollen zuordnen
Aus SSP generieren	Assessment Plan automatisch aus bestehendem SSP ableiten

Use Case: Sie bereiten ein ISO 27001-Audit vor: Aus dem SSP wird ein Assessment Plan generiert, der alle implementierten Controls, die Prüfmethoden (Interview mit Admin, Test der Firewall-Regeln) und den Zeitplan enthält.

6.6.5 Assessment Results-Tab

Dokumentation von Prüfergebnissen und Findings.

Funktion	Detail
Findings erfassen	Schwachstellen mit Severity und Beschreibung
Observationen	Zusätzliche Erkenntnisse und Notizen
Evidenz-Verknüpfung	Links zu Nachweisdokumenten und Ressourcen
Risiko-Bewertung	Severity-Klassifikation (kritisch, hoch, mittel, niedrig)
Assessment-Log	Audit-Trail aller Prüfschritte
POA&M-Verknüpfung	Findings direkt in Maßnahmenplan überführen
Aus Assessment Plan generieren	Results-Vorlage aus bestehendem Plan

Use Case: Während des Audits dokumentieren Sie Ihre Findings: „Firewall-Regeln nicht dokumentiert" (Severity: hoch), „Backup-Test nicht durchgeführt" (Severity: mittel). Jedes Finding wird direkt als POA&M-Item in den Maßnahmenplan überführt.

6.6.6 POA&M-Tab (Plan of Action & Milestones)

Tracking von Maßnahmen und Remediation nach Audits.

Funktion	Detail
Remediation-Items	Maßnahmen mit Zieldatum und Beschreibung

Funktion	Detail
Risiko-Priorisierung	Impact-basierte Priorisierung
Meilenstein-Tracking	Start → Geplant → Abschluss mit Fortschrittskontrolle
Status-Verwaltung	Geplant, In Arbeit, Abgeschlossen, Abgebrochen
Ressourcen-Zuordnung	Verantwortliche und Budget pro Maßnahme
Automatische Status-Updates	Bei Verlinkung mit Assessment Results
Aus SSP/AR generieren	POA&M direkt aus bestehenden Dokumenten

Use Case: Nach dem Audit erstellen Sie aus den Assessment Results automatisch einen Maßnahmenplan: 5 kritische Maßnahmen (Deadline: 30 Tage), 12 mittlere (90 Tage), 3 niedrige (180 Tage). Jede Maßnahme hat einen Verantwortlichen und Meilensteine.

6.6.7 Component Definition-Tab

Bibliothek wiederverwendbarer Systemkomponenten.

Funktion	Detail
Komponenten-Bibliothek	Wiederverwendbare Bausteine (Software, Hardware, Service, Policy)
Control-Implementierung	Welche Controls implementiert jede Komponente
Implementierungs-Statements	Beschreibung der konkreten Umsetzung
Revisionshistorie	Remarks-History über Versionen
Import/Export	Komponenten zwischen Projekten austauschen

Use Case: Ihre Organisation nutzt standardisierte Linux-Server. Sie definieren eine „Ubuntu 24.04 LTS"-Komponente mit allen implementierten Controls (Patch-Management, Logging, Zugriffssteuerung). Diese Komponente wird in jedem neuen SSP wiederverwendet.

6.6.8 Mapping Collection-Tab — Das offline-gestützte KI-Mapping-Cockpit

Der **OSCAL-Mapping-Tab** ist eines der herausragenden Alleinstellungsmerkmale von mjEdit. Er erlaubt es, Controls aus zwei verschiedenen OSCAL-Dokumenten (Quelle → Ziel) systematisch aufeinander abzubilden — zwischen Katalogen (z. B. NIST SP 800-53 ↔ BSI IT-Grundschutz), zwischen einem Katalog und einem Profil, oder zwischen zwei Profilen unterschiedlicher Tailoring-Stufen. Das Ergebnis ist ein OSCAL-konformes **Mapping-Collection-Dokument** (`mapping-collection`), das jede Zuordnung mit Quell-/Ziel-UUID, Relations-Typ (`equivalent-to`, `subset-of`, `superset-of`, `intersects-with`), Confidence-Score und optionalen Notizen versieht.

> **Screenshot-Hinweis:** *Mapping-Tab im geteilten Layout: links die Source-Control-Liste mit Such-/Filterleiste, rechts die Target-Control-Liste; in der Mitte die Vorschlagsliste mit Confidence-Balken (0–*

> *100 %) und Buttons „Accept", „Reject", „Auto-Accept".*

Funktionsübersicht:

Funktion	Detail
Auto-Suggest mit drei Methoden	`syntactic` (Token-Overlap), `semantic` (Sentence-Transformer-Embeddings), `auto` (Kombination + Score-Fusion)
Offline-KI-Matching	Lokales Sentence-Transformer-Modell (`paraphrase-multilingual-MiniLM-L12-v2`, 3.9 GB), keine Cloud, keine Datenfreigabe
Sprachübergreifend	Kontrollen in DE/EN/FR/IT können direkt verglichen werden — Embeddings sind multilingual
Auto-Accept-Schwelle	Konfigurierbarer Confidence-Schwellenwert (Standard 0.4); Vorschläge darüber werden automatisch übernommen
Bidirektional & visuell	Verbundene Controls werden farblich markiert; ungemappte Controls sind sofort sichtbar
Mapping-Status	`complete`, `not-complete`, `draft`, `deprecated`, `superseded` pro Zuordnung (OSCAL 1.2.1 Schema-Enum)
Bulk-Operationen	Alle Vorschläge über Schwelle akzeptieren, alle ablehnen, Filter nach Score-Range
Schema-Validierung	Vor jedem Speichern Validierung gegen das OSCAL `mapping-collection`-Schema
Markdown-Export	Vollständige Mapping-Tabelle als Bericht (mit Confidence-Spalte und Begründungen)

> **Screenshot-Hinweis:** *Detail-Ansicht eines aufgeklappten Vorschlags: Quell-Control „A.5.1 Information security policies" (ISO 27001) neben Ziel-Control „AC-1 Policy and Procedures" (NIST SP 800-53) mit Confidence 0.87, Begründung der drei Methoden (syntactic 0.42, semantic 0.91, kombiniert 0.87) und freiem Notiz-Feld.*

Das offline-gestützte KI-Matching im Detail **Warum offline?** Compliance-Dokumente — vor allem SSPs und interne Kontroll-Kataloge — enthalten oft hochsensible Informationen über Systemarchitekturen, Schwachstellen und Sicherheitsmaßnahmen. Sie dürfen die Organisation nicht verlassen. Cloud-basierte KI-Mapper (OpenAI, Anthropic…) sind hier rechtlich (DSGVO, BSI C5, BSI Mindeststandards, ITSiG, EU-AI-Act) und vertraglich oft schlicht **nicht zulässig**. mjEdit nutzt deshalb ein **rein lokales** multilinguales Sentence-Transformer-Modell, das einmalig heruntergeladen und im Anwender-Profil (`%APPDATA%\mjEdit\models\`) gespeichert wird. **Kein Token verlässt den Rechner.**

Wie funktioniert es? Der Mapping-Service bildet Titel und Beschreibung jeder Control auf einen 384-dimensionalen Embedding-Vektor ab und berechnet die Cosinus-Ähnlichkeit aller Quelle-×-Ziel-Paare. Im `auto`-Modus werden die semantische und die syntaktische Ähnlichkeit (Token-Überlappung, Lemmatisierung, Levenshtein) gewichtet kombiniert. Das Modell ist in 50+ Sprachen trainiert — ein BSI-Grundschutz-Baustein auf Deutsch wird zuverlässig dem englischsprachigen NIST-Control zugeordnet, ohne dass der Anwender die Texte vorher übersetzen muss.

Performance: Auf einem typischen Büro-Notebook (CPU-only) werden ca. 2.000–10.000 Vergleichspaare pro Sekunde verarbeitet. Eine vollständige Auto-Suggest-Analyse zwischen 200 BSI-Bausteinen und 800 NIST-Controls (=160.000 Paare) dauert typischerweise unter 30 Sekunden.

> **Screenshot-Hinweis:** *Statusleiste während Auto-Suggest mit Fortschrittsbalken „768 von 2.304 Vergleichsoperationen … ETA 12 s“ und Logfile-Eintrag* `[MAPPING-USE] Nutze Modell 'paraphrase-multilingual-MiniLM-L12-v2' fuer Mapping: nist-800-53.json -> bsi-grund`

Drei Anwendungsfälle, in denen der Mapping-Tab seinen Wert besonders ausspielt Use Case 1 — Multi-Framework-Compliance: BSI-Grundschutz ↔ NIST SP 800-53 ↔ ISO 27001

Ausgangslage: Ein international tätiger Mittelständler muss gegenüber deutschen Behörden BSI IT-Grundschutz nachweisen, gegenüber US-Kunden NIST SP 800-53 und gegenüber Konzernzentrale ISO 27001. Eine manuelle Cross-Reference-Tabelle in Excel ist fehleranfällig und veraltet ständig.

Mit dem Mapping-Tab: 1. Drei Kataloge öffnen (BSI, NIST, ISO). 2. Für jedes Paar (BSI↔NIST, BSI↔ISO, NIST↔ISO) ein Mapping-Collection-Dokument anlegen. 3. Auto-Suggest mit `auto`-Methode auslösen — das System schlägt in 1–2 Minuten alle plausiblen Korrespondenzen vor. 4. Mit Auto-Accept-Schwelle 0.7 die hochkonfidenten 60–70 % der Mappings automatisch übernehmen. 5. Restliche Vorschläge manuell prüfen und akzeptieren oder ablehnen. 6. Markdown-Export als Cross-Reference-Matrix für das Audit.

Mehrwert: Statt 4–6 Personentage manueller Recherche reicht ein halber Tag. Das Mapping ist versionierbar (jede Mapping-Datei kann unter Versionskontrolle stehen), wiederverwendbar und revisionssicher.

> **Screenshot-Hinweis:** *Drei nebeneinander geöffnete Tabs (BSI↔NIST, BSI↔ISO, NIST↔ISO), jeder mit Statusbalken und Anzahl gemappter Controls; im Hintergrund die Cross-Reference-Tabelle als Markdown-Vorschau.*

Use Case 2 — Profil-Tailoring nachvollziehen und visualisieren

Ausgangslage: Aus dem NIST-SP-800-53-Katalog wird ein **High-Baseline-Profil** abgeleitet, daraus ein noch strikteres organisationsspezifisches Profil. Auditoren möchten den Tailoring-Pfad nachvollziehen: Welche Controls wurden übernommen, welche modifiziert, welche entfernt?

Mit dem Mapping-Tab: Der Tab funktioniert nicht nur zwischen Katalogen, sondern auch zwischen einem Katalog und einem davon abgeleiteten Profil. Er zeigt sofort:

- **Übernommene Controls** (semantic = 1.0, syntactic = 1.0): unverändert.

- **Modifizierte Controls** (semantic > 0.85, syntactic < 0.95): Inhalt angepasst, z. B. Parameter-Tuning.
- **Entfernte Controls** (im Quell-Katalog, aber kein Mapping-Eintrag): aus der Baseline herausgenommen.
- **Neue organisationsspezifische Controls** (im Profil, aber kein Quell-Mapping): eigene Ergänzungen.

Mehrwert: Der Tab erstellt automatisch die Begründungs-Dokumentation, die viele Auditoren zwingend einfordern („Warum wurde Control XY ausgeschlossen?"). Mapping-Notizen werden direkt zu Audit-Belegen.

> **Screenshot-Hinweis:** *Filter-Ansicht „Nur entfernte Controls anzeigen": 12 BSI-Bausteine, die im Tailoring-Profil entfallen sind, jeder mit Notiz-Feld für die Begründung.*

Use Case 3 — Gap-Analyse und Risiko-Identifikation bei Framework-Migration

Ausgangslage: Eine Bundesbehörde migriert von einem bestehenden BSI IT-Grundschutz 2023-Kontrollwerk auf das neue **BSI IT-Grundschutz++** (Nachfolger ab 2026/2027). Die zentrale Frage: **Welche bestehenden Bausteine und Anforderungen haben kein Pendant im neuen Framework?** Diese sind potenzielle Risikoquellen.

Mit dem Mapping-Tab: 1. BSI IT-Grundschutz 2023 als Quelle, BSI IT-Grundschutz++ als Ziel. 2. Auto-Suggest mit niedriger Schwelle (0.3), um auch schwache Korrespondenzen sichtbar zu machen. 3. Filter „Ungemappte Quell-Controls" — das sind Bausteine ohne Pendant im neuen Framework. 4. Für jeden ungemappten Eintrag: Risiko-Analyse, ob die Anforderung weiterhin notwendig ist (→ als organisationsspezifische Ergänzung im Profil) oder im neuen Framework anders abgedeckt wird. 5. Markdown-Export als **Gap-Analyse-Bericht** mit Risiko-Score-Spalte.

Mehrwert: Eine Aufgabe, die ohne Tooling Wochen dauert (manuelles Durchgehen von 800+ Bausteinen/Anforderungen), wird auf wenige Stunden reduziert. Die Lücken-Liste ist die Grundlage für eine fundierte Migrations-Entscheidung von Grundschutz 2023 nach Grundschutz++.

> **Screenshot-Hinweis:** *Filter-Modus „Ungemappte Quell-Controls" mit roter Markierung; rechts Sidebar zeigt Statistik „23 von 312 Controls ohne Mapping (7.4 %)" und einen „Gap-Bericht erstellen"-Button.*

Vom Mapping zur OSCAL-Dokumentenkette — Generierung weiterer Dokumente Das im Mapping-Tab erzeugte `mapping-collection`-Dokument ist nicht das Endergebnis — es ist die **Grundlage** für die automatische Erzeugung weiterer OSCAL-Dokumente. Aus einem fertigen Mapping lassen sich über die Schaltfläche **„OSCAL-Dokumente generieren…"** im Mapping-Tab folgende abgeleiteten Dokumente erzeugen:

Generiertes Dokument	Anwendungszweck
Resolved Profile (`profile`)	Ein Tailoring-Profil aus dem Ziel-Katalog, das genau die im Mapping akzeptierten Controls als Baseline enthält. *Anwendung:* Direkter Einsatz als Compliance-Vorgabe für Auditoren.

Generiertes Dokument	Anwendungszweck
SSP-Vorschlag (`system-security-plan`)	Ein SSP-Stub mit `control-implementation`-Einträgen für jede gemappte Ziel-Control, vorbefüllt mit `description` aus der Quell-Control. *Anwendung:* Beschleunigt die SSP-Erstellung dramatisch — die organisationsspezifischen Beschreibungen aus dem Quell-Framework werden 1:1 übernommen und müssen nur noch verfeinert werden.
Cross-Reference-Bericht (Markdown)	Tabelle aller Mappings mit Confidence, Begründung, Status. *Anwendung:* Audit-Beleg, Management-Reporting, interne Schulungsunterlage.
Gap-Analyse-Bericht (Markdown)	Liste aller ungemappten Quell- bzw. Ziel-Controls mit Begründungsfeldern. *Anwendung:* Risiko-Identifikation bei Framework-Migrationen, Basis für POA&M-Einträge.
Component-Definition-Stub (`component-definition`)	Für jede Komponente eine Liste der gemappten Ziel-Controls mit übernommenen Implementations-Statements. *Anwendung:* Aufbau wiederverwendbarer Komponenten-Bibliotheken (z. B. „Unsere Standard-Linux-Server-Härtung gegenüber NIST und BSI").
POA&M-Stub (`plan-of-action-and-milestones`)	Für jeden ungemappten Quell-Control automatisch ein Eintrag mit Status „open". *Anwendung:* Schließung von Kontroll-Lücken im neuen Framework systematisch nachverfolgen.

> **Screenshot-Hinweis:** *Dialog „OSCAL-Dokumente generieren" mit Checkbox-Liste der sechs Optionen, Zielverzeichnis-Auswahl und Vorschau-Bereich, der das geplante Output-Set zeigt.*

Wo der Mapping-Tab Automatisierungsvorteile bringt

Aufgabe	Manuell	Mit Mapping-Tab	Zeitersparnis
Cross-Reference NIST↔BSI (200×800 Controls)	4–6 Personentage	2–3 Stunden	**> 95 %**
Tailoring-Dokumentation (Katalog→Profil)	1–2 Personentage	30 Minuten	~ 90 %

Aufgabe	Manuell	Mit Mapping-Tab	Zeitersparnis
SSP-Vorbefüllung mit Implementations-Texten	Wochen (manuelles Copy-Paste)	Minuten (Generierung aus Mapping)	**> 95 %**
Gap-Analyse bei Framework-Migration	2–3 Wochen	1 Tag	~ 90 %
Audit-Cross-Reference-Bericht	2 Tage	1 Klick (Markdown-Export)	**> 99 %**
POA&M-Erstellung aus Lücken	Tage	Minuten	**> 95 %**

Besonders wertvoll: Die **Wiederverwendbarkeit**. Ein einmal erstelltes BSI↔NIST-Mapping kann von allen Projekten der Organisation genutzt werden, automatisch bei Framework-Updates aktualisiert (Auto-Suggest gegen die neue Version, Diff-Ansicht der Veränderungen) und lässt sich versionsgesteuert in Git ablegen — als wertvolles Stamm-Asset der Compliance-Organisation.

> **Screenshot-Hinweis:** *Mapping-Datei in der Datei-Tree-View mit Versionshistorie („v1.0 BSI Grundschutz 2023↔NIST rev5", „v2.0 BSI Grundschutz++↔NIST rev5") und Diff-Anzeige der geänderten Zuordnungen.*

5.7 Übergreifende Funktionen

Datei-Management

Funktion	Detail
Öffnen/Speichern	JSON, Markdown, Text mit Encoding-Erkennung
Export	XML, CSV, PDF, HTML
Templates	Dateien aus Vorlagen erstellen (mit Variablen-Substitution)
Snippets	Code-/JSON-Fragmente per Strg+< einfügen
Auto-Open	Letzte Datei beim Start wiederherstellen
Projekt-Verzeichnisse	Schnellzugriff auf Projektordner

5.8 Datei-Tree-View – Zentrale Projektverwaltung

Der **Datei-Tree-View** ist weit mehr als eine einfache Ordnerübersicht. Er dient als **zentrales Cockpit** für die Verwaltung flexibler Projektverzeichnisstrukturen und ist die primäre Anlaufstelle für alle dateiorientierten

Aktionen in mjEdit.

Flexible Projektstruktur aufbauen

OSCAL-Compliance-Projekte bestehen typischerweise aus einer Vielzahl miteinander verbundener Dokumente – Kataloge, Profile, SSPs, Assessment Plans, Assessment Results und POA&M-Dateien. mjEdit empfiehlt und unterstützt eine klare, navigierbare Verzeichnishierarchie:

```
010-ISMS/                          ← Projektverzeichnis (via Config: sys_project_path)
├── 010-Kataloge/
│   ├── bsi-grundschutz-2024.json  ← OSCAL-Katalog
│   └── nist-sp-800-53.json
├── 020-Profile/
│   ├── baseline-server.json       ← OSCAL-Profil (verweist auf Katalog)
│   └── baseline-client.json
├── 030-SSP/
│   ├── ssp-webserver-prod.json    ← System Security Plan (verweist auf Profil)
│   └── ssp-db-cluster.json
├── 040-Assessment/
│   ├── ap-2026-q1.json            ← Assessment Plan
│   └── ar-2026-q1.json            ← Assessment Results
├── 050-POAM/
│   └── poam-2026.json             ← Plan of Action & Milestones
└── 060-Berichte/
    ├── compliance-report-q1.md    ← Markdown-Bericht
    └── audit-evidence.pdf
```

Projektverzeichnis konfigurieren: Über *Einstellungen → Projektpfad* (oder `sys_project_path` in der Config) legen Sie das Stammverzeichnis fest. Dieses Verzeichnis erscheint als Wurzel im Tree-View und ist gleichzeitig die Basis für relative Pfad-Auflösung zwischen OSCAL-Dokumenten.

Funktionen im Überblick

Funktion	Detail
Lazy-Loading	Große Verzeichnisse (1.000+ Dateien) werden schrittweise geladen
Farbige Datei-Icons	Dateitypen per Farbe unterscheiden – konfigurierbar pro Endung
Echtzeit-Filter	Case-insensitive Sofortsuche filtert den gesamten Baum inkl. Unterordnern
Kontextmenü	Rechtsklick öffnet kontextbezogene Aktionen je nach Dateityp
Template-basiertes Erstellen	Neue Dateien direkt aus Template im gewünschten Verzeichnis anlegen
Backup-Indikator	Symbol zeigt an, ob für eine Datei Backups vorhanden sind

Funktion	Detail
Multi-Selektion	Mehrere Dateien gleichzeitig für Batch-Operationen markieren
Umbenennen per F2	Datei oder Ordner direkt im Baum umbenennen
Drag & Drop	Dateien per Drag & Drop verschieben
Neuer Ordner	Unterordner direkt im Baum erstellen

Kontextmenü – Aktionen per Rechtsklick

Das Kontextmenü zeigt je nach Dateityp unterschiedliche Optionen:

Menü-Eintrag	Dateityp	Aktion
Öffnen	Alle	Öffnet die Datei im passenden Tab (OSCAL → OSCAL-Tab, .md → MD-Tab)
Öffnen als Text	Alle	Öffnet die Datei immer im Text-Tab, bypassed OSCAL/MD-Erkennung
Umbenennen	Alle	Datei/Ordner umbenennen (F2)
Kopieren	Alle	Datei in Zwischenablage kopieren
Verschieben	Alle	Datei verschieben
Löschen	Alle	Datei löschen (mit Bestätigungsdialog)
Neue Datei aus Template	Ordner	Template-Auswahl-Dialog öffnen, Datei im Ordner erstellen
Neuer Ordner	Ordner	Unterordner erstellen
Backup erstellen	JSON/Markdown	Sofortiges Backup der Datei
Versionen anzeigen	JSON/Markdown	Versions-Archiv-Dialog öffnen

> **Tipp: „Öffnen als Text"** — OSCAL-Dateien werden normalerweise automatisch im spezialisierten OSCAL-Tab geöffnet. Mit „ Öffnen als Text" können Sie die Datei gezielt im Text-Tab öffnen, um den Rohinhalt zu sehen oder manuell zu bearbeiten, ohne dass das OSCAL-Routing eingreift.

Echtzeit-Filter – Schnell navigieren in großen Projekten

Der Filter über dem Tree-View ist ein leistungsstarkes Navigationswerkzeug für große OSCAL-Projekte mit vielen Dateien. Die Suche ist:

- **Case-insensitiv** – „SSP", „ssp" und „Ssp" finden dieselben Dateien
- **Rekursiv** – durchsucht alle Ebenen der Verzeichnishierarchie
- **Sofort** – gefiltert wird bei jeder Eingabe ohne Verzögerung
- **Pfad-durchsuchend** – sucht in Dateinamen **und** Verzeichnisnamen

Beispiele:

Eingabe	Gefundene Dateien
`ssp`	`ssp-webserver.json`, `ssp-db.json`, `030-SSP/`
`q1`	`ap-2026-q1.json`, `ar-2026-q1.json`
`grundschutz`	`bsi-grundschutz-2024.json`
`.md`	Alle Markdown-Dateien im Projekt

Farbige Datei-Icons konfigurieren

Über *Einstellungen → Dateifarben* lassen sich Endungen individuell einfärben:

Standard-Konfiguration	Farbe	Beschreibung
`.json`	Grün	JSON- und OSCAL-Dateien
`.md`	Blau	Markdown-Dokumentation
`.pdf`	Rot	PDF-Dokumente
`.txt`	Grau	Textdateien
`.csv`	Orange	Tabellendaten

Die Konfiguration erfolgt in `config/config.json` unter dem Schlüssel `file_color_map`. Neue Endungen können jederzeit ohne Neustart hinzugefügt werden.

Use Cases

Use Case 1 – OSCAL-Projektverzeichnis strukturieren: Sie beginnen ein neues ISMS-Projekt. Im Tree-View erstellen Sie per Kontextmenü die gewünschte Ordnerstruktur (Kataloge/, Profile/, SSP/ etc.). Anschließend nutzen Sie „Neue Datei aus Template" für jeden OSCAL-Dokumenttyp und erhalten sofort vorbefüllte Startdokumente an der richtigen Stelle.

Use Case 2 – Dateien zwischen Projekten tauschen: Sie arbeiten parallel an zwei Kundenprojekten. Per Drag & Drop im Tree-View verschieben Sie ein bewährtes SSP-Profil aus Projekt A in das Profil-Verzeichnis von Projekt B. mjEdit erkennt beim nächsten Öffnen automatisch den OSCAL-Typ und die veränderten relativen Pfad-Referenzen.

Use Case 3 – Schnellnavigation per Filter: Ihr Projekt hat 200+ Dateien. Sie suchen alle Assessment-Results-Dokumente: Einfach „ar-" in das Filter-Feld eingeben – nur die relevanten Dateien bleiben sichtbar. Doppelklick öffnet die gewünschte Datei sofort.

Use Case 4 – Datei als Text prüfen: Ein Kollege hat eine OSCAL-Profil-Datei übermittelt, die sich nicht öffnen lässt. Per Rechtsklick → „ Öffnen als Text" sehen Sie den Rohinhalt im Text-Tab – Sie erkennen sofort, dass ein Pflichtfeld fehlt. Nach der Korrektur öffnet sich die Datei korrekt im Profil-Tab.

5.9 Backup- und Versionierungskonzept

mjEdit bietet ein **zweistufiges Sicherheitsnetz** für alle bearbeiteten Dateien: automatische **Backups** (zeitbasierte Vollkopien) und ein **Versions-Archiv** (manuelle Momentaufnahmen). Beide Mechanismen sind unabhängig voneinander und ergänzen sich optimal.

Stufe 1: Automatisches Backup-System

Das automatische Backup-System erstellt komprimierte Kopien im Hintergrund – ohne manuelle Eingriffe und ohne Arbeitsunterbrechung.

Wie es funktioniert:

1. Beim **Öffnen einer Datei** prüft mjEdit, ob das letzte Backup älter als das konfigurierte Intervall ist
2. Beim **Speichern** (Strg+S) wird optional automatisch ein Backup angelegt
3. **Komprimierte Speicherung** als `.json.gz`-Archiv – OSCAL-Dateien schrumpfen typischerweise auf 5–10 % der Originalgröße
4. **Namenskonvention:** `{originaldateiname}_YYYY-MM-DD_HH-MM-SS.json.gz`
5. **Backup-Verzeichnis:** `config/backups/{projektpfad-hash}/`

Konfigurierbare Parameter (über *Einstellungen → Backup*):

Parameter	Standardwert	Beschreibung
Automatische Backups	aktiviert	Master-Schalter für automatische Backups
Backup-Intervall	30 Minuten	Mindestabstand zwischen zwei automatischen Backups
Max. Backups pro Datei	10	Älteste Backups werden automatisch gelöscht
Komprimierung	aktiviert	GZIP-Komprimierung spart bis zu 95 % Speicherplatz
Backup bei Speichern	aktiviert	Zusätzliches Backup bei jedem Strg+S

Manuelles Backup (Strg+Umschalt+B): Über den Shortcut oder das Menü *Datei → Backup erstellen* wird sofort eine komprimierte Kopie der aktuellen Datei angelegt. Ideal vor größeren Änderungen oder vor dem Ausprobieren einer automatischen OSCAL-Transformation.

Backup-Status im Tree-View: Dateien mit vorhandenen Backups erhalten ein kleines Symbol im Tree-View. So sehen Sie auf einen Blick, welche Dateien gesichert sind – und welche noch nicht.

Stufe 2: Versions-Archiv

Das Versions-Archiv ist eine explizite Sammlung benannter Momentaufnahmen. Im Gegensatz zu automatischen Backups können Versionen mit einem beschreibenden Namen versehen werden – wie Commits in einem Versionsverwaltungssystem.

Versions-Archiv-Dialog öffnen:

- Über *Datei → Versionen anzeigen*
- Über Rechtsklick im Tree-View → „🕐 Versionen anzeigen"
- Via Kontextmenü im Text-Tab

Versions-Archiv-Dialog – Funktionen:

Funktion	Beschreibung
Versionsliste	Alle Versionen mit Zeitstempel, Dateigröße und Beschreibung
Vorschau	Inhalt der ausgewählten Version vor der Wiederherstellung prüfen (JSON-formatiert)
Diff anzeigen	Zeilenweiser Vergleich zwischen ausgewählter Version und aktueller Datei
Wiederherstellen	Ausgewählte Version ersetzt die aktuelle Datei (nach Bestätigung)
Löschen	Version aus dem Archiv entfernen
Exportieren	Version als separate JSON-Datei an einem anderen Ort speichern

Sicherer Wiederherstellungs-Workflow

Der Wiederherstellungsprozess ist bewusst mehrstufig gestaltet, um versehentlichen Datenverlust zu verhindern:

```
1. Versionen-Dialog öffnen
       ↓
2. Version in der Liste auswählen
       ↓
3. Vorschau prüfen → ist es die richtige Version?
       ↓
4. Optional: Diff zur aktuellen Datei anzeigen
       ↓
5. „Wiederherstellen" klicken
       ↓
6. Bestätigungsdialog: „Aktuelle Datei wirklich überschreiben?"
       ↓
7. Automatisches Backup der aktuellen Datei VOR dem Überschreiben
       ↓
8. Version wird wiederhergestellt und in mjEdit geöffnet
```

Sicherheitsnetz beim Wiederherstellen: Vor jeder Wiederherstellung wird automatisch ein Backup der aktuellen Datei angelegt. Damit ist auch das Zurücksetzen einer Wiederherstellung möglich – ein zweistufiges Sicherheitsnetz.

Backup und Versionierung im Zusammenspiel

Szenario	Empfohlene Maßnahme
Tägliche Arbeit absichern	Automatische Backups aktiv lassen (Standardeinstellung)
Vor einem größeren Bearbeitungsschritt	Strg+Umschalt+B → manuelles Backup mit Zeitstempel
Meilenstein dokumentieren (z.B. „v1.0 Review“)	Version anlegen mit beschreibendem Namen
Datei versehentlich überschrieben	Backup-Dialog → älteste Version wählen → Vorschau → Wiederherstellen
Zwei Versionen inhaltlich vergleichen	Versions-Archiv → Diff anzeigen
Backup an einen anderen Ort kopieren	Version exportieren → Datei in anderem Verzeichnis speichern

Praxis-Beispiel – OSCAL-Profil-Tailoring absichern:

1. Sie öffnen ein bewährtes BSI-Grundschutz-Profil und wollen es für einen neuen Kunden anpassen
2. Strg+Umschalt+B → Backup mit Bezeichnung „Original BSI-Profil vor Kundenprojekt X“
3. Sie beginnen mit dem Tailoring – 15 Controls ausschließen, 8 Parameter anpassen
4. Beim Testen der Profil-Auflösung stellen Sie fest, dass ein Control fälschlicherweise ausgeschlossen wurde
5. Backup-Dialog → Version von Schritt 2 wählen → Vorschau zeigt das Original → Wiederherstellen

Technische Details

Aspekt	Details
Komprimierungsformat	GZIP (Python `gzip`-Modul), Level 6 (Balance Speed/Ratio)
Speicherort	`config/backups/` (plattformunabhängiger Pfad via `path_helper`)
Max. Speicherverbrauch	Konfigurierbar; Standard: 10 Backups × komprimierte Größe
Thread-Sicherheit	Backups werden asynchron im Hintergrund erstellt (kein UI-Freeze)
Dateiintegrität	CRC32-Prüfung beim Einlesen komprimierter Backups
Portabilität	Backups können manuell kopiert und auf anderen Systemen geöffnet werden

---

5.10 OSCAL: Intelligente Pfad-Korrektur für nicht erreichbare absolute Pfade

OSCAL-Dokumente referenzieren einander über **Href-Attribute** – Profil-Dateien verweisen auf Kataloge, SSPs auf Profile, und so weiter. Werden absolute Pfade verwendet (z. B. `C:\Users\max\Projekte\katalog.json`)

und die Projektstruktur anschließend auf einen anderen Rechner kopiert oder in ein anderes Verzeichnis verschoben, werden diese Referenzen ungültig.

mjEdit erkennt diesen Zustand automatisch und bietet eine **intelligente Pfad-Korrektur** mit zwei Auflösungsstufen.

Das Problem: Absolute Pfade nach dem Verschieben

Typisches Szenario:

1. OSCAL-Dokumente wurden auf Rechner A erstellt: Profil verweist auf Katalog via absolutem Pfad `C:\Users\alice\ISMS\katalog.json`
2. Projektverzeichnis wird auf USB-Stick kopiert oder per Git auf Rechner B übertragen
3. Auf Rechner B öffnet mjEdit das Profil – die absolute Pfad-Referenz `C:\Users\alice\...` ist nicht erreichbar
4. **Bisheriges Verhalten:** Einfache Warnmeldung, kein Lösungsangebot
5. **Neues Verhalten:** Intelligente Analyse, Vorschau der Korrekturen, automatische Konvertierung in relative Pfade

Zweistufige Pfad-Auflösung

Wenn mjEdit nicht-erreichbare absolute Pfade erkennt, prüft es automatisch, ob die referenzierten Dateien lokal gefunden werden können:

Stufe 1 – Quellverzeichnis-Suche: Sucht den Dateinamen (z. B. `bsi-grundschutz.json`) rekursiv im Verzeichnis der OSCAL-Quelldatei und ihren Unterverzeichnissen. Wird die Datei gefunden, berechnet mjEdit den relativen Pfad vom Standort der Quelldatei.

```
Absoluter Pfad (nicht erreichbar): C:\Users\alice\ISMS\Kataloge\bsi-grundschutz.json
Quelldatei liegt in:               /Projektverzeichnis/Profile/baseline.json
Datei gefunden in:                 /Projektverzeichnis/Kataloge/bsi-grundschutz.json
→ Vorgeschlagener relativer Pfad:  ../Kataloge/bsi-grundschutz.json
```

Stufe 2 – Projektindex-Suche (nur im Projektmodus): Wenn Stufe 1 keinen Treffer liefert, baut mjEdit einen Index aller Dateien im Projektverzeichnis auf (`{Dateiname → alle absoluten Pfade}`) und sucht darin nach dem Dateinamen der nicht-erreichbaren Referenz.

```
Datei nicht in Quellverzeichnis gefunden.
Projektindex enthält: "bsi-grundschutz.json" → "/Projekte/Lib/Kataloge/bsi-grundschutz.json"
Quelldatei liegt in:  "/Projekte/SSP/ssp-server.json"
→ Vorgeschlagener relativer Pfad: ../Lib/Kataloge/bsi-grundschutz.json
```

Wenn keine Auflösung möglich ist: Absolute Pfade, deren Dateien auch über den Projektindex nicht gefunden werden, werden unverändert belassen. mjEdit gibt einen Hinweis, welche Pfade nicht aufgelöst werden konnten.

Der Korrektur-Dialog

Sobald mjEdit mindestens einen auflösbaren Pfad findet, erscheint statt der einfachen Warnmeldung der interaktive **„Nicht erreichbare Pfade"-Dialog**:

Tabelle mit Vorschau:

Spalte	Inhalt
JSON-Pfad	Position des href-Attributs im OSCAL-Dokument
Absoluter Pfad	Ursprünglicher, nicht-erreichbarer absoluter Pfad
Vorgeschlagener relativer Pfad	Berechneter relativer Pfad (leer bei nicht auflösbar)
Status	✓ Auflösbar (grün) / ✗ Nicht auflösbar (rot)

Korrektur-Modi (RadioButtons):

Modus	Verhalten
Pfade beibehalten (keine Änderung)	Dialog schließen, Datei unverändert öffnen
Nur diese Datei korrigieren (Stufe 1)	Auflösbare Pfade in der aktuell geöffneten Datei auf relative Pfade setzen
Alle Dateien im Verzeichnis korrigieren	Alle OSCAL-JSON-Dateien im Quellverzeichnis prüfen und korrigieren (Stufe 1 je Datei)
Alle Dateien im Projektverzeichnis	Alle OSCAL-JSON-Dateien im Projekt prüfen (Stufe 1+2 mit vorab aufgebautem Projektindex)

> **Tipp:** Der Modus „Alle Dateien im Projektverzeichnis" ist ideal nach dem ersten Öffnen eines migrierten Projekts. Er konvertiert alle absoluten Pfad-Referenzen im gesamten Projekt auf einmal – danach funktioniert das Projekt auf jedem Rechner ohne Anpassung.

Batch-Verarbeitung mit Fortschrittsanzeige

Im Verzeichnis- oder Projektmodus startet mjEdit eine Hintergrundverarbeitung mit Fortschrittsanzeige:

- **Fortschrittsbalken** zeigt `Datei 12/47: ssp-server.json`
- **Abbruch-Schaltfläche** stoppt die Verarbeitung nach der nächsten Datei
- **Abschlussmeldung** zeigt: Geprüfte Dateien / geänderte Dateien / korrigierte Pfade gesamt

Übersprungene Verzeichnisse: Systemverzeichnisse wie `.venv`, `.git`, `__pycache__`, `node_modules` werden automatisch ausgeschlossen.

Empfohlener Arbeitsablauf bei Projekt-Migration

```
1. Projektverzeichnis auf neuen Rechner kopieren
       ↓
2. Erstes OSCAL-Dokument in mjEdit öffnen
       ↓
3. Pfad-Korrektur-Dialog erscheint (wenn absolute Pfade erkannt)
       ↓
4. Korrektur-Modus „Alle Dateien im Projektverzeichnis" wählen
       ↓
5. Batch-Verarbeitung startet → Fortschrittsanzeige
       ↓
6. Abschlussmeldung: „42 Dateien geprüft, 38 geändert, 156 Pfade korrigiert"
       ↓
7. Alle OSCAL-Dokumente öffnen und verlinken korrekt
```

Warum relative Pfade in OSCAL besser sind

Eigenschaft	Absolute Pfade	Relative Pfade
Portabilität	✗ Rechner-abhängig	☑ Überall verwendbar
Team-Arbeit	✗ Jeder Nutzer braucht gleichen Pfad	☑ Relativ zum Projektordner
Versionskontrolle	✗ Lokale Pfade im Repo	☑ Keine Umgebungs-Abhängigkeiten
Deployment	✗ Muss angepasst werden	☑ Funktioniert auf jedem System
OSCAL-Best-Practice	Nicht empfohlen	☑ NIST-Empfehlung für portable Dokumente

Theme-System

Theme	Beschreibung
Dark	Professioneller Dark-Mode (Standard), hoher Kontrast
Light	Helles Interface für Tageslicht-Arbeitsplätze
Blue	Blau-akzentuiertes Design
Green	Grün-akzentuiertes Design

- Dateifarben im Baum konfigurierbar pro Dateiendung
- Live-Umschaltung ohne Neustart
- Erweiterbar über QSS (Qt StyleSheets)

Druck und Export

Format	Verfügbar aus
Drucken	Text-Tab, Markdown-Tab, PDF-Tab (mit Seiteneinrichtung und Vorschau)
PDF-Export	Markdown-Tab, OSCAL-Tabs (mit Titel, Autor, Header/Footer, Font-Auswahl)
HTML-Export	Markdown-Tab
XML-Export	Text-Tab (JSON → XML)
CSV-Export	Text-Tab (JSON → CSV), SSP-Tab (Inventar)
Markdown-Export	Alle OSCAL-Tabs

Internationalisierung

Sprache	Status
Deutsch	☑ Vollständig (Quellsprache)
Englisch	☑ Vollständig übersetzt

- Sprachumschaltung ohne Neustart
- Alle GUI-Texte, Fehlermeldungen und Menüs übersetzt
- Erweiterbar für weitere Sprachen (gettext-basiert)

Globale Tastenkürzel

Kürzel	Aktion
Strg+N	Neue Datei
Strg+O	Datei öffnen
Strg+S	Speichern
Strg+Umschalt+S	Speichern unter
Strg+Q	Beenden
Strg+Tab / Strg+Umschalt+Tab	Nächster / Vorheriger Tab
Strg+W	Tab schließen
F6	Fokus wechseln (Baum ↔ Editor)
Strg+P	Drucken
Strg+Umschalt+B	Backup erstellen
Strg+<	Snippet einfügen
F1	Hilfe öffnen

Alle Kürzel sind in `keybindings.json` frei konfigurierbar.

6. Plugin-System im Detail

mjEdit ist von Grund auf **modular** aufgebaut. Statt alle Funktionen fest in den Kern einzubauen, lagert mjEdit spezialisierte Fähigkeiten in austauschbare **Plugins** aus – nach dem Prinzip der **funktionalen Modularität**. Das Ergebnis: Eine schlanke Kernanwendung, die nur das lädt, was Sie tatsächlich brauchen.

6.1 Warum ein Plugin-System?

Vorteil	Beschreibung
Schlankerer Start	Nur aktivierte Plugins werden geladen – weniger RAM, schnellerer Start
Funktionale Modularität	Jedes Plugin kapselt genau eine Fachdomäne (OSCAL, Browser, MCP, Datenbank) – klar getrennte Verantwortlichkeiten
Erweiterbarkeit	Neue Funktionen als Plugin hinzufügen, ohne den Kern-Code zu ändern
Individuelle Konfiguration	Jeder Anwender aktiviert nur die Plugins, die er benötigt
Unabhängige Updates	Plugins können einzeln aktualisiert werden, ohne die gesamte Anwendung neu auszuliefern
Entwicklerfreundlich	Eigene Plugins mit `BasePlugin`-Klasse, Hook-System und Service-Fassade in wenigen Zeilen erstellen

6.2 Technische Architektur

Das Plugin-System basiert auf drei Säulen:

1. Dreiphasiger Lifecycle – Sicherheit durch kontrolliertes Laden

Jedes Plugin durchläuft drei klar definierte Phasen:

Phase	Methode	Zeitpunkt	Was passiert
1. Laden	`on_load()`	Vor GUI-Initialisierung	Hooks registrieren, Menüs definieren. GUI ist noch **nicht** verfügbar!

Phase	Methode	Zeitpunkt	Was passiert
2. GUI Ready	`on_gui_ready()`	Nach GUI-Aufbau	Tabs erstellen, Widgets initialisieren, auf MainGUI zugreifen
3. Entladen	`on_unload()`	Beim Beenden	Ressourcen freigeben, Menüs entfernen, Hooks deregistrieren

Diese Trennung verhindert Race-Conditions und stellt sicher, dass Plugins nie auf eine unfertige GUI zugreifen.

2. Hook-basierte Kommunikation – Lose Kopplung statt harter Abhängigkeiten

Plugins kommunizieren über ein **Event-Hook-System** mit der Kernanwendung und untereinander. Statt direkte Methodenaufrufe zu verwenden, registrieren Plugins Callbacks für bestimmte Ereignisse:

```
# Plugin registriert sich für das Event "Datei geöffnet"
self.register_hook("file_opened", self.on_file_opened)
```

Wenn mjEdit eine Datei öffnet, werden alle registrierten Callbacks benachrichtigt – mit automatischer Fehler-Isolation: Wenn ein Plugin abstürzt, laufen alle anderen weiter.

Verfügbare Hooks: `file_opened`, `file_saved`, `file_renamed`, `save_active_plugin_tab`, `add_menu`, `add_toolbar`, `open_external_url`, `on_tab_changed` und weitere.

3. Service-Fassade – Kontrollierter Zugriff auf die Kernanwendung

Plugins greifen nicht direkt auf GUI-Interna zu, sondern über eine `PluginCoreServices`-Fassade. Das reduziert die Kopplung und macht Plugins robuster gegenüber internen Änderungen:

```
main_gui = self.get_main_gui()          # MainGUI-Instanz
self.services.set_status("Geladen")      # Statusbar setzen
self.services.log("Plugin aktiv", "info") # Logging mit Plugin-Name
```

6.3 Plugin-Manager – Plugins verwalten

Der integrierte **Plugin-Manager** (erreichbar über das Menü *Extras → Plugin-Manager*) bietet eine komfortable Oberfläche zur Verwaltung aller Plugins:

Funktion	Beschreibung
Aktivieren	Checkbox anhaken – Plugin wird beim nächsten Start geladen
Deaktivieren	Checkbox entfernen – Plugin wird nicht mehr geladen, spart Ressourcen

Funktion	Beschreibung
Plugin-Details	Name, Version, Typ (GUI/Editor/Tool) und Beschreibung einsehen
Aktualisieren	Plugin-Verzeichnis neu scannen (z.B. nach manuellem Hinzufügen eines Plugins)
Neustart	Speichert alle offenen Dateien und startet mjEdit mit der neuen Plugin-Konfiguration neu

So installieren Sie ein neues Plugin:

1. Kopieren Sie den Plugin-Ordner (mit `__init__.py` und `plugin.py`) in das Verzeichnis `plugins/`
2. Öffnen Sie den Plugin-Manager (*Extras → Plugin-Manager*)
3. Klicken Sie auf **Aktualisieren** – das neue Plugin erscheint in der Liste
4. Aktivieren Sie das Plugin per Checkbox und klicken Sie **Speichern**
5. Starten Sie mjEdit neu (Button **Neustart** oder manuell)

So deaktivieren Sie ein nicht benötigtes Plugin:

1. Öffnen Sie den Plugin-Manager
2. Entfernen Sie den Haken beim gewünschten Plugin
3. Klicken Sie **Speichern** und starten Sie mjEdit neu

Nicht benötigte Plugins (z.B. das Datenbank-Plugin oder das Beispiel-Plugin) können Sie jederzeit deaktivieren, um Ressourcen zu sparen und die Oberfläche übersichtlicher zu gestalten.

> **Hinweis für kompilierte Versionen (.exe):** In der als Standalone-Anwendung kompilierten Version sind alle Plugins fest im Anwendungspaket eingebettet. Sie können Plugins **aktivieren und deaktivieren**, jedoch nicht nachträglich entfernen oder neue hinzufügen. Wenn Sie eine maßgeschneiderte Version mit einem individuellen Plugin-Set benötigen – beispielsweise ohne Browser-Plugin oder mit branchenspezifischen Erweiterungen – können Sie eine **angepasste Professional-Version lizenzieren**. Kontaktieren Sie uns für Details zu kundenspezifischen Builds.

6.4 Mitgelieferte Plugins

mjEdit wird mit folgenden Plugins ausgeliefert:

Plugin	Typ	Standard	Beschreibung
OSCAL Compliance	Editor	☑ aktiv	Alle 8 OSCAL-Dokumenttypen mit spezialisierten Tabs (inkl. **OSCAL-Mapping-Tab** mit offline KI-Matching)
MCP Server	Tool	☑ aktiv	88 MCP-Tools für KI-Integration (STDIO + SSE)
Browser	Tool	☑ aktiv	Chromium-basierter Webbrowser mit Lesezeichen
Transform-Script	Editor	☑ aktiv	QC-Script-Erstellung und Konvertierung (F12)
Network Discovery	Tool	deaktiviert	Scapy-basierter LAN-Scan (ARP/ICMP); übernimmt Hosts als OSCAL `inventory-items` direkt in den aktiven SSP-Tab
Datenbank	Editor	deaktiviert	Datenbank-Anbindung für JSON-Daten
Beispiel-Plugin	Editor	deaktiviert	Template und Dokumentation für eigene Plugin-Entwicklung

---

6.5 OSCAL-Plugin (Kern-Plugin)

Das größte und umfangreichste Plugin – spezialisiert auf alle 8 OSCAL-Dokumenttypen.

Katalog-Funktionen:

- Gruppen-Navigation mit Baumstruktur
- Control-Anzeige mit vollständigen Details (Titel, Beschreibung, Properties, Parts)
- Control-Suche und -Filter nach ID, Titel oder Inhalt
- Controls hinzufügen, bearbeiten, löschen

- Verschachtelte Gruppen-Unterstützung (bis 3 Ebenen)
- Controls aus anderen Katalogen importieren
- UUID-Auto-Generierung (RFC 4122)
- Metadata-Verwaltung (Titel, Version, Datum)
- Back-Matter-Ressourcen (Base64-Anhänge)
- Kaskadenauswahl für ganze Gruppen per MCP

Profil-Funktionen:

- Profil-Erstellung mit Metadata
- Baseline-Auswahl aus Import-Katalogen
- Modifications: Ergänzungen, Löschungen, Ersetzungen
- Parameter-Tuning für Controls
- Profil-Auflösung (resolved catalog)
- Profil-Merge (mehrere Profile kombinieren)
- Exclusion-Listen
- Speichern-unter-Dialog mit Zielverzeichnis-Auswahl

SSP-Funktionen:

- System-Definition mit Metadata
- Komponenten-Management (Software, Hardware, Service, etc.)
- Inventar-Management (konkrete Assets mit Properties)
- Control-Implementierung dokumentieren (by-component)
- Verantwortlichkeits-Zuordnung (Rollen und Parteien)
- Netzwerk-Discovery (IP, Hostname, MAC automatisch)
- CSV-Import/Export für Inventar
- Netzwerk-Diagramm-Generierung (NWDiag)
- Compliance-Check (Control-Abdeckung analysieren)
- Impact-Analyse (CVE → betroffene Komponenten)
- Dynamische Dokumentation (Auto-Update bei Inventar-Änderungen)

Assessment Plan-Funktionen:

- Assessment-Aktivitäten definieren
- Assessment-Subjects auswählen
- Scope und Zeitrahmen festlegen
- Prüfmethoden zuordnen
- Schema-konforme Pflichtfelder (activities, subjects, back-matter)

Assessment Results-Funktionen:

- Findings mit Severity und Evidenz erfassen

- Observationen und Notizen dokumentieren
- Evidenz-Verknüpfung mit Ressourcen
- Berichts-Generierung

POA&M-Funktionen:

- Remediation-Items mit Zieldaten erstellen
- Risiko-Priorisierung
- Meilenstein-Tracking
- Status-Verwaltung (geplant, in Arbeit, abgeschlossen)
- Ressourcen-Zuordnung

Component Definition-Funktionen:

- Wiederverwendbare Komponentenbibliotheken erstellen
- Komponentenarten: Software, Hardware, Service, Policy, etc.
- Control-Implementierung pro Komponente dokumentieren

Mapping Collection-Funktionen:

- Bidirektionale Control-Zuordnung zwischen Frameworks
- **Offline-KI-Matching** mit lokalem multilingualem Sentence-Transformer-Modell (keine Cloud, keine Datenfreigabe)
- Drei Methoden: `syntactic`, `semantic`, `auto` (Score-Fusion)
- NIST ↔ interner Format-Adapter
- Visuelles Highlighting verbundener Controls
- Auto-Suggest mit konfigurierbarem Confidence-Schwellenwert
- Auto-Accept-Modus für Bulk-Übernahme hochkonfidenter Vorschläge
- Generierung abgeleiteter OSCAL-Dokumente (Resolved Profile, SSP-Vorschlag, Component-Definition-Stub, POA&M-Stub) aus dem Mapping
- Markdown-Export der Zuordnungstabelle (Cross-Reference, Gap-Analyse)
- Status-Unterstützung (complete, not-complete, draft, deprecated, superseded gemäß OSCAL 1.2.1)

6.6 MCP-Server-Plugin

Stellt mjEdit als MCP-Server für KI-Systeme bereit.

Server-Funktionen:

- STDIO und SSE Transport
- Auto-Start beim Anwendungsstart (konfigurierbar)
- Rate-Limiting (100 Requests/Minute, konfigurierbar)
- Path-Whitelist für Dateizugriff

- Audit-Logging aller Tool-Aufrufe
- Activity Monitor für Echtzeit-Kommunikationsanzeige
- Tool-Pagination (20 pro Seite) für LLM-Token-Limits
- Pre-Validation Sanitizer (Parameter-Korrektur vor Schema-Prüfung)
- Non-blocking Kommunikation
- SSE-Reconnection mit exponentieller Backoff

6.7 Browser-Plugin

Integrierter Webbrowser als Tab in mjEdit.

- URL-Navigation mit Adressleiste
- Lesezeichen-Verwaltung
- HTTPS-Status-Anzeige
- Mehrere Browser-Tabs
- Lazy-Loading (WebEngine erst bei Bedarf)
- Konfigurierbare Startseite
- Externe URLs aus Markdown automatisch öffnen

6.8 Transform-Script-Plugin

QC-Script-Transformation für qFORM-Formulare.

- GUI-Dialog für QC-Script-Erstellung
- Python → QC-String Konvertierung
- Syntax-Highlighting im Dialog
- QC-Validierung
- Hilfsfunktionen: `verify()`, `parse()`, `is_qc_string()`

6.9 Datenbank-Plugin

Datenbank-Anbindung für JSON-Daten.

- Datenbankverbindungen konfigurieren
- JSON aus Datenbank laden
- JSON in Datenbank speichern
- Bidirektionale Synchronisation
- Datenbank-Viewer
- SQL-Abfrage-Tool

6.10 Network-Discovery-Plugin — LAN-Inventarisierung direkt in den SSP

Das **Network-Discovery-Plugin** schließt eine der zeitintensivsten Lücken in jedem SSP-Projekt: die Erfassung des tatsächlich vorhandenen IT-Inventars. Das Plugin scannt das lokale Netzwerk per ARP-Sweep (scapy-basiert) und überträgt die gefundenen Hosts — inklusive IP-Adresse, MAC-Adresse, FQDN und Hostname — als OSCAL v1.2.1-konforme `inventory-item`-Einträge **direkt** in einen offenen SSP-Tab.

Architektur und Sicherheit:

- **scapy** als Netzwerkbibliothek (Raw-Sockets); kein `nmap`, keine Shell-Aufrufe, kein externer Dienst
- Subnet-Größe auf **maximal 4096 IP-Adressen** begrenzt (Anti-DoS)
- Vor jedem Scan **Bestätigungsdialog** mit Hinweis „Nur in eigenen Netzen scannen"
- **Reverse-DNS** mit 1 Sekunde Timeout pro Host, parallelisiert
- Unter Windows zusätzlich Npcap als Treiber erforderlich (Raw-Socket-Zugriff)
- Plugin ist **standardmäßig deaktiviert** und muss im Plugin-Manager bewusst aktiviert werden — keine Hintergrund-Scans ohne Anwender-Aktion

Wie es mit SSP-Dokumenten zusammenarbeitet:

Ein SSP (System Security Plan) enthält im Abschnitt `system-implementation.inventory-items` die Liste aller dem System zugeordneten konkreten Assets (Server, Switches, Endgeräte, IoT-Komponenten…). In klassischen Compliance-Projekten wird diese Liste manuell aus Excel-Inventarlisten oder aus CMDBs zusammengetragen — fehleranfällig, schnell veraltet, oft unvollständig. Das Network-Discovery-Plugin erlaubt einen **Live-Abgleich**:

1. Im OSCAL-SSP-Tab des Zielsystems arbeiten.
2. über Menü **Plugins** → 🔎 **Netzwerk-Discovery…** den Scan-Dialog öffnen.
3. Subnet (CIDR, z. B. `10.0.5.0/24`) und Timeout eingeben, **Scan starten**.
4. In der Ergebnis-Tabelle die Hosts an-/abwählen, die in den SSP übernommen werden sollen.
5. Ziel-SSP aus Dropdown wählen (alle offenen SSP-Tabs werden gelistet); ggf. „Duplikate (gleiche IP) überspringen" aktivieren, um Bestandsdaten nicht zu überschreiben.
6. **„In SSP übernehmen"** klicken — die ausgewählten Hosts werden an `system-implementation.inventory-items` angehängt; der SSP-Tab wird als geändert markiert (`*` im Tab-Titel) und kann mit `Ctrl+S` gespeichert werden.

OSCAL-Mapping der Scan-Ergebnisse (jeder Host wird zu einem `inventory-item`):

Scan-Ergebnis	OSCAL-Feld
IP-Adresse	`props[].name="ipv4-address"`
MAC-Adresse	`props[].name="mac-address"`
FQDN (vollständiger DNS-Name)	`props[].name="fqdn"`

Scan-Ergebnis	OSCAL-Feld
Hostname (Kurzform)	`props[].name="hostname"`
Scan-Zeitstempel	`props[].name="discovered-at"`

Jeder Eintrag erhält zusätzlich eine RFC-4122-konforme UUID, ein leeres `description`-Feld für die manuelle Ergänzung der Asset-Funktion und eine standardmäßig leere `responsible-parties`-Liste für die spätere Verantwortlichkeits-Zuordnung.

Anwendungsbeispiel:

Ausgangslage: Ein Beratungsunternehmen erstellt für einen Kunden einen SSP nach BSI IT-Grundschutz für ein neu in Betrieb genommenes Subnetz `192.168.50.0/24`. Die manuelle Inventarisierung würde mehrere Stunden dauern (Switch-Konfigurationen, ARP-Tabellen, manuelle Erfassung).

Mit dem Plugin: Der Berater öffnet den vorbereiteten SSP-Tab, löst einen Scan auf `192.168.50.0/24` aus (Dauer ca. 30 Sekunden), erhält eine Liste von 47 Hosts inklusive Hostnamen aus dem internen DNS, wählt 41 produktive Systeme aus (6 Test-VMs werden ausgeschlossen), übernimmt sie mit einem Klick in den SSP. Anschließend werden im SSP-Tab die `description`- und `responsible-parties`-Felder pro Host ergänzt — das Asset-Inventar ist in **unter 15 Minuten** aufgebaut, vollständig OSCAL-konform und auditfest.

Wiederholung beim nächsten Audit: Der gleiche Scan im selben Subnetz, mit aktivierter Option „Duplikate (gleiche IP) überspringen", erkennt **neue Hosts** (Asset-Drift) und macht sie sichtbar — die Basis für eine systematische Inventar-Pflege über den gesamten Lebenszyklus des Systems.

> **Screenshot-Hinweis 1:** *Discovery-Dialog: oben CIDR-Eingabefeld mit Validierungs-Hinweis („Maximal 4096 IPs"), Timeout-Spinbox, daneben „Scan starten"-Button im Primärfarbton; mittig die Ergebnis-Tabelle mit Spalten Checkbox, IP, MAC, Hostname, FQDN; unten Ziel-SSP-Dropdown und „In SSP übernehmen"-Button.*

> **Screenshot-Hinweis 2:** *Bestaetigungsdialog vor dem Scan mit deutlichem Warnsymbol und Text „ARP-Scan auf 192.168.50.0/24 (256 IPs) starten? Scannen Sie ausschließlich eigene Netzwerke!" — zwei Buttons „Abbrechen" und „Fortfahren" (Letzterer rot eingefärbt).*

> **Screenshot-Hinweis 3:** *SSP-Tab nach Übernahme: in der `inventory-items`-Liste 41 neu hinzugefügte Einträge (grün hervorgehoben), je Eintrag ausgeklappt mit Properties `ipv4-address`, `mac-address`, `fqdn`, `hostname` und der UUID; Tab-Titel mit `*` als Hinweis auf ungespeicherte Änderungen.*

Vorteile gegenüber externen Inventar-Tools:

- **Keine Datenexfiltration:** Scan-Ergebnisse bleiben vollständig in mjEdit — keine Übertragung an externe CMDBs, keine Cloud-Anbindung.
- **Direkter SSP-Bezug:** Anders als allgemeine Netzwerk-Scanner schreibt das Plugin nicht in CSV oder eine

Datenbank, sondern in das **richtige OSCAL-Feld** des **richtigen SSP-Dokuments**.

- **Reproduzierbarkeit:** Scan-Konfiguration (CIDR, Timeout) wird zusammen mit Zeitstempel im `inventory-item` - `prop` festgehalten — Auditoren können den Scan jederzeit wiederholen.
- **Schemavalidiert:** Vor dem Speichern des SSP wird das Dokument inkl. neuer Inventar-Items gegen das OSCAL 1.2.1-Schema validiert.

6.11 Plugin-Entwicklung

mjEdit bietet ein offenes Plugin-System für eigene Erweiterungen:

- Basis-Klasse `BasePlugin` mit vollständigem Lifecycle
- Hook-System für Event-basierte Integration
- Menü- und Toolbar-Integration
- Tab-Registrierung
- Konfigurationszugriff
- Beispiel-Plugin als Template

---

7. Template- und Snippet-System

mjEdit bringt ein zweistufiges Vorlagensystem mit, das sich konsequent durch die gesamte Anwendung zieht: **Templates** für vollständige Dateien und **Snippets** für wiederverwendbare Textbausteine. Beide Systeme arbeiten dateibasiert, sind sofort einsatzbereit und lassen sich ohne Programmierung erweitern – einfach eine neue Datei in den passenden Ordner legen.

7.1 Templates – Neue Dateien aus Vorlagen erstellen

Ein Template ist eine vollständige Vorlagedatei, die als Ausgangspunkt für ein neues Dokument dient. mjEdit kopiert das Template, benennt die Kopie um und öffnet sie im passenden Tab – fertig zum Bearbeiten.

Aufruf:

Weg	Beschreibung
Strg+N → „Aus Template"	Im Dialog „Neue Datei" die vierte Option wählen
Datei-Baum → Kontextmenü	Rechtsklick auf einen Ordner → „Neue Datei aus Template"
OSCAL-Tab → Kontextmenü	Element-spezifische Templates direkt im OSCAL-Editor
MCP-KI-Client	`template_list`, `template_create`, `editor_insert_template`

Der Template-Auswahl-Dialog (Strg+N → „Aus Template") ist als Dreistufen-Workflow aufgebaut:

1. **Template auswählen** – Links ein Baum mit allen Vorlagen aus `data/templates/`, rechts eine Live-Vorschau des Dateiinhalts. Ordner werden mit 🗀-Icons, Dateien nach Typ (🗎 JSON, 🖉 Markdown, 🗎 Text) dargestellt.
2. **Zielverzeichnis festlegen** – Ein zweiter Baum zeigt die Projektverzeichnisse. Per Kontextmenü lassen sich neue Unterordner anlegen, umbenennen oder löschen.
3. **Dateinamen vergeben** – Der Template-Name wird vorgeschlagen, automatisch mit dem Suffix `_neu` versehen. Eine farbige Anzeige signalisiert, ob der Name bereits vergeben ist (rot) oder frei (grün).

Intelligente Dateierkennung: Nach dem Erstellen öffnet mjEdit die neue Datei automatisch im richtigen Tab:

- `.md`-Dateien → Markdown-Tab
- qFORM-Dateien (JSON-Array mit `"type": "qFORM"`) → Formular-Tab
- OSCAL-Dateien (mit `catalog`, `profile`, `ssp` etc.) → OSCAL-Tab

- Alle anderen → Text-Tab

Als Template speichern: Jeder Editorinhalt lässt sich per Rechtsklick → „Als Template speichern" direkt in die Template-Bibliothek aufnehmen. Der Dialog bietet Verzeichnisnavigation, Dateiname und Typ-Auswahl (.json/.md/.txt) sowie eine JSON-Validierung beim Speichern.

7.2 OSCAL-Templates – Kontextbezogene Filterung nach Schlüsselwörtern

Das Herzstück des Template-Systems für OSCAL-Anwender ist die **strikte Dateinamen-Filterung**. Wenn Sie in einem OSCAL-Tab ein neues Element per Template hinzufügen, zeigt mjEdit **nur** die Templates an, deren Dateiname zum aktuellen Kontext passt – nicht die gesamte Bibliothek.

Das Prinzip: Der Array-Typ (z. B. `roles`, `parties`, `inventory-items`) wird mit den Dateinamen im Template-Verzeichnis abgeglichen. Nur Dateien, die den Suchbegriff im Namen enthalten, erscheinen im Dialog.

Beispiel:

Kontext im OSCAL-Tab	Angezeigte Templates
Neue **Rolle** hinzufügen	`roles-template-users.json`, `roles-template23.json`
Neues **Inventory-Item**	`inventory-item-basic-template.json`, `inventory-item-server-template.json`, `inventory-item-network-template.json`, `inventory-item-software-template.json`
Neue **Partei** (Person/Organisation)	`parties-person-template.json`, `parties-organization-template.json`
Neuer **Information-Type**	`information-types-fips199-high.json`, `information-types-grundschutz-personenbezogen.json`, `information-types-nist-pii.json`
Neues **Security-Impact-Level**	`security-impact-level-grundschutz.json`, `security-impact-level-iso27001.json`, `security-impact-level-nist-800-53.json`

Warum diese Filterung wichtig ist: OSCAL kennt über 25 verschiedene Array-Typen (Rollen, Parteien, Standorte, Controls, Implementierungen, Protokolle, Properties u. v. m.). Ohne Filterung müsste der Anwender in einer Liste von 73+ Templates das richtige finden. Die Dateinamen-Konvention (`{typ}-template.json`) macht die Zuordnung automatisch – und erweiterbar: Eigene Templates folgen einfach der gleichen Namenskonvention.

Die folgende Tabelle zeigt alle vom Template-System unterstützten Array-Typen mit ihrer Bedeutung im OSCAL-Kontext:

Metadata-Arrays (in allen OSCAL-Dokumenttypen)

Array-Typ	OSCAL-Kontext	Beschreibung
`roles`	Metadata	Rollen wie „System-Owner", „Authorizing-Official" oder „ISB" – definieren Verantwortlichkeiten im Sicherheitsprozess
`parties`	Metadata	Organisationen und Personen, die am System beteiligt sind (Betreiber, Dienstleister, Auditoren)
`locations`	Metadata	Physische Standorte (Rechenzentren, Büros, Cloud-Regionen) mit Adress- und Kontaktdaten
`responsible-parties`	Metadata	Zuordnung von Parteien zu Rollen – wer füllt welche Rolle im konkreten Fall aus
`document-ids`	Metadata	Dokumenten-Identifikatoren (DOI, interne IDs) zur eindeutigen Referenzierung
`revisions`	Metadata	Änderungshistorie des Dokuments mit Datum, Autor und Beschreibung

Array-Typ	OSCAL-Kontext	Beschreibung

Component-Definition und Implementierung

Array-Typ	OSCAL-Kontext	Beschreibung
`props`	Alle Ebenen	Schlüssel-Wert-Eigenschaften (Properties) für beliebige Metadaten an jedem OSCAL-Objekt
`links`	Alle Ebenen	Verweise auf externe Ressourcen, Dokumente oder andere OSCAL-Objekte per URI
`responsible-roles`	Components, SSP	Welche Rollen für eine Komponente oder Implementierung verantwortlich sind
`protocols`	Components	Netzwerk-Protokolle (HTTPS, SSH, LDAP) mit Port-Bereichen, die eine Komponente nutzt
`control-implementations`	Components	Gruppen von implementierten Controls, die eine Komponente erfüllt – das Kernstück der Compliance-Aussage
`implemented-requirements`	SSP, Components	Einzelne implementierte Anforderungen mit Beschreibung, wie ein Control umgesetzt wird
`set-parameters`	Profile, SSP	Parameterwerte für Controls setzen (z. B. „Passwortlänge = 12 Zeichen“)
`statements`	SSP	Detaillierte Umsetzungsbeschreibungen pro Control-Statement (a, b, c, …)

System Security Plan (SSP)

Array-Typ	OSCAL-Kontext	Beschreibung
`inventory-items`	SSP	IT-Inventar: Server, Workstations, Netzwerkgeräte, Software – jedes Einzelsystem mit Attributen
`information-types`	SSP	Klassifizierung der verarbeiteten Daten nach Schutzbedarf (FIPS 199, BSI-Grundschutz, NIST)
`security-impact-level`	SSP	Sicherheitseinstufung des Gesamtsystems (Vertraulichkeit, Integrität, Verfügbarkeit)
`system-status`	SSP	Betriebsstatus: operational, under-development, under-major-modification, disposition
`users`	SSP	Benutzergruppen des Systems mit Zugriffsrechten und Berechtigungsstufen
`components`	SSP	Im System eingesetzte Komponenten (Hardware, Software, Services, Policies)
`by-components`	SSP	Zuordnung: Welche Komponente implementiert welches Control – die Compliance-Matrix
`leveraged-authorizations`	SSP	Referenzen auf bereits autorisierte Systeme (z. B. Cloud-Provider mit eigenem ATO)

Array-Typ	OSCAL-Kontext	Beschreibung

Assessment und Plan of Action

Array-Typ	OSCAL-Kontext	Beschreibung
`activities`	Assessment Plan	Geplante Prüfaktivitäten: Interviews, Tests, Dokumentenprüfungen
`findings`	Assessment Results, POAM	Festgestellte Abweichungen und Schwachstellen aus dem Audit
`observations`	Assessment Results	Einzelne Beobachtungen (Evidenzen) aus der Prüfung
`risks`	Assessment Results, POAM	Identifizierte Risiken mit Bewertung und Einstufung
`poam-items`	POAM	Maßnahmen mit Fristen und Verantwortlichen zur Behebung von Findings

> **Tipp:** Eigene Templates für neue Array-Typen sind einfach zu erstellen: Eine JSON-Datei mit dem Typ-Schlüsselwort im Dateinamen (z. B. `inventory-item-firewall-template.json`) im Ordner `data/templates/OSCAL/elements/` ablegen – der Dialog findet sie automatisch.

Zusätzliche Features des OSCAL-Template-Dialogs:

- **Rekursive Suche** in allen Unterverzeichnissen von `data/templates/OSCAL/`
- **Template-Mapping** mit 25+ vordefinierten Zuordnungen für Standard-Typen
- **Vorschau** mit automatischer Bereinigung von `_comment`-Feldern
- **Pfad-Persistenz** – der zuletzt gewählte Ordner wird in der Konfiguration gespeichert
- **Bearbeiten nach Einfügen** – optionale Checkbox öffnet das eingefügte Element direkt im Bearbeitungs-dialog
- **UUID-Generierung** – Platzhalter-UUIDs im Template werden automatisch durch gültige UUIDv4-Werte ersetzt

Verfügbare OSCAL-Tabs mit Template-Unterstützung:

OSCAL-Tab	Template-Funktionen
Katalog	Neues Control, neue Gruppe, Subcontrol aus Template

OSCAL-Tab	Template-Funktionen
Komponenten-Definition	Neue Komponente, neue Eigenschaft (Property) aus Template
System Security Plan	Inventory-Items, Information-Types, Security-Impact-Level, System-Status
Assessment Plan	Array-Elemente (Rollen, Parteien, Locations, Activities)
Assessment Results	Array-Elemente (Findings, Observations, Risks)
Plan of Action	Array-Elemente (Findings, Risks, Poam-Items)
Profil	Imports, Alters, Parameter-Settings

7.3 Mitgelieferte Template-Bibliothek

mjEdit wird mit über **73 OSCAL-Templates** ausgeliefert, die alle auf OSCAL v1.2.1 Konformität geprüft sind:

Kategorie	Anzahl	Beispiele
Dokumenttypen	6	Katalog, Profil, SSP, Assessment Plan/Results, POAM
Komponenten	13	Software, Hardware, Service, Netzwerk, Policy, Guidance, Infrastruktur, Plan, Process, Standard, System, Validation, Interconnection
Elemente	43	Controls, Gruppen, Rollen, Parteien, Inventory-Items, Information-Types, Security-Impact-Levels, Properties, Links, Protokolle, Statements
Metadaten	5	Document-IDs, Locations, Parties, Responsible-Parties, Roles
Mapping	1	Mapping-Collection für Framework-Zuordnungen

Zusätzlich gibt es Templates für **JSON** (6), **Markdown** (1), **qFORM** (2) und **QC-Scripts** (1).

7.4 Snippets – Textbausteine an der Cursorposition einfügen

Ein Snippet ist ein wiederverwendbarer Textbaustein, der an der aktuellen Cursorposition eingefügt wird – ideal für häufig benötigte JSON-Strukturen, Markdown-Fragmente oder OSCAL-Elemente.

Aufruf:

Weg	Beschreibung
Strg + <	Snippet-Dialog öffnen (konfigurierbarer Shortcut)
Rechtsklick → „Snippet einfügen"	Im Text- oder Markdown-Tab
MCP-KI-Client	`markdown_insert_snippet`

Der Snippet-Dialog zeigt links den Verzeichnisbaum aus `data/snippets/` mit automatisch expandierten Ordnern und rechts eine Live-Vorschau des ausgewählten Snippets. Navigation per Pfeiltasten, Einfügen per Enter oder Doppelklick. Der Dialog merkt sich die zuletzt verwendete Datei (`sys_snippet_last_file`).

Mitgelieferte Snippets:

Kategorie	Snippets	Beschreibung
JSON	`adresse_leer.json`, `person_leer.json`	Leere Strukturvorlagen
Markdown	`markdown_tabelle.md`, `markdown_vorlage.md`	Tabellen, Seitenvorlagen
qFORM	5 Snippets	Neue Felder, Anmerkungen, Medikamente, Patientendaten

Als Snippet speichern: Per Rechtsklick → „Als Snippet speichern" lässt sich jeder markierte oder vollständige Editorinhalt direkt als Snippet ablegen. Der Dialog unterstützt Verzeichnisnavigation, neue Ordner erstellen und JSON-Validierung.

7.5 Template vs. Snippet – Wann nutze ich was?

Kriterium	Template	Snippet
Zweck	Neue Datei erstellen	Textbaustein einfügen
Granularität	Vollständige Datei	Fragment / Ausschnitt
Ergebnis	Neue Datei im Dateisystem	Text an Cursorposition
Typischer Einsatz	Neues OSCAL-Dokument, neue Konfiguration	JSON-Objekt, Markdown-Tabelle, OSCAL-Element
Speicherort	`data/templates/`	`data/snippets/`

Kriterium	Template	Snippet
Shortcut	Strg+N → „Aus Template"	Strg+<

7.6 Eigene Templates und Snippets erstellen

Das System ist vollständig **dateibasiert** – keine Datenbank, kein Registry, kein Build-Prozess. Neue Vorlagen erstellen Sie in drei Schritten:

1. **Datei erstellen** – Legen Sie eine `.json`-, `.md`- oder `.txt`-Datei im passenden Unterordner ab
2. **Namenskonvention beachten** – Für OSCAL-Templates: `{typ}-{variante}-template.json` (z. B. `inventory-item-router-template.json`)
3. **Fertig** – Das Template erscheint sofort im Dialog, für OSCAL-Templates auch automatisch im kontextbezogenen Filter

Alternativ: Per Rechtsklick im Editor → „Als Template speichern" oder „Als Snippet speichern" – mjEdit erledigt Verzeichnisauswahl, Benennung und Speicherung über den Dialog.

Verzeichnisstruktur:

```
data/templates/                    data/snippets/
├── JSON/           # JSON-Vorlagen  ├── JSON/        # JSON-Bausteine
├── MARKDOWN/       # MD-Vorlagen    ├── MARKDOWN/    # MD-Bausteine
├── OSCAL/          # OSCAL-Vorlagen ├── OSCAL/       # OSCAL-Fragmente
│   ├── components/  # 13 Typen      ├── QFORM/       # qFORM-Felder
│   └── elements/    # 43 Elemente   └── QSCRIPT/     # QC-Script-Bausteine
├── QFORM/          # Formulare
└── QSCRIPT/        # QC-Scripts
```

7.7 KI-Integration: Templates per MCP

Über den MCP-Server stehen Templates auch KI-Clients zur Verfügung:

MCP-Tool	Funktion
`template_list`	Alle verfügbaren Templates nach Kategorie auflisten
`template_create`	Neues Template aus KI-generiertem Inhalt erstellen
`template_delete`	Template entfernen
`template_export` / `template_import`	Templates zwischen Installationen austauschen
`editor_insert_template`	Template per Fuzzy-Name-Suche in den Editor einfügen
`oscal_model_create`	OSCAL-Dokument mit Template-Support erstellen

MCP-Tool	Funktion
`oscal_model_list_templates`	Verfügbare OSCAL-Templates auflisten

MCP-Ressourcen für Snippets:

Ressource	Beschreibung
`mjedit://snippets/json`	JSON-Code-Snippets
`mjedit://snippets/markdown`	Markdown-Snippets
`mjedit://snippets/oscal`	OSCAL-Snippets
`mjedit://snippets/qform`	qFORM-Snippets
`mjedit://snippets/qscript`	QC-Script-Vorlagen
`mjedit://templates/list`	Alle verfügbaren Templates

8. Sicherheit und Qualität

8.1 Anwendungssicherheit

Maßnahme	Detail
RestrictedPython-Sandbox	Benutzer-Code läuft ohne System-Zugriff
Pfad-Traversal-Schutz	`../`-Angriffe werden blockiert
Null-Byte-Blockierung	Keine Null-Bytes in Dateipfaden
Datei-Validierung	Inhalte werden vor dem Öffnen geprüft
Keine verbotenen Funktionen	`eval()`, `exec()`, `os.system()`, `pickle` sind verboten
MCP Rate-Limiting	Sliding-Window gegen Überlastung
MCP Path-Whitelist	Nur konfigurierte Verzeichnisse zugänglich
Audit-Logging	Alle MCP-Operationen werden protokolliert

8.2 Code-Qualität

Metrik	Wert
Bandit Security-Audit	0 Medium/High Issues
Implementierte User Stories	900+ Features
Automatische Tests	1350+ Tests (alle grün)
Dokumentation	Vollständig (DE + EN)
i18n-Abdeckung	100% aller GUI-Texte

8.3 OSCAL-Datenqualität

- Pydantic-Modelle mit Runtime-Validierung bei jeder Erstellung und Bearbeitung
- JSON-Schema-Validierung gegen offizielle NIST OSCAL v1.2.1 Schemas
- UUID-Format-Prüfung (RFC 4122) mit Auto-Korrektur
- Referenz-Integritätsprüfung (UUID/Href/Control-ID)
- Duplikatschutz bei Inventar-Komponenten-Verknüpfungen

9. Unterstützte KI-Clients

mjEdit stellt seinen MCP-Server über zwei Transport-Protokolle bereit: **STDIO** (direkte Prozesskommunikation) und **SSE** (HTTP Server-Sent Events). Je nach KI-Client wird einer oder beide Transporte unterstützt.

9.1 Vollständige Unterstützung (88 Tools + 22 Ressourcen + 15 Prompts)

Client	Transport	Konfiguration
Claude Desktop	STDIO	`claude_desktop_config.json`
Cursor IDE	STDIO	MCP-Server in Settings
VS Code (GitHub Copilot)	STDIO	`.vscode/mcp.json`

9.2 AnythingLLM – Open-Source-Referenz-Client für MCP

AnythingLLM nimmt unter den unterstützten KI-Clients eine besondere Stellung ein: Als **Open-Source-Plattform** (MIT-Lizenz) mit vollständiger MCP-Unterstützung ist AnythingLLM der **empfohlene Referenz-MCP-HOST-Client** für mjEdit – insbesondere für Organisationen, die Wert auf Datensouveränität, Self-Hosting und RAG-Integration legen.

Eigenschaft	Detail
Lizenz	MIT (Open Source) – kostenlos nutzbar, auch kommerziell
MCP-Transport	SSE (HTTP) – ideal für Docker, Netzwerk-Setups und Multi-User
Deployment	Desktop-App (Windows, macOS, Linux) oder Docker-Container
RAG-Wissensbasis	Eigene Compliance-Dokumente einbetten und mit MCP-Tools kombinieren
LLM-Auswahl	Frei wählbar: OpenAI, Anthropic Claude, Ollama (lokal), LM Studio, Azure u.v.m.
Offline-Fähigkeit	Ja – mit lokalen Modellen (z.B. Ollama) vollständig ohne Internet nutzbar
Multi-User	Team-Workspaces mit Rechteverwaltung

Warum AnythingLLM als Referenz-Client?

- **Open Source + Self-Hosted:** Keine Abhängigkeit von Cloud-Diensten, volle Kontrolle über Daten und Infrastruktur
- **RAG + MCP-Synergie:** Als einziger unterstützter Client kombiniert AnythingLLM eine RAG-Wissensbasis (Ihre eigenen Compliance-Dokumente) mit den 88 MCP-Tools von mjEdit – die KI kennt Ihre Organisation und kann gleichzeitig OSCAL-Dokumente erstellen
- **Flexibles LLM-Backend:** Von GPT-4o über Claude bis zu kostenlosen lokalen Modellen via Ollama – Sie entscheiden, welches Modell Ihre Compliance-Daten verarbeitet
- **Docker-Integration:** Einfaches Deployment als Container im internen Netzwerk, SSE-Transport über `host.docker.internal:8765`

Konfiguration:

Variante	MCP-URL	Besonderheit
AnythingLLM Desktop	`http://localhost:8765/sse`	Direkte Verbindung auf demselben Rechner
AnythingLLM Docker	`http://host.docker.internal:8765/sse`	Bridge-Netzwerk zum Host-System

```
{
  "mcpServers": {
    "mjEdit": {
      "url": "http://localhost:8765/sse",
      "transport": "sse"
    }
  }
}
```

> **Tipp:** AnythingLLM unterstützt Single-Call-MCP. Nutzen Sie das mjEdit-Tool `execute_steps`, um bis zu 20 Tool-Aufrufe in einem einzigen Request zu bündeln – das spart Token und beschleunigt komplexe Workflows erheblich.

Für eine ausführliche Beschreibung der RAG-Integration und praxisnahe Use Cases siehe AnythingLLM.

9.3 Beispiel-Konfiguration (Claude Desktop)

```
{
  "mcpServers": {
    "mjedit": {
      "command": "python",
      "args": ["C:/Pfad/zu/mjEdit/src/main.py", "--mcp-stdio"]
```

```
    }
  }
}
```

---

10. Technische Daten

10.1 Systemvoraussetzungen

Anforderung	Minimum
Python	3.10+
Betriebssystem	Windows 10/11, Linux (Ubuntu 22.04+)
RAM	512 MB (4 GB für große Kataloge empfohlen)
Festplatte	200 MB + Daten
GUI-Framework	PySide6 (Qt 6)

10.2 Performance-Kennzahlen

Operation	Typische Dauer
Anwendungsstart	2–3 Sekunden (Lazy-Loading)
JSON öffnen (10 KB)	< 100 ms
JSON öffnen (1 MB)	1–2 Sekunden
JSON formatieren	< 500 ms
Suche (10.000 Einträge)	< 200 ms
OSCAL-Katalog laden	2–5 Sekunden
Profil-Auflösung	1–3 Sekunden
Baum-Filter	< 100 ms

10.3 Performance-Optimierungen

Schwellenwert	Optimierung
100 KB	Performance-Modus aktiviert
200 KB	Rechtschreibprüfung deaktiviert
1 MB	Syntax-Highlighting deaktiviert
1.000 Controls	Fortschrittsanzeige für OSCAL-Kataloge

10.4 Vorinstallierte Compliance-Daten

Datensatz	Umfang
BSI IT-Grundschutz	111 Gruppen, 2.128 Controls, 444 Parts, 817 Sub-Parts
NIST SP 800-53	6 Familien, 468 Controls
Grundschutz++ Profil	489 Controls

10.5 Deployment

Plattform	Format
Windows	EXE via PyInstaller (inklusive Daten)
Linux	AppImage

10.6 Lizenz

AGPL-3.0 – Freie Nutzung, Modifikation und Weitergabe unter den Bedingungen der GNU Affero General Public License v3.0.

11. Roadmap – Geplante Funktionen und Erweiterungen

Funktion	Beschreibung	Priorität
Bugfixes und Stabilitätsverbesserungen	Kontinuierliche Fehlerbehebung und Optimierung der Anwendung	Sehr Hoch
Migration und Spezialisierung von OSCAL-Dialoge	Verbesserte Dialoge für die Bearbeitung von OSCAL-Elementen (Controls, Gruppen, Komponenten) mit erweiterten Funktionen wie Standard-GUI-Formulare, Inline-Bearbeitung und Validierung	Sehr Hoch
Erweiterung der Template- und Text-Snippet-Bibliothek	Weitere vorgefertigte Templates und Snippets für häufige Anwendungsfälle	Sehr Hoch
Optimierung OSCAL-MAPPING	Optimierung zur Selektion und Verarbeitung der OSCAL-Controls im Mapping-Prozess	Sehr Hoch
GIT-Integration	Gruppenarbeit im Projektteam	Hoch
Erweitern und optimieren der MCP-API	Weitere Tools, Resourcen und Prompts für die Nutzung durch KI-Hosts	Hoch
Aktiver KI-REST-API Rückkanal zu AnythingLLM	Ermöglicht die aktive Kommunikation mit AnythingLLM über REST-API vom MCP-SERVER zu MCP-HOST	Hoch
JIRA-Integration	Integration mit JIRA für Aufgaben- und Projektmanagement	Mittel
Confluence-Integration	Dokumentation und Wissensmanagement mit Confluence	Mittel
ServiceNow-Integration	Anbindung an ServiceNow für ITSM-Workflows	Mittel

Funktion	Beschreibung	Priorität
Dic2Pydantic Migration von OSCAL-TABs	Migration von OSCAL-TABs zu Pydantic-Modellen für bessere Validierung und Typensicherheit	Niedrig
Integration mit kommerziellen 42tec-Tools	Anbindung an kommerzielle 42tec-Tools für erweiterte Funktionalitäten zur operativen Security und dem Projektmanagement	Niedrig
Migration von Python nach RUST-Programmierung	Verbesserung des Deployments und der Performance	Sehr Niedrig