WolfISMS REST-API

Anleitung für die Anbindung externer Systeme

Version 4.00 · Stand 01.11.2026

Anleitung für die Anbindung externer Systeme an WolfISMS. Die API stellt sechs Bereiche bereit – **Incidents** (Sicherheitsvorfälle), **Maßnahmen**, **Assets**, **Risiken**, **Lieferanten** und **Schwachstellen** – jeweils zum Anlegen, Lesen und zum Quittieren des Synchronisationsstands.

Typische Einsatzfälle: ein SIEM meldet Vorfälle automatisch nach WolfISMS, ein Ticketsystem holt sich offene Maßnahmen ab, eine CMDB gleicht den Asset-Bestand ab, ein Einkaufssystem meldet neue Lieferanten, ein Schwachstellenscanner liefert seine Funde ein, oder ein Reporting-Werkzeug liest Risiken und Bestand für den Managementbericht.

Inhalt

1. Voraussetzungen
2. Authentifizierung
3. Grundlagen
4. Incidents
5. Maßnahmen
6. Assets
7. Risiken
8. Lieferanten
9. Schwachstellen
10. Synchronisation
11. Testen mit curl
12. Hinweise für den Betrieb

1. Voraussetzungen

API-Key anlegen

Jeder Aufruf braucht einen API-Key. Diesen legt ein Administrator in der Anwendung an:

Administration → API-Keys (`/admin/api_keys`)

1. Beschreibung eintragen (z. B. „SIEM Produktivsystem") und anlegen.
2. Der Schlüssel wird **genau einmal** angezeigt – 64 Hexadezimalzeichen. Sofort kopieren und im aufrufenden System sicher hinterlegen.
3. In der Datenbank speichert WolfISMS nur den SHA-256-Hash. Ein verlorener Schlüssel lässt sich nicht wiederherstellen, sondern nur ersetzen.

Bestehende Schlüssel können in derselben Maske deaktiviert (`aktiv=0`) oder gelöscht werden. Ein deaktivierter Schlüssel wird sofort abgewiesen – das ist der schnellste Weg, eine Anbindung stillzulegen.

Basis-URL

```
https://<ihr-host>/api/v1/
```

Nur über HTTPS betreiben. Der API-Key wandert im Klartext im Header und wäre über HTTP mitlesbar.

2. Authentifizierung

Der Schlüssel gehört in den Header `X-API-Key`:

```
curl -H "X-API-Key: 8f3c…ab12" "https://isms.example.com/api/v1/incidents"
```

Fehlt der Header oder ist der Schlüssel unbekannt bzw. deaktiviert, antwortet die API mit **401**:

```
{"error": "Ungültiger oder fehlender API-Key"}
```

Ein CSRF-Token wird **nicht** benötigt – die REST-API ist davon ausgenommen und sichert sich allein über den Schlüssel. Eine Anmeldung mit Benutzername und Kennwort findet nicht statt; die API kennt keine Sitzung.

Rate-Limit

Die REST-API unterliegt **keiner Drosselung**. Sync-Clients dürfen im Sekundentakt lesen und quittieren. Wer die API aus dem Internet erreichbar macht, sollte eine Begrenzung im vorgelagerten Reverse Proxy setzen (z. B. nginx `limit_req` nur für `/api/`).

3. Grundlagen

Thema	Festlegung
Format	JSON, UTF-8
Content-Type	`application/json` (wird beim Lesen des Bodys toleriert, wenn er fehlt)
Zeitangaben	Datum `YYYY-MM-DD`, Zeitstempel `YYYY-MM-DD HH:MM:SS`
Sprache der Werte	Status, Priorität und Kategorie werden **deutsch** gespeichert, auch wenn die Oberfläche auf Englisch steht
Umlaute	in Feldnamen bewusst vermieden (`geaendert`, `prioritaet`)

Mehrere Besitzer

Seit v3.89 kann ein Datensatz mehrere Besitzer haben. `besitzer` bleibt der **Hauptbesitzer** (eine Benutzer-ID), `weitere_besitzer` ist die Liste der übrigen Besitzer – ebenfalls Benutzer-IDs, ohne den Hauptbesitzer. Das gilt für alle sechs Bereiche:

Stelle	Verhalten
Liste und Einzelabruf	liefern `weitere_besitzer` (leere Liste, wenn es keine gibt); mit `?besitzer_name=1` bzw. im Einzelabruf kommen die Namen als `weitere_besitzer_namen`
Filter `?besitzer=`	trifft Datensätze, bei denen die ID Haupt- **oder** weiterer Besitzer ist
POST	`weitere_besitzer` optional als Liste (`[4, 9]`), einzelne Zahl oder kommagetrennte Zeichenkette; unbekannte IDs ergeben 400
PATCH	`weitere_besitzer` ersetzt die Liste vollständig, `[]` oder `null` leert sie – bei Maßnahmen, Assets, Risiken, Lieferanten und Schwachstellen, nicht bei Incidents

Rechte, Aufgaben und Erinnerungsmails behandeln alle Besitzer gleich; die Oberfläche zeigt sie durch Semikolon getrennt. Namen liefern Maßnahmen, Assets, Risiken und Lieferanten; Incidents und Schwachstellen kennen nur die IDs.

Statuscodes

Code	Bedeutung
200	Erfolg (GET, PATCH/PUT)
201	Datensatz angelegt (POST)
400	Eingabefehler – `error` nennt alle beanstandeten Felder
401	API-Key fehlt, ist unbekannt oder deaktiviert
403	Das Modul ist abgeschaltet (Administration → Module)
404	Nummer nicht gefunden
409	Nummer ist bereits vergeben
500	Datenbankfehler (Details stehen im Server-Log, nicht in der Antwort)

Jede Fehlerantwort hat dieselbe Form:

```
{"error": "'prio' ist ein Pflichtfeld (auch 'prioritaet' möglich)"}
```

Mehrere Fehler werden mit `;` verkettet, sodass ein einziger Aufruf alle Beanstandungen zurückmeldet.

Feldlängen

Textfelder werden vor dem Schreiben gegen die Spaltenbreite der Datenbank geprüft. Ein zu langer Wert kommt als **400** zurück und nennt Grenze und übergebene Länge; er läuft nicht mehr als Datenbankfehler (500) auf.

Bereich	Feld	max. Zeichen
Incidents	`nummer` · `titel`	64 · 256
Maßnahmen	`nummer` · `name` · `abteilung` · `prioritaet`	64 · 128 · 128 · 20
Assets	`nummer` · `name` · `version` · `abteilung` · `datenhaltung`	16 · 64 · 64 · 64 · 100
Assets	`betroffene_personen` · `art_pbd` (fertige Liste)	je 500
Risiken	`nummer` · `name` · `abteilung`	64 · 256 · 64
Lieferanten	`nummer` · `name` · `ansprechpartner` · `email`	64 · 64 · 256 · 64
Schwachstellen	`nummer` · `titel` · `quelle` · `status`	20 · 200 · 100 · 50

Die Freitextfelder `beschreibung`, `businessimpact`, `massnahme` und `ticketnummer` fassen 65.535 **Bytes** – nicht Zeichen: ein Umlaut belegt in UTF-8 zwei Bytes, ein Emoji vier. Die `beschreibung` einer Maßnahme ist mit 16 MB deutlich größer.

Was der Body sein muss

Der Body ist ein JSON-**Objekt**. Ein Array, eine nackte Zeichenkette oder eine Zahl wird mit 400 abgewiesen, ebenso ein Objekt oder eine Liste in einem Textfeld (`{"titel": {"a": 1}}`). Zahlenfelder nehmen keine Werte an, die die Spalte nicht speichern kann – `kosten` etwa höchstens 9.999.999.999,99 und weder `NaN` noch `Infinity`.

4. Incidents

Methode	Pfad	Zweck
POST	`/api/v1/incidents`	Vorfall anlegen
GET	`/api/v1/incidents`	Liste mit Filtern und Paginierung
GET	`/api/v1/incidents/<nummer>`	einzelner Vorfall
PATCH / PUT	`/api/v1/incidents/<nummer>`	Status und Sync-Status schreiben

Felder beim Anlegen

Feld	Pflicht	Werte
`nummer`	ja	eindeutig, z. B. `INC-00042`
`titel`	ja	Kurzbeschreibung
`prio`	ja	`niedrig` · `mittel` · `hoch` · `kritisch`
`beschreibung`	nein	Fließtext
`status`	nein	Vorgabe `Gemeldet`
`kategorie`	nein	Vorgabe `ohne`
`besitzer`	nein	Benutzer-ID des Hauptbesitzers, wird geprüft; Vorgabe `0` (keiner)
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`sync` / `synctime`	nein	siehe Abschnitt 10

Fehlen `sync` und `synctime`, legt die API den Vorfall als **synchronisiert** an: `sync = 1`, `synctime` = Zeitpunkt des Aufrufs. Das Anlegen über die Schnittstelle ist der Datenaustausch mit dem angebundenen System – ein weiteres Quittieren ist nicht nötig. Wer den Vorfall trotzdem noch abholen lassen will, gibt `"sync": 0` mit, wer ihn ganz ausnehmen will `"sync": null`.

Für `prio` werden auch die Schreibweisen `prioritaet`, `priorität` und `prioritat` angenommen.

Status: `ohne` · `Gemeldet` · `in Bewertung` · `Bestätigt` · `Verworfen/Fehlalarm` · `in Behandlung` · `Überwacht` · `Abgeschlossen` · `Nachbearbeitung` · `Geschlossen`

Kategorie: `ohne` und die Kategorien, die der Administrator unter `/admin/kategorien` (Bereich Incidents) pflegt. Ausgeliefert werden `Malware` · `Phishing` · `Datenabfluss` · `DoS` · `Fehlkonfiguration` · `Datenschutzverletzung` · `Sonstiges`; seit v3.92 prüft die API gegen den aktuellen Katalog, eine neu angelegte Kategorie ist also sofort zulässig. Gespeichert und verglichen wird der deutsche Name.

Groß- und Kleinschreibung zählt bei Status und Kategorie; `prio` wird automatisch klein geschrieben.

Beispiel: Vorfall anlegen

```
curl -X POST "https://isms.example.com/api/v1/incidents" -H "X-API-Key: $KEY" -H
"Content-Type: application/json" -d '{"nummer":"INC-00042","titel":"Phishing-Welle
Buchhaltung","prio":"hoch","status":"Gemeldet","kategorie":"Phishing","beschreibung":"1
4 Mails mit gefälschtem Lieferantenkonto.","sync":0}'
```

Antwort **201**:

```
{
  "id": 512,
  "nummer": "INC-00042",
  "sync": 0,
  "synctime": null,
  "message": "Incident erfolgreich angelegt"
}
```

Von der API gesetzte Vorgaben: alle Datumsfelder (`erkannt`, `gemeldet`, `beginn`, `beendet`, `geschlossen`, `erstellt`, `geändert`) auf das heutige Datum, die Personenfelder auf 0 (kein Benutzer), `typ`/`ursache`/`quelle` auf `ohne`, die Kennzeichen DSGVO/NIS2/CRA und `archiviert` auf 0. Diese Angaben pflegt anschließend die Oberfläche.

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/incidents?
status=Gemeldet,in%20Behandlung&prio=hoch,kritisch&limit=20&sort=erstellt&order=desc"

{
  "total": 37,
  "count": 20,
  "limit": 20,
  "offset": 0,
  "items": [
    {
      "id": 512,
      "nummer": "INC-00042",
      "titel": "Phishing-Welle Buchhaltung",
      "prio": "hoch",
      "status": "Gemeldet",
      "kategorie": "Phishing",
```

```
      "typ": "ohne",
      "ursache": "ohne",
      "quelle": "ohne",
      "erkannt": "2026-08-14",
      "gemeldet": "2026-08-14",
      "beginn": "2026-08-14",
      "beendet": "2026-08-14",
      "geschlossen": "2026-08-14",
      "erstellt": "2026-08-14",
      "geaendert": "2026-08-14",
      "archiviert": false,
      "dsgvo": false,
      "nis2": false,
      "cra": false,
      "ticketnummer": "",
      "sync": 0,
      "synctime": null,
      "besitzer": 17,
      "weitere_besitzer": [23]
    }
  ]
}
```

Der Beschreibungstext fehlt bewusst – er kann sehr lang werden. Mit `?beschreibung=1` wird er mitgeliefert.

Einzelner Vorfall

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/incidents/INC-00042"
```

Die Einzelabfrage liefert den Kerndatensatz mit `id`, `nummer`, `titel`, `beschreibung`, `prio`, `status`, `kategorie`, `erstellt`, `sync`, `synctime`, `besitzer` und `weitere_besitzer` – die Beschreibung also immer, dafür ohne die Zusatzfelder der Liste (Datumsfelder, DSGVO/NIS2/CRA, Ticketnummer). Wer den vollständigen Satz zu einem einzelnen Vorfall braucht, fragt ihn über die Liste ab: `?q=INC-00042&beschreibung=1`.

Eine unbekannte Nummer beantwortet die API mit **404**.

Vorfall ändern

`PATCH` (oder `PUT`) schreibt den **Status** und den Synchronisationszustand eines vorhandenen Vorfalls. Alle drei Felder sind einzeln oder gemeinsam angebbar, mindestens eines muss im Body stehen:

Body	Wirkung
`{"status": "in Behandlung"}`	Status setzen; `sync` wird auf 1 gesetzt, `synctime` auf jetzt
`{"status": "Geschlossen", "sync": 0}`	Status setzen, Sync-Zustand ausdrücklich selbst führen
`{"sync": 1}`	nur quittieren, Zeitpunkt = jetzt
`{"sync": 0}`	zurücksetzen; `synctime` bleibt stehen
`{"sync": null}`	von der Synchronisation ausnehmen
`{"synctime": null}`	nur den Zeitpunkt löschen

```
curl -X PATCH "https://isms.example.com/api/v1/incidents/INC-00042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"status": "in Behandlung"}'
```

Antwort **200**:

```
{
  "nummer": "INC-00042",
  "status": "in Behandlung",
  "sync": 1,
  "synctime": "2026-08-21 08:15:37",
  "message": "Vorfall aktualisiert"
}
```

`status` muss einer der oben genannten Werte sein, sonst antwortet die API mit 400. Mit dem Status führt die API `geändert` (heutiges Datum) mit; als Änderer wird `0` für die Schnittstelle eingetragen. Jedes andere Feld im Body – etwa `titel` – führt zu einem 400 mit dem Hinweis, welche Namen unbekannt sind.

Sync wird beim Schreiben mitgesetzt: Enthält der Body weder `sync` noch `synctime`, trägt die API `sync = 1` und den aktuellen Zeitpunkt ein. Führt der Aufruf eines der beiden Felder selbst, bleibt die Entscheidung beim Client – `{"synctime": null}` löscht also weiterhin nur den Zeitpunkt.

Listen-Parameter

Parameter	Wirkung
`status`, `prio`, `kategorie`	mehrfach (`?status=A&status=B`) oder kommagetrennt (`?status=A,B`)
`besitzer`	Benutzer-ID, mehrfach; trifft Haupt- und weitere Besitzer
`q`	Freitextsuche in Nummer und Titel
`sync`	`0`, `1` oder `null`, auch kombiniert (`?sync=0,null`)
`archiviert`	`0` (Vorgabe) oder `1`
`von` / `bis`	Zeitraum `YYYY-MM-DD`
`datumsfeld`	Bezugsfeld für `von`/`bis`: `erstellt` (Vorgabe), `gemeldet`, `erkannt`, `beginn`, `beendet`, `geschlossen`, `geaendert`, `synctime`
`beschreibung`	`1` = Beschreibungstext mitliefern
`sort`	`nummer` (Vorgabe), `id`, `titel`, `prio`, `status`, `kategorie`, `erstellt`, `gemeldet`, `erkannt`, `geschlossen`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

5. Maßnahmen

Methode	Pfad	Zweck
POST	`/api/v1/massnahmen`	Maßnahme anlegen
GET	`/api/v1/massnahmen`	Liste mit Filtern und Paginierung
GET	`/api/v1/massnahmen/<nummer>`	einzelne Maßnahme
PATCH / PUT	`/api/v1/massnahmen/<nummer>`	Ticketnummer und/oder Sync-Status schreiben

Felder beim Anlegen

Feld	Pflicht	Werte
`nummer`	ja	eindeutig, z. B. `MAS-0042`
`name`	ja	Kurztitel (Alias: `titel`)
`beschreibung`	nein	HTML, wird serverseitig bereinigt (Alias: `text`)
`status`	nein	Vorgabe `offen`
`prioritaet`	nein	`niedrig` · `mittel` · `hoch` · `sehr hoch` (Alias: `prio`)
`enddatum`	nein	`YYYY-MM-DD`, Vorgabe heute
`besitzer`	nein	Benutzer-ID, wird gegen die Benutzertabelle geprüft
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`abteilung`	nein	Freitext
`abteilung_id`	nein	ID aus der Abteilungsverwaltung
`erledigt`	nein	Fortschritt 0–100 (%)
`interner_aufwand`	nein	Manntage, Komma oder Punkt erlaubt
`kosten`	nein	EUR, Komma oder Punkt erlaubt
`ticketnummer`	nein	Freitext; als einziges Sachfeld auch per PATCH änderbar
`archiviert`	nein	`0` (Vorgabe) oder 1
`sync` / `synctime`	nein	siehe Abschnitt 10

Zusätzlich liefert jede Leseantwort das abgeleitete Feld `ticketsync` (`true`/`false`). Es zeigt das Ja/Nein-Feld **Mit Ticketsystem synchronisieren** aus der Maßnahmen-Maske und ist genau dann `true`, wenn `sync` nicht `null` ist. Geschrieben wird es nicht selbst, sondern über `sync` (Abschnitt 10).

Status: `offen` · `in Planung` · `genehmigt` · `in Umsetzung` · `umgesetzt` · `archiviert`

Zwei Automatiken arbeiten wie in der Oberfläche: Status `umgesetzt` oder `archiviert` setzt `erledigt` auf 100, und das Review-Datum wird beim Anlegen auf heute plus ein Jahr gelegt.

Der Unterschied zwischen dem Status `archiviert` und dem Feld `archiviert` ist gewollt: Ersteres beschreibt den fachlichen Zustand der Maßnahme, Letzteres blendet den Datensatz aus der normalen Ansicht aus. Beide sind unabhängig voneinander.

Nur lesbare Felder: `id`, `ersteller`, `erstellt`, `aenderer`, `geaendert`, `reviewer`, `review` sowie `besitzer_name`, `weitere_besitzer_namen` und `ersteller_name` im Format „Nachname, Vorname".

Beispiel: Maßnahme anlegen

```
curl -X POST "https://isms.example.com/api/v1/massnahmen" -H "X-API-Key: $KEY" -H
"Content-Type: application/json" -d '{"nummer":"MAS-0042","name":"MFA für
Fernzugriff","status":"in Planung","prioritaet":"hoch","enddatum":"2026-12-
31","besitzer":17,"abteilung":"IT","erledigt":10,"interner_aufwand":"12,5","kosten":"18
000,00","ticketnummer":"CR-765898","sync":0}'
```

Antwort **201**:

```
{
  "id": 128,
  "nummer": "MAS-0042",
  "ticketsync": true,
  "sync": 0,
  "synctime": null,
  "message": "Maßnahme erfolgreich angelegt"
}
```

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/massnahmen?
status=in%20Umsetzung&erledigt_max=80&sort=enddatum&order=asc&besitzer_name=1"
```

Ein Eintrag der Liste:

```
{
  "id": 128,
  "nummer": "MAS-0042",
  "name": "MFA für Fernzugriff",
  "besitzer": 17,
  "besitzer_name": "Müller, Anna",
  "weitere_besitzer": [23],
  "weitere_besitzer_namen": ["Schmidt, Jonas"],
  "abteilung": "IT",
  "abteilung_id": 3,
  "status": "in Umsetzung",
  "prioritaet": "hoch",
  "enddatum": "2026-12-31",
  "erledigt": 80,
  "interner_aufwand": 12.5,
  "kosten": 18000.0,
  "ticketnummer": "CR-765898",
  "archiviert": false,
  "ersteller": 0,
  "erstellt": "2026-08-14",
  "aenderer": 0,
  "geaendert": "2026-08-14",
  "reviewer": 0,
  "review": "2027-08-14",
  "ticketsync": true,
  "sync": 0,
  "synctime": null
```

```
}
```

Einzelne Maßnahme

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/massnahmen/MAS-0042"
```

Anders als bei den Incidents liefert die Einzelabfrage hier den **vollständigen** Datensatz: alle Felder der Liste, dazu `beschreibung` sowie `besitzer_name`, `weitere_besitzer_namen` und `ersteller_name` – ohne dass dafür Parameter nötig wären.

Ticketnummer nachtragen

Die Ticketnummer ist neben den weiteren Besitzern (Abschnitt 3) das einzige Sachfeld, das nicht nur beim Anlegen, sondern auch später über PATCH geschrieben werden kann – gedacht für den Fall, dass das Ticketsystem die Nummer erst beim Übernehmen vergibt:

```
curl -X PATCH "https://isms.example.com/api/v1/massnahmen/MAS-0042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"ticketnummer":"CR-765898","sync":1}'
```

Antwort **200**:

```
{
  "nummer": "MAS-0042",
  "ticketnummer": "CR-765898",
  "ticketsync": true,
  "sync": 1,
  "synctime": "2026-08-21 10:15:00",
  "weitere_besitzer": [23],
  "message": "Maßnahme aktualisiert"
}
```

`{"ticketnummer": null}` leert das Feld. `{"weitere_besitzer": [23, 31]}` ersetzt die weiteren Besitzer. Weil beides fachliche Änderungen sind, führt die API dabei – wie die Oberfläche – Änderungsdatum und Änderer mit; `Aenderer 0` steht für die Schnittstelle. Den Sync-Zustand rührt sie nicht an: wer mit derselben Anfrage quittieren will, gibt `sync` mit (wie im Beispiel oben). Enthält der Body nur `sync`/`synctime`, bleibt die Meldung wie bisher `Sync-Status aktualisiert`.

Listen-Parameter

Parameter	Wirkung
`status`, `prioritaet`	mehrfach oder kommagetrennt; `prio` ist ein erlaubter Aliasname
`abteilung`, `besitzer`	ebenso; `besitzer` ist die Benutzer-ID und trifft Haupt- wie weitere Besitzer
`q`	Freitextsuche in Nummer, Name und Ticketnummer
`sync`	`0`, `1` oder `null`, auch kombiniert
`archiviert`	`0` (Vorgabe) oder `1`
`erledigt_min` / `erledigt_max`	Fortschritt in Prozent
`von` / `bis`	Zeitraum `YYYY-MM-DD`
`datumsfeld`	`enddatum` (Vorgabe), `erstellt`, `geaendert`, `review`, `synctime`
`beschreibung`	`1` = Beschreibungstext mitliefern
`besitzer_name`	`1` = Namen zu Besitzer, weiteren Besitzern und Ersteller mitliefern
`sort`	`nummer` (Vorgabe), `id`, `name`, `status`, `prioritaet`, `enddatum`, `erledigt`, `kosten`, `aufwand`, `abteilung`, `erstellt`, `geaendert`, `review`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

Nützliche Abfragen:

```
# Überfällige Maßnahmen, die noch nicht fertig sind
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/massnahmen?bis=2026-08-
14&erledigt_max=99&status=offen,in%20Planung,genehmigt,in%20Umsetzung"

# Alle Maßnahmen einer Abteilung mit Kosten, absteigend
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/massnahmen?
abteilung=IT&sort=kosten&order=desc"
```

6. Assets

Methode	Pfad	Zweck
POST	`/api/v1/assets`	Asset anlegen
GET	`/api/v1/assets`	Liste mit Filtern und Paginierung
GET	`/api/v1/assets/<nummer>`	einzelnes Asset
GET	`/api/v1/assets/werte`	zulässige Werte der Stammdatenfelder
PATCH / PUT	`/api/v1/assets/<nummer>`	Sync-Status schreiben

Wertelisten kommen aus der Datenbank

Hier unterscheiden sich Assets von Incidents und Maßnahmen: Die zulässigen Werte für `kategorie`, `status` und `klassifizierung` stehen nicht im Quelltext, sondern in Stammdatentabellen, die die Administration pflegt (`asset_kategorie`, `asset_status`, `klassifizierung`). Was gilt, hängt also von der Installation ab. Die Liste zu `kritikalitaet` (Tabelle `kritikalität`) nennt nur die Stufen, nach denen sich filtern lässt – gesetzt wird die Kritikalität nicht, sie wird berechnet (siehe unten).

Ein Client sollte die Listen deshalb nicht fest verdrahten, sondern abfragen:

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets/werte"

{
  "typ": ["primär", "sekundär"],
  "kategorie": [
    "A – Application", "B – Business Process",
    "Daten / Information", "H – Hardware",
    "I – Infrastructure", "O – Organisation", "S – System"
  ],
  "status": ["aktiv", "entsorgt", "in Beschaffung",
             "in Entsorgung", "inaktiv"],
  "klassifizierung": ["allgemein", "streng vertraulich",
                      "vertraulich", "öffentlich"],
  "kritikalitaet": ["hoch", "kritisch", "niedrig", "normal"],
  "cia_stufen": ["niedrig", "normal", "hoch", "kritisch"],
  "hinweis": "null bedeutet: Stammdatentabelle nicht vorhanden, das Feld wird ungeprüft
übernommen"
}
```

Die Werte oben sind die Auslieferungsvorgaben; eigene Einträge erscheinen automatisch mit. Fehlt eine der Tabellen, liefert das Feld `null` und die API übernimmt den Wert ungeprüft.

Für die Schutzziele (`vertraulichkeit`, `integritaet`, `verfuegbarkeit`, `authentizitaet`) und den `wert` gilt die volle Skala **niedrig · normal · hoch · kritisch** (`cia_stufen`).

Die Kritikalität wird berechnet

Seit v3.94 nimmt die API `kritikalitaet` nicht mehr entgegen. Sie bildet den Wert wie die Oberfläche nach dem **Maximumprinzip**: Die Kritikalität ist die höchste Stufe aus `vertraulichkeit`, `integritaet`, `verfuegbarkeit`, `authentizitaet` und `wert`.

Schutzziele und Wert	Kritikalität
alle leer	`niedrig`
höchste Angabe `normal`	`normal`
ein Feld `hoch`, alle übrigen `niedrig`	`hoch`
ein Feld `kritisch`	`kritisch`

Leere Felder zählen nicht mit. Ein einzelnes hohes Schutzziel bestimmt damit die Einstufung – ein niedriger Wert oder niedrige übrige Ziele senken sie nicht. Dieselbe Regel gilt bei der Vererbung des Schutzbedarfs an Tochterassets.

Schickt ein Client `kritikalitaet` trotzdem mit, wird das Feld ohne Fehler übergangen; bestehende Anbindungen laufen also weiter. Die Antwort auf das Anlegen nennt den berechneten Wert. Liste, Filter und Sortierung arbeiten auf demselben Wert, den die Oberfläche anzeigt. Bestände aus älteren Versionen, deren Kritikalität noch von Hand oder per API gesetzt war, gleicht WolfISMS beim ersten Zugriff einmal ab.

Felder beim Anlegen

Feld	Pflicht	Werte
`nummer`	ja	eindeutig, z. B. `AST-00042`
`name`	ja	Kurzbezeichnung
`version`	nein	Freitext, z. B. `2024 R2`, `15.0.4355`
`beschreibung`	nein	Fließtext
`typ`	nein	`primär` · `sekundär` (auch `primaer`/`sekundaer`)
`kategorie`	nein	aus `asset_kategorie`
`status`	nein	aus `asset_status`
`klassifizierung`	nein	aus `klassifizierung`
`vertraulichkeit`	nein	niedrig · normal · hoch · kritisch
`integritaet`	nein	dieselbe Skala
`verfuegbarkeit`	nein	dieselbe Skala
`authentizitaet`	nein	dieselbe Skala
`wert`	nein	dieselbe Skala
`besitzer`	nein	Benutzer-ID, wird geprüft
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`abteilung` / `abteilung_id`	nein	Freitext / ID
`personenbezogen`	nein	0 (Vorgabe) oder 1
`betroffene_personen`	nein	Liste oder kommagetrennt
`art_pbd`	nein	Liste oder kommagetrennt
`datenhaltung`	nein	Freitext, z. B. `On-Prem`, `Cloud EU`
`besondere_pbd`	nein	0 (Vorgabe) oder 1
`review`	nein	`YYYY-MM-DD`, Vorgabe heute + 1 Jahr
`archiviert`	nein	0 (Vorgabe) oder 1
`sync` / `synctime`	nein	siehe Abschnitt 10

Die Spaltennamen der Tabelle tragen teils Umlaute; die API benutzt durchgängig ASCII-Namen: `integritaet` → `Integrität`, `verfuegbarkeit` → `Verfügbarkeit`, `authentizitaet` → `Authentizität`, `kritikalitaet` → `Kritik`, `personenbezogen` → `Person`, `geaendert` → `Änderdatum`.

Nur lesbar: `id`, `kritikalitaet` (berechnet), `ersteller`, `erstellt`, `aenderer`, `geaendert`, `datum`, `reviewer` sowie `besitzer_name`, `weitere_besitzer_namen` und `ersteller_name`.

Beispiel: Asset anlegen

```
curl -X POST "https://isms.example.com/api/v1/assets" -H "X-API-Key: $KEY" -H "Content-Type: application/json" -d '{"nummer":"AST-00042","name":"SAP-Produktivsystem","version":"15.0.4355","typ":"primär","kategorie":"A – Application","status":"aktiv","klassifizierung":"vertraulich","besitzer":17,"abteilung":"IT","vertraulichkeit":"hoch","integritaet":"hoch","verfuegbarkeit":"kritisch","personenbezogen":1,"betroffene_personen":["Mitarbeitende"],"art_pbd":["Stammdaten"],"datenhaltung":"On-Prem","sync":0}'
```

Antwort **201**:

```
{
  "id": 312,
  "nummer": "AST-00042",
  "kritikalitaet": "kritisch",
  "sync": 0,
  "synctime": null,
  "message": "Asset erfolgreich angelegt"
}
```

`kritikalitaet` ist hier `kritisch`, weil `verfuegbarkeit` auf `kritisch` steht – die höchste Stufe entscheidet.

`betroffene_personen` und `art_pbd` nehmen sowohl ein JSON-Array als auch eine kommagetrennte Zeichenkette an und werden in beiden Fällen kommagetrennt gespeichert – dasselbe Format, das die Oberfläche schreibt. Beim Lesen liefert die API sie als Array zurück.

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets?kategorie=A%20%E2%80%93%20Application&status=aktiv&sort=nummer&besitzer_name=1"
```

Ein Eintrag der Liste:

```
{
  "id": 312,
  "nummer": "AST-00042",
  "name": "SAP-Produktivsystem",
  "version": "15.0.4355",
  "besitzer": 17,
  "besitzer_name": "Müller, Anna",
  "weitere_besitzer": [23],
  "weitere_besitzer_namen": ["Schmidt, Jonas"],
  "abteilung": "IT",
  "abteilung_id": 3,
  "typ": "primär",
  "kategorie": "A – Application",
  "status": "aktiv",
  "kritikalitaet": "kritisch",
  "personenbezogen": true,
```

```
  "klassifizierung": "vertraulich",
  "vertraulichkeit": "hoch",
  "integritaet": "hoch",
  "verfuegbarkeit": "kritisch",
  "authentizitaet": "",
  "wert": "",
  "betroffene_personen": ["Mitarbeitende"],
  "art_pbd": ["Stammdaten"],
  "datenhaltung": "On-Prem",
  "besondere_pbd": false,
  "archiviert": false,
  "ersteller": 0,
  "erstellt": "2026-08-14",
  "aenderer": 0,
  "geaendert": "2026-08-14",
  "datum": "2026-08-14",
  "reviewer": 0,
  "review": "2027-08-14",
  "sync": 0,
  "synctime": null
}
```


Wie bei den Maßnahmen liefert die Einzelabfrage `/api/v1/assets/<nummer>` den vollständigen Datensatz einschließlich `beschreibung` und der Namen.

Listen-Parameter

Parameter	Wirkung
`typ`, `kategorie`, `status`	mehrfach oder kommagetrennt
`klassifizierung`, `kritikalitaet`	ebenso; `kritikalitaet` filtert auf den berechneten Wert
`abteilung`, `besitzer`	ebenso; `besitzer` ist die Benutzer-ID und trifft Haupt- wie weitere Besitzer
`cia`	Mindeststufe: liefert Assets, bei denen Vertraulichkeit, Integrität, Verfügbarkeit **oder** Authentizität diese Stufe erreicht
`personenbezogen`	0 oder 1
`q`	Freitextsuche in Nummer, Name, Beschreibung und Version
`sync`	0, 1 oder null, auch kombiniert
`archiviert`	0 (Vorgabe) oder 1
`von` / `bis`	Zeitraum als `YYYY-MM-DD`
`datumsfeld`	`erstellt` (Vorgabe), `geaendert`, `review`, `datum`, `synctime`
`beschreibung`	1 = Beschreibungstext mitliefern
`besitzer_name`	1 = Namen zu Besitzer, weiteren Besitzern und Ersteller mitliefern
`sort`	`nummer` (Vorgabe), `id`, `name`, `version`, `typ`, `kategorie`, `status`, `klassifizierung`, `kritikalitaet`, `abteilung`, `erstellt`, `geaendert`, `review`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

`sort=kritikalitaet` ordnet nach Stufe (niedrig · normal · hoch · kritisch), nicht alphabetisch.

Ein Filter auf einen unbekannten Wert wird nicht als Fehler behandelt – er liefert schlicht keine Treffer. So bleibt die Liste auch dann benutzbar, wenn eine Installation die Stammdaten umbenannt hat.

Nützliche Abfragen:

```
# Hoch und kritisch eingestufte Assets, die kritischsten zuerst
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets?
kritikalitaet=hoch,kritisch&sort=kritikalitaet&order=desc"

# Assets mit hohem Schutzbedarf in mindestens einem Schutzziel
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets?cia=hoch&limit=100"

# Personenbezogene Assets, die in der Cloud liegen
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets?
personenbezogen=1&q=Cloud"
```

```
# Assets, deren Review fällig ist
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/assets?
datumsfeld=review&bis=2026-12-31&sort=review"
```

7. Risiken

Methode	Pfad	Zweck
POST	`/api/v1/risiken`	Risiko anlegen
GET	`/api/v1/risiken`	Liste mit Filtern und Paginierung
GET	`/api/v1/risiken/<nummer>`	einzelnes Risiko
GET	`/api/v1/risiken/werte`	Werte, Skalen und Score-Grenzen
PATCH / PUT	`/api/v1/risiken/<nummer>`	Sync-Status schreiben

Die Bewertung wird berechnet, nicht übergeben

Das ist die Besonderheit dieses Bereichs: `riskscore` und `risiko` (die Stufe) sind abgeleitete Werte. Ein POST, der sie mitschickt, wird mit 400 abgewiesen – sonst entstünde ein Datensatz, dessen Stufe nicht zu seiner Bewertung passt.

Die API rechnet nach derselben Regel wie die Oberfläche:

Fall	Grundlage des Scores
Behandlung `Akzeptieren`	Werte **vor** Maßnahmen
Status `behandelt`	Werte **nach** Maßnahmen
sonst	Werte **vor** Maßnahmen

Der Score ist das Produkt aus Auswirkung und Wahrscheinlichkeit. Die Stufe dazu ergibt sich aus den Grenzen der Tabelle `risiko_score`, die die Administration pflegt – in der Auslieferung:

Score	Stufe
0	nicht bewertet
1–4	niedrig
5–8	mittel
9–12	hoch
ab 13	kritisch

Beide Werte stehen in der Antwort des POST, sodass der Client sie nicht selbst nachrechnen muss.

Bewertungsskalen

`wahrscheinlichkeit_vm`, `auswirkung_vm`, `wahrscheinlichkeit_nm` und `auswirkung_nm` sind ganze Zahlen von 0 bis 5; 0 heißt „nicht bewertet". Statt der Zahl wird auch die deutsche Bezeichnung angenommen:

Zahl	Wahrscheinlichkeit	Auswirkung
0	nicht bewertet	nicht bewertet
1	selten	unbedeutend
2	unwahrscheinlich	gering
3	möglich	mäßig
4	wahrscheinlich	groß
5	sicher	katastrophal

`vm` steht für „vor Maßnahmen", `nm` für „nach Maßnahmen".

Werte abfragen

Kategorie und Status kommen wie bei den Assets aus Stammdatentabellen (`risiko_kategorie`, `risiko_status`). Die Behandlungsarten stehen dagegen fest im Programm. Alles zusammen liefert:

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/risiken/werte"
```

Die Antwort enthält `kategorie`, `status`, `behandlung`, beide Skalen als Zuordnung Zahl → Bezeichnung, die Score-Grenzen und eine Kurzbeschreibung der Berechnung.

Behandlung: `nicht bewertet` · `Akzeptieren` · `Reduzieren` · `Vermeiden` · `Übertragen` · `Teilen`

Felder beim Anlegen

Feld	Pflicht	Werte
`nummer`	ja	eindeutig, z. B. `RSK-0042`
`name`	ja	Kurzbezeichnung
`beschreibung`	nein	HTML, wird bereinigt
`businessimpact`	nein	HTML, wird bereinigt
`massnahme`	nein	HTML-Freitext zur Behandlung
`kategorie`	nein	aus `risiko_kategorie`
`status`	nein	aus `risiko_status`
`behandlung`	nein	Vorgabe `nicht bewertet`
`wahrscheinlichkeit_vm`	nein	0–5 oder Bezeichnung
`auswirkung_vm`	nein	0–5 oder Bezeichnung
`wahrscheinlichkeit_nm`	nein	0–5 oder Bezeichnung
`auswirkung_nm`	nein	0–5 oder Bezeichnung
`besitzer`	nein	Benutzer-ID, wird geprüft
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`abteilung` / `abteilung_id`	nein	Freitext / ID
`umgesetzt`	nein	0 (Vorgabe) oder 1
`review`	nein	`YYYY-MM-DD`, Vorgabe heute + 1 Jahr
`archiviert`	nein	0 (Vorgabe) oder 1
`sync` / `synctime`	nein	siehe Abschnitt 10

Achtung auf zwei ähnlich klingende Felder: `massnahme` ist das Textfeld zur Risikobehandlung, `massnahmen` die Verknüpfungsliste zu Maßnahmen-Datensätzen. Nur `massnahme` lässt sich schreiben.

Nur lesbar: `id`, `risiko`, `riskscore`, `ersteller`, `erstellt`, `aenderer`, `geaendert`, `reviewer` sowie `besitzer_name`, `weitere_besitzer_namen` und `ersteller_name`.

Verknüpfungen sind nur lesbar

Die Zuordnungen zu Maßnahmen, Assets, Normen, Bedrohungen, Schwachstellen, Lieferanten und Dokumenten liegen als kommagetrennte ID-Listen in der Tabelle. Die API liefert sie beim Lesen als Zahlen-Arrays:

```
{
  "massnahmen": [1, 3, 5, 7],
```

```
  "assets": [1, 3, 4, 14],
  "normen": [107, 158, 193]
}
```

Beim Anlegen nimmt sie diese Felder **nicht** entgegen und weist einen POST mit 400 ab. Ein Schreiben ohne Prüfung der Gegenseite würde Verknüpfungen auf gelöschte oder gar nicht vorhandene Datensätze erzeugen. Verknüpft wird in der Oberfläche. Altbestände enthalten in diesen Spalten teils den Text `None`; beim Lesen fällt er weg.

Beispiel: Risiko anlegen

```
curl -X POST "https://isms.example.com/api/v1/risiken" -H "X-API-Key: $KEY" -H
"Content-Type: application/json" -d '{"nummer":"RSK-0042","name":"Ransomware über
Lieferkette","kategorie":"Technische Risiken","status":"in
Bewertung","behandlung":"Reduzieren","besitzer":17,"abteilung":"IT","wahrscheinlichkeit
_vm":4,"auswirkung_vm":5,"wahrscheinlichkeit_nm":2,"auswirkung_nm":3,"beschreibung":"<p
>Verschlüsselung durch kompromittierten Dienstleister.</p>","sync":0}'
```

Antwort **201** – Score und Stufe sind berechnet:

```
{
  "id": 88,
  "nummer": "RSK-0042",
  "riskscore": 20,
  "risiko": "kritisch",
  "sync": 0,
  "synctime": null,
  "message": "Risiko erfolgreich angelegt"
}
```

Der Status ist hier `in Bewertung`, also zählen die Werte vor Maßnahmen: 4 × 5 = 20, und 20 liegt über 13 – damit `kritisch`. Wechselt der Status später in der Oberfläche auf `behandelt`, rechnet die Anwendung mit 2 × 3 = 6 und die Stufe fällt auf `mittel`.

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/risiken?
risiko=hoch,kritisch&sort=riskscore&order=desc&besitzer_name=1"
```

Ein Eintrag der Liste:

```
{
  "id": 88,
  "nummer": "RSK-0042",
  "name": "Ransomware über Lieferkette",
  "besitzer": 17,
  "besitzer_name": "Müller, Anna",
  "weitere_besitzer": [23],
  "weitere_besitzer_namen": ["Schmidt, Jonas"],
```

```
  "abteilung": "IT",
  "abteilung_id": 3,
  "kategorie": "Technische Risiken",
  "status": "in Bewertung",
  "behandlung": "Reduzieren",
  "umgesetzt": false,
  "wahrscheinlichkeit_vm": 4,
  "auswirkung_vm": 5,
  "wahrscheinlichkeit_nm": 2,
  "auswirkung_nm": 3,
  "risiko": "kritisch",
  "riskscore": 20,
  "massnahmen": [1, 3, 5],
  "assets": [14],
  "normen": [],
  "bedrohungen": [],
  "schwachstellen": [],
  "lieferanten": [],
  "dokumente": [],
  "archiviert": false,
  "ersteller": 0,
  "erstellt": "2026-08-14",
  "aenderer": 0,
  "geaendert": "2026-08-14",
  "reviewer": 0,
  "review": "2027-08-14",
  "sync": 0,
  "synctime": null
}
```


Mit `?beschreibung=1` kommen `beschreibung`, `businessimpact` und `massnahme` hinzu – drei potenziell lange HTML-Texte, deshalb standardmäßig weggelassen. Die Einzelabfrage `/api/v1/risiken/<nummer>` liefert sie immer mit.

Listen-Parameter

Parameter	Wirkung
`kategorie`, `status`, `behandlung`	mehrfach oder kommagetrennt
`risiko`	Stufe: `nicht bewertet` · `niedrig` · `mittel` · `hoch` · `kritisch`
`abteilung`, `besitzer`	ebenso; `besitzer` ist die Benutzer-ID und trifft Haupt- wie weitere Besitzer
`score_min` / `score_max`	Bereich des berechneten Scores
`umgesetzt`	0 oder 1
`q`	Freitextsuche in Nummer und Name
`sync`	0, 1 oder null, auch kombiniert
`archiviert`	0 (Vorgabe) oder 1
`von` / `bis`	Zeitraum als `YYYY-MM-DD`
`datumsfeld`	`erstellt` (Vorgabe), `geaendert`, `review`, `synctime`
`beschreibung`	1 = die drei Textfelder mitliefern
`besitzer_name`	1 = Namen zu Besitzer, weiteren Besitzern und Ersteller mitliefern
`sort`	`nummer` (Vorgabe), `id`, `name`, `kategorie`, `status`, `behandlung`, `risiko`, `riskscore`, `abteilung`, `erstellt`, `geaendert`, `review`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

Nützliche Abfragen:

```
# Risiken oberhalb der Akzeptanzschwelle, größte zuerst
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/risiken?
score_min=9&sort=riskscore&order=desc"

# Akzeptierte Risiken – für den Managementbericht
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/risiken?
behandlung=Akzeptieren"

# Risiken, deren Review fällig ist
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/risiken?
datumsfeld=review&bis=2026-12-31&sort=review"
```

8. Lieferanten

Methode	Pfad	Zweck
POST	`/api/v1/lieferanten`	Lieferant anlegen
GET	`/api/v1/lieferanten`	Liste mit Filtern und Paginierung
GET	`/api/v1/lieferanten/<nummer>`	einzelner Lieferant
GET	`/api/v1/lieferanten/werte`	Werte, Gewichte und Score-Grenzen
PATCH / PUT	`/api/v1/lieferanten/<nummer>`	Sync-Status, weitere Besitzer, Beschreibung und Cloud-Exit-Strategie schreiben

Kriterien als Worte statt als Zahlen

Die Lieferantenbewertung beruht auf acht Kriterien, die die Oberfläche als Index in die Tabelle `vendor_kurzbewertung` speichert – 0 = Ja, 1 = Nein, 2 = Teilweise, 3 = N/A. Diese Zahlen sind von außen nicht zu erraten, deshalb nimmt die API die Worte entgegen und liefert sie auch so zurück:

```
{"personendaten": "Ja", "zert": "Nein", "recht": "Teilweise", "sicher": "N/A"}
```

Die Zahl wird weiterhin akzeptiert, falls ein Client sie bereits kennt.

Die acht Kriterien: `personendaten` · `schutz` · `recht` · `zert` · `fragebogen` · `vertrag` · `frueher` · `sicher`

Zwei Punkte, die leicht überraschen:

- **N/A zählt wie Ja.** So rechnet auch die Oberfläche.
- **Ein fehlendes Kriterium bedeutet Ja**, nicht „unbekannt" – in der Neuanlagemaske ist der erste Eintrag vorausgewählt, und die API hält sich daran. Bei den Schadenskriterien (`personendaten`, `schutz`, `recht`) ist `Ja` der ungünstigste Wert. Ein POST mit nur `nummer` und `name` erzeugt deshalb bereits eine Schadenshöhe von 4. Wer das nicht möchte, setzt die Kriterien ausdrücklich.

Tier und Typ

Auch hier weicht die Speicherung von der Anschauung ab, und auch hier rechnet die API um:

- **`tier`** wird als Index in `vendor_tier` gespeichert, wo in der Auslieferung Tier 3 an erster Stelle steht. Die API nimmt und liefert die **Tier-Nummer**: 1 = kritisch, 2 = Standard, 3 = unkritisch. Der vollständige Bezeichner („Tier 1 – Kritisch") wird ebenfalls angenommen.
- **`typ`** verweist als ID auf `vendor_typ`. Die API nimmt den **Namen** entgegen („Cloud Anbieter") und liefert ihn zurück; die ID steht zusätzlich als `typ_id` in der Antwort.

Die Bewertung wird berechnet

Wie bei den Risiken sind `risiko` und `risikoscore` abgeleitet; ein POST mit diesen Feldern wird mit 400 abgewiesen. Die API rechnet wie die Oberfläche:

Größe	Formel
Schadenshöhe	round((personendaten + schutz + tier + recht) / 4), mindestens 1
Eintrittswahrscheinlichkeit	round((zert + fragebogen + vertrag + frueher) / n), mindestens 1
`risikoscore`	Eintrittswahrscheinlichkeit × Schadenshöhe
`risiko`	Stufe laut Tabelle `risiko_score`

Die Punkte je Antwort stehen in `vendor_gewichte` und sind pro Installation einstellbar – in der Auslieferung geben die Schadenskriterien Ja = 5, Teilweise = 3, Nein = 0, die Eintrittskriterien umgekehrt Ja = 0, Teilweise = 2, Nein = 5. Der Tier-Beitrag ist fest: Tier 1 = 5, Tier 2 = 3, Tier 3 = 0 Punkte.

Der POST liefert alle vier Werte zurück, sodass nachvollziehbar ist, wie der Score zustande kam.

Cloud-Exit-Strategie

Seit v3.93 brauchen Lieferanten, deren Typ unter `/admin/vendor_typen` als Cloud/SaaS markiert ist (`vendor_typ.cloud`), eine Exit-Strategie (`vendor.cloud_exit`). Die API liefert dazu drei Felder:

Feld	Bedeutung
`cloud`	`true`, wenn der Typ ein Cloud-/SaaS-Typ ist
`cloud_exit`	die Strategie als HTML, leer als `""`
`cloud_exit_fehlt`	`true` bei Cloud-Lieferanten ohne Strategie

Geschrieben wird `cloud_exit` per POST und nachträglich per PATCH bzw. PUT:

- **HTML** wird wie in der Oberfläche bereinigt, Skript-Inhalte fallen weg.
- **Reiner Text** wird zeilenweise in Absätze umgesetzt, Sonderzeichen werden maskiert.
- `null`, `""` oder ein leerer Absatz löschen die Strategie.
- Ein PATCH setzt das Änderungsdatum und schreibt „API: Cloud-Exit-Strategie geändert“ bzw. „… gelöscht“ ins Aktivitätsprotokoll.
- Falsche Typen – etwa eine Liste statt Text – werden mit **400** abgewiesen.

```
curl -X PATCH "https://isms.example.com/api/v1/lieferanten/LIF-00042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"cloud_exit":"Datenexport monatlich als
CSV.\nLöschbestätigung nach Vertragsende.\nAlternativanbieter: Beispiel AG."}'
```

`GET /api/v1/lieferanten/werte` nennt unter `cloud_typen` die als Cloud/SaaS markierten Typen. Legt die API die beiden Spalten als Erste an, werden Typen mit „Cloud“ oder „SaaS“ im Namen

genauso vorbelegt wie über die Oberfläche.

Beschreibung

Freitext zum Lieferanten (`vendor.beschreibung`), in der Oberfläche ein TinyMCE-Feld. Die API liefert ihn als `beschreibung` (HTML, leer als `""`) und nimmt ihn per POST und nachträglich per PATCH bzw. PUT entgegen – mit denselben Regeln wie `cloud_exit`: HTML wird bereinigt, reiner Text zeilenweise in Absätze umgesetzt, `null` oder `""` löschen die Beschreibung. Ein PATCH setzt das Änderungsdatum und schreibt „API: Beschreibung geändert“ bzw. „… gelöscht“ ins Aktivitätsprotokoll.

```
curl -X PATCH "https://isms.example.com/api/v1/lieferanten/LIF-00042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"beschreibung":"Hosting der
Kundenportale.\nRechenzentren in Frankfurt und Berlin."}'
```

Felder beim Anlegen

Feld	Pflicht	Werte
`nummer`	ja	eindeutig, z. B. `LIF-00042`
`name`	ja	Firmenname
`status`	nein	aus `vendor_status`
`typ`	nein	Name oder ID aus `vendor_typ`
`tier`	nein	1, 2 oder 3
`besitzer`	nein	Benutzer-ID, wird geprüft
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`ansprechpartner`	nein	Freitext
`email`	nein	Freitext
`personendaten`	nein	Ja · Nein · Teilweise · N/A
`schutz`	nein	dieselbe Auswahl
`recht`	nein	dieselbe Auswahl
`zert`	nein	dieselbe Auswahl
`fragebogen`	nein	dieselbe Auswahl
`vertrag`	nein	dieselbe Auswahl
`frueher`	nein	dieselbe Auswahl
`sicher`	nein	dieselbe Auswahl
`beschreibung`	nein	HTML oder Text, siehe Beschreibung
`cloud_exit`	nein	HTML oder Text, siehe Cloud-Exit-Strategie
`review`	nein	`YYYY-MM-DD`, Vorgabe heute + 1 Jahr
`sync` / `synctime`	nein	siehe Abschnitt 10

Nur lesbar: `id`, `risiko`, `risikoscore`, `typ_id`, `tier_bezeichnung`, `massnahmen`, `assets`, `ersteller`, `erstellt`, `aenderer`, `geaendert`, `reviewer`, `cloud`, `cloud_exit_fehlt` sowie `besitzer_name`, `weitere_besitzer_namen` und `ersteller_name`. `massnahmen` und `assets` kommen als Zahlen-Arrays und werden – wie bei den Risiken – nur in der Oberfläche gepflegt.

Die Tabelle kennt keine Archivierung; einen Filter `archiviert` gibt es hier also nicht.

Beispiel: Lieferant anlegen

```
curl -X POST "https://isms.example.com/api/v1/lieferanten" -H "X-API-Key: $KEY" -H
"Content-Type: application/json" -d '{"nummer":"LIF-00042","name":"Beispiel Cloud
GmbH","status":"Aktiv","typ":"Cloud
Anbieter","tier":1,"besitzer":17,"ansprechpartner":"Meier","email":"kontakt@example.com
```

```
","personendaten":"Ja","schutz":"Ja","recht":"Teilweise","zert":"Nein","fragebogen":"Ne
in","vertrag":"Ja","frueher":"Nein","sicher":"N/A","sync":0}'
```

Antwort **201** – die Bewertung ist gerechnet:

```
{
  "id": 42,
  "nummer": "LIF-00042",
  "eintrittswahrscheinlichkeit": 4,
  "schadenshoehe": 4,
  "risikoscore": 16,
  "risiko": "kritisch",
  "sync": 0,
  "synctime": null,
  "message": "Lieferant erfolgreich angelegt"
}
```

Nachgerechnet: Schaden aus personendaten 5 + schutz 5 + Tier 1 mit 5 + recht (Teilweise) 3 = 18, geteilt durch 4 und gerundet = 4. Eintritt aus zert (Nein) 5 + fragebogen (Nein) 5 + vertrag (Ja) 0 + frueher (Nein) 5 = 15, geteilt durch 4 und gerundet = 4. Zusammen 16 – oberhalb von 13 und damit `kritisch`.

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?
tier=1&sort=risikoscore&order=desc&besitzer_name=1"
```

Ein Eintrag der Liste:

```
{
  "id": 42,
  "nummer": "LIF-00042",
  "name": "Beispiel Cloud GmbH",
  "personendaten": "Ja",
  "vertrag": "Ja",
  "recht": "Teilweise",
  "zert": "Nein",
  "sicher": "N/A",
  "status": "Aktiv",
  "typ": "Cloud Anbieter",
  "typ_id": 3,
  "tier": 1,
  "tier_bezeichnung": "Tier 1 – Kritisch",
  "besitzer": 17,
  "besitzer_name": "Müller, Anna",
  "weitere_besitzer": [23],
  "weitere_besitzer_namen": ["Schmidt, Jonas"],
  "fragebogen": "Nein",
  "ansprechpartner": "Meier",
  "email": "kontakt@example.com",
  "schutz": "Ja",
  "risiko": "kritisch",
```

```
  "risikoscore": 16,
  "frueher": "Nein",
  "massnahmen": [],
  "assets": [131, 125, 113],
  "ersteller": 0,
  "erstellt": "2026-08-14",
  "aenderer": 0,
  "geaendert": "2026-08-14",
  "reviewer": 0,
  "review": "2027-08-14",
  "beschreibung": "<p>Hosting der Kundenportale.</p>",
  "cloud": true,
  "cloud_exit": "",
  "cloud_exit_fehlt": true,
  "sync": 0,
  "synctime": null
}
```


Listen-Parameter

Parameter	Wirkung
`status`	mehrfach oder kommagetrennt
`typ`	Name oder ID, mehrfach
`tier`	Tier-Nummer 1–3, mehrfach
`risiko`	`niedrig` · `mittel` · `hoch` · `kritisch`
`besitzer`	Benutzer-ID, mehrfach; trifft Haupt- und weitere Besitzer
`cloud`	1 = nur Cloud-/SaaS-Lieferanten, `0` = nur die übrigen
`cloud_exit`	`fehlt` = Cloud-Lieferanten ohne Strategie, `vorhanden` = mit Strategie
`score_min` / `score_max`	Bereich des berechneten Scores
`q`	Freitextsuche in Nummer, Name, Ansprechpartner und E-Mail
`sync`	0, 1 oder null, auch kombiniert
`von` / `bis`	Zeitraum als `YYYY-MM-DD`
`datumsfeld`	`erstellt` (Vorgabe), `geaendert`, `review`, `synctime`
`besitzer_name`	1 = Namen zu Besitzer, weiteren Besitzern und Ersteller mitliefern
`sort`	`nummer` (Vorgabe), `id`, `name`, `status`, `typ`, `tier`, `risiko`, `risikoscore`, `besitzer`, `erstellt`, `geaendert`, `review`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

Nützliche Abfragen:

```
# Kritische Lieferanten mit hohem Risikoscore
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?
tier=1&score_min=9&sort=risikoscore&order=desc"

# Lieferanten ohne ausgefüllten Sicherheitsfragebogen finden – dazu die Liste
# holen und clientseitig auf fragebogen != "Ja" prüfen
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?
status=Aktiv&limit=200"

# Cloud-Lieferanten ohne dokumentierte Exit-Strategie
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?
cloud_exit=fehlt"
```

Der Filter läuft in der Datenbank, `total` und die Seitenaufteilung stimmen also.

```
# Lieferanten, deren Review fällig ist
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?
datumsfeld=review&bis=2026-12-31&sort=review"
```

Auf die einzelnen Kriterien lässt sich nicht filtern – sie stehen als Index in der Tabelle, und ein Filter darauf wäre genau die Zahlenfalle, die diese API vermeiden soll. Für Auswertungen die Liste holen und im Client prüfen.

9. Schwachstellen

Methode	Pfad	Zweck
POST	`/api/v1/schwachstellen`	Schwachstelle anlegen
GET	`/api/v1/schwachstellen`	Liste mit Filtern und Paginierung
GET	`/api/v1/schwachstellen/<nummer>`	einzelne Schwachstelle
GET	`/api/v1/schwachstellen/werte`	Status, Quellen und Score-Bänder
PATCH / PUT	`/api/v1/schwachstellen/<nummer>`	Status, Score, Besitzer und Sync-Status schreiben

Dieser Bereich ist auf den Scannerbetrieb zugeschnitten: ein Schwachstellenscanner, ein Penetrationstest-Werkzeug oder ein Advisory-Feed liefert Funde ein, die Behandlung läuft in der Oberfläche, und das liefernde System holt sich den Stand später wieder ab.

Die Nummer darf fehlen

Anders als in den übrigen Bereichen ist `nummer` beim POST **optional** – ein Scanner kennt den Nummernkreis von WolfISMS nicht. Fehlt das Feld, vergibt die API die nächste freie Nummer im Format `WEA-NNNNN` und nennt sie in der Antwort. Eine mitgegebene Nummer wird wie überall auf

Eindeutigkeit geprüft; eine doppelte endet mit **409**.

Score und Schweregrad

`score` ist der CVSS-Basiswert von 0.0 bis 10.0 mit einer Nachkommastelle, `null` heißt „nicht bewertet". Der Schweregrad wird daraus abgeleitet und nicht gespeichert – er steht in jeder Antwort als `schwere`:

schwere	Score
`kritisch`	ab 9.0
`hoch`	ab 7.0 bis unter 9.0
`mittel`	ab 4.0 bis unter 7.0
`niedrig`	über 0.0 bis unter 4.0
`info`	genau 0.0
`ohne`	kein Score hinterlegt

Dieselben Grenzen nennt `GET /api/v1/schwachstellen/werte`, zusammen mit den zulässigen Status- und Quellenwerten. Ein Client sollte sie dort abfragen, statt sie fest zu hinterlegen.

Felder beim Anlegen

Feld	Pflicht	Werte
`titel`	ja	Kurzbezeichnung, höchstens 200 Zeichen
`nummer`	nein	eindeutig, z. B. `WEA-00042`; fehlt sie, vergibt die API sie
`beschreibung`	nein	Freitext: betroffene Komponente, CVE, technische Details, Behebung
`score`	nein	0.0–10.0, eine Nachkommastelle; `null` = nicht bewertet
`quelle`	nein	`Schwachstellenscan` · `Penetrationstest` · `Hersteller-Advisory` · `CERT/BSI-Warnung` · `Interne Meldung` · `Audit` · `Bug Bounty` · `Sonstige`
`status`	nein	Neu (Vorgabe) · `in Bewertung` · `Bestätigt` · `in Behandlung` · `Risiko akzeptiert` · `Behoben` · `Verworfen/Fehlalarm`
`besitzer`	nein	Benutzer-ID, wird geprüft
`weitere_besitzer`	nein	Liste von Benutzer-IDs, siehe Abschnitt 3
`sync` / `synctime`	nein	siehe Abschnitt 10

Nur lesbar: `id`, `schwere`, `ersteller`, `erstellt`, `aenderer`, `geaendert` sowie die vier Verknüpfungslisten.

Als erledigt gelten die drei Status `Behoben`, `Risiko akzeptiert` und `Verworfen/Fehlalarm`; alles andere zählt als offen und lässt sich mit `?offen=1` abrufen. Welche Status das sind, steht ebenfalls

unter `/werte`.

Verknüpfungen sind nur lesbar

Die Zuordnungen zu Assets, Incidents, Risiken und Maßnahmen liegen als kommagetrennte ID-Listen in der Tabelle. Die API liefert sie als Zahlen-Arrays – beim Einzelabruf immer, in der Liste nur mit `?links=1`. Beim Schreiben nimmt sie sie **nicht** entgegen: ein POST oder PATCH mit `assets`, `incidents`, `risiken` oder `massnahmen` endet mit 400. Verknüpft wird in der Oberfläche, die dabei die Gegenseite prüft – wie bei den Risiken (Abschnitt 7).

Beispiel: Schwachstelle anlegen

```
curl -X POST "https://isms.example.com/api/v1/schwachstellen" -H "X-API-Key: $KEY" -H
"Content-Type: application/json" -d '{"titel":"Veraltetes OpenSSL auf dem
Webserver","beschreibung":"CVE-2026-1234, betroffen: web01,
web02","score":7.5,"quelle":"Schwachstellenscan","status":"Neu","besitzer":17}'
```

Antwort **201** – die Nummer hat die API vergeben, der Schweregrad ist gerechnet:

```
{
  "id": 42,
  "nummer": "WEA-00042",
  "titel": "Veraltetes OpenSSL auf dem Webserver",
  "score": 7.5,
  "schwere": "hoch",
  "status": "Neu",
  "sync": 1,
  "synctime": "2026-08-22 09:15:00",
  "message": "Schwachstelle erfolgreich angelegt"
}
```

Weil der Body weder `sync` noch `synctime` enthält, trägt die API `sync = 1` und den aktuellen Zeitpunkt ein: der Schreibzugriff über die Schnittstelle ist selbst der Datenaustausch. Wer den Fund erneut übertragen will, setzt `sync` ausdrücklich auf 0.

Beispiel: Liste lesen

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/schwachstellen?
schwere=kritisch,hoch&offen=1&limit=50"

{
  "total": 2,
  "count": 2,
  "limit": 50,
  "offset": 0,
  "items": [
    {
      "id": 42,
      "nummer": "WEA-00042",
```

```
      "titel": "Veraltetes OpenSSL auf dem Webserver",
      "score": 7.5,
      "schwere": "hoch",
      "quelle": "Schwachstellenscan",
      "status": "Neu",
      "besitzer": 17,
      "weitere_besitzer": [],
      "ersteller": 0,
      "erstellt": "2026-08-22 09:15:00",
      "aenderer": null,
      "geaendert": null,
      "sync": 1,
      "synctime": "2026-08-22 09:15:00"
    }
  ]
}
```

`beschreibung` und die Verknüpfungen bleiben aus der Liste heraus, solange sie nicht mit `?beschreibung=1` bzw. `?links=1` angefordert werden – eine Scanner-Liste wird sonst schnell unhandlich.

Einzelne Schwachstelle

```
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/schwachstellen/WEA-00042"
```

Die Antwort enthält wie die Liste `weitere_besitzer`, zusätzlich `beschreibung` und die vier Verknüpfungslisten `assets`, `incidents`, `risiken` und `massnahmen`. Eine unbekannte Nummer ergibt **404**.

Status, Score und Besitzer ändern

Anders als bei Assets, Risiken und Lieferanten darf PATCH hier auch fachliche Felder schreiben – der Behandlungsstand einer Schwachstelle wandert oft aus dem liefernden System zurück:

```
curl -X PATCH "https://isms.example.com/api/v1/schwachstellen/WEA-00042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"status":"Behoben","score":0}'
```

Änderbar sind `status`, `score`, `besitzer`, `weitere_besitzer`, `sync` und `synctime`. Ein neuer Hauptbesitzer fällt dabei aus den weiteren Besitzern heraus, damit niemand doppelt geführt wird. Jedes andere Feld führt zu einem 400, das den unbekannten Namen nennt. Eine fachliche Änderung setzt `geaendert` auf jetzt und `Aenderer` auf 0 – so ist in der Oberfläche erkennbar, dass die Schnittstelle geschrieben hat.

Listen-Parameter

Parameter	Wirkung
`status`	mehrfach oder kommagetrennt
`quelle`	mehrfach oder kommagetrennt
`schwere`	`kritisch` · `hoch` · `mittel` · `niedrig` · `info` · `ohne`, mehrfach
`besitzer`	Benutzer-ID, mehrfach; trifft Haupt- und weitere Besitzer
`score_min` / `score_max`	Bereich des CVSS-Scores (0–10)
`offen`	1 = nur unerledigte Schwachstellen
`q`	Freitextsuche in Nummer und Titel
`sync`	0, 1 oder null, auch kombiniert
`von` / `bis`	Zeitraum als `YYYY-MM-DD`
`datumsfeld`	`erstellt` (Vorgabe), `geaendert`, `synctime`
`beschreibung`	1 = Beschreibungstext mitliefern
`links`	1 = Verknüpfungs-IDs mitliefern
`sort`	`nummer` (Vorgabe), `id`, `titel`, `score`, `status`, `quelle`, `besitzer`, `erstellt`, `geaendert`, `sync`, `synctime`
`order`	`asc` (Vorgabe) oder `desc`
`limit` / `offset`	`limit` 1–200 (Vorgabe 50), `offset` ab 0

Nützliche Abfragen:

```
# Die schwersten offenen Funde zuerst
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/schwachstellen?
offen=1&sort=score&order=desc&limit=20"

# Alles, was ein Scan seit dem 1. August gemeldet hat
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/schwachstellen?
quelle=Schwachstellenscan&von=2026-08-01&datumsfeld=erstellt"

# Noch nicht zurückgemeldete Funde – die Arbeitsmenge des Sync-Clients
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/schwachstellen?
sync=0&limit=100"
```

10. Synchronisation

Alle sechs Bereiche führen zwei Spalten, mit denen ein externes System festhält, was es bereits übernommen hat:

sync	Bedeutung
null	Datensatz ist von der Synchronisation ausgenommen
0	noch nicht übertragen
1	übertragen

synctime hält den Zeitpunkt der letzten Übertragung (YYYY-MM-DD HH:MM:SS) oder ist null.

Bei Maßnahmen und Vorfällen ist dieser Zustand auch in der Oberfläche sichtbar. In der Maßnahmen-Maske steht dafür das Ja/Nein-Feld **Mit Ticketsystem synchronisieren**: nicht angehakt entspricht sync = null, angehakt setzt sync = 0 – die Maske ändert einen bereits übertragenen Datensatz (sync = 1) also nicht zurück. Die **Letzte Synchronisation** wird dort nur angezeigt; geschrieben wird sie ausschließlich über diese Schnittstelle.

Wird sync auf 1 gesetzt, ohne synctime mitzugeben, trägt die API den aktuellen Zeitpunkt selbst ein. Ein Rücksetzen auf 0 lässt synctime als letzten bekannten Zeitpunkt stehen.

Ablauf eines Sync-Clients

```
# 1. Offene Datensätze holen
curl -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/massnahmen?sync=0&limit=50"

# 2. Nach erfolgreicher Übertragung quittieren
curl -X PATCH "https://isms.example.com/api/v1/massnahmen/MAS-0042" -H "X-API-Key:
$KEY" -H "Content-Type: application/json" -d '{"sync": 1}'
```

Antwort **200**:

```
{
  "nummer": "MAS-0042",
  "ticketsync": true,
  "sync": 1,
  "synctime": "2026-08-14 09:30:12",
  "message": "Sync-Status aktualisiert"
}
```

Gültige Bodys im Überblick – jede Zeile ist ein vollständiger Aufruf für sich:

Body	Wirkung
{"sync": 1}	quittieren, Zeitpunkt = jetzt
{"sync": 1, "synctime": "2026-08-14 09:30:00"}	quittieren mit ausdrücklichem Zeitpunkt
{"sync": 0}	zurücksetzen; synctime bleibt als letzter Zeitpunkt stehen
{"sync": null}	von der Synchronisation ausnehmen
{"synctime": null}	nur den Zeitpunkt löschen

Bei Maßnahmen darf zusätzlich `ticketnummer` im Body stehen – etwa `{"ticketnummer": "CR-765898", "sync": 1}`, um die vom Ticketsystem vergebene Nummer beim Quittieren gleich mit zurückzuschreiben (Abschnitt 5).

Mindestens eines der Felder muss im Body stehen, sonst antwortet die API mit 400.

PATCH ändert bei Assets, Risiken und Lieferanten **ausschließlich** diese beiden Felder und `weitere_besitzer` (siehe Abschnitt 3); bei Incidents kommt `status` hinzu (siehe Abschnitt 4), bei Maßnahmen `ticketnummer` und `weitere_besitzer` (siehe Abschnitt 5) und bei Schwachstellen `status`, `score`, `besitzer` und `weitere_besitzer` (siehe Abschnitt 9). Jedes andere Feld im Body führt zu einem 400 mit dem Hinweis, welche Namen unbekannt sind – so fällt ein Tippfehler sofort auf, statt still ignoriert zu werden. PUT verhält sich identisch zu PATCH.

Incidents und Schwachstellen setzen den Sync-Zustand beim Schreiben selbst: Enthält der Body eines POST oder PATCH weder `sync` noch `synctime`, trägt die API `sync = 1` und den aktuellen Zeitpunkt ein – der Schreibzugriff über die Schnittstelle ist der Datenaustausch. Die übrigen vier Bereiche legen weiterhin mit `sync = null` an und ändern den Zustand nur auf ausdrückliche Angabe.

Beispiel: vollständiger Sync-Client in Python

```
#!/usr/bin/env python3
"""Holt offene Maßnahmen aus WolfISMS, übergibt sie an das Zielsystem
und quittiert jeden erfolgreich übertragenen Datensatz."""

import requests

BASIS = "https://isms.example.com/api/v1"
KEY   = "…"                       # aus sicherer Quelle laden, nicht hart kodieren
KOPF  = {"X-API-Key": KEY}


def offene_massnahmen(seite=200):
    """Alle Datensätze mit sync=0 – erst vollständig einlesen, dann liefern.

    Wichtig: Das Einlesen ist abgeschlossen, bevor der erste Datensatz
    quittiert wird. Würde man seitenweise lesen und dazwischen quittieren,
    schrumpfte die Treffermenge von sync=0 unter dem laufenden `offset`
    weg und jede zweite Seite würde übersprungen.
    """
    eintraege, offset = [], 0
    while True:
        antwort = requests.get(
            f"{BASIS}/massnahmen",
            headers=KOPF,
            params={"sync": 0, "limit": seite, "offset": offset,
                    "beschreibung": 1},
            timeout=30,
        )
        antwort.raise_for_status()
```

```
        daten = antwort.json()
        eintraege.extend(daten["items"])
        offset += daten["count"]
        if daten["count"] == 0 or offset >= daten["total"]:
            return eintraege


def quittieren(nummer):
    antwort = requests.patch(
        f"{BASIS}/massnahmen/{nummer}",
        headers=KOPF,
        json={"sync": 1},
        timeout=30,
    )
    antwort.raise_for_status()
    return antwort.json()["synctime"]


for massnahme in offene_massnahmen():
    # ... hier die Übergabe an das Zielsystem ...
    print(f"{massnahme['nummer']}: {massnahme['name']}")
    zeitpunkt = quittieren(massnahme["nummer"])
    print(f"  quittiert um {zeitpunkt}")
```

Erst quittieren, wenn das Zielsystem den Datensatz wirklich angenommen hat. Bricht der Lauf vorher ab, bleibt `sync=0` stehen und der nächste Durchgang holt den Datensatz erneut. Genau deshalb quittiert das Beispiel einzeln pro Datensatz und nicht gesammelt am Ende.

Bei sehr großen Beständen ist die Alternative, immer nur die erste Seite abzurufen (`offset` bleibt 0) und den Aufruf zu wiederholen, bis `total` auf 0 steht – dann muss nichts im Speicher gehalten werden. Das funktioniert nur, solange jeder gelesene Datensatz auch quittiert wird, sonst dreht sich die Schleife endlos über denselben Einträgen.

11. Testen mit curl

Die frühere REST-Konsole unter Administration ist mit v4.00 entfallen. Zum Ausprobieren genügt ein gewöhnlicher HTTP-Client wie `curl`; die Beispiele in den Abschnitten 4 bis 10 lassen sich unverändert absetzen. Den Schlüssel legt man dazu am besten in eine Umgebungsvariable, damit er nicht in der Shell-Historie landet:

```
read -rs KEY        # API-Key eingeben, er wird nicht angezeigt
curl -s -H "X-API-Key: $KEY" "https://isms.example.com/api/v1/lieferanten?limit=5"
```

`-i` zeigt zusätzlich Statuscode und Antwort-Header. Für schreibende Aufrufe empfiehlt sich ein eigener Schlüssel auf einer Testinstanz – ein POST legt echte Datensätze an.

12. Hinweise für den Betrieb

- **Nummernkreise:** Die API vergibt keine Nummern – mit einer Ausnahme. Das aufrufende System muss `nummer` selbst setzen und für Eindeutigkeit sorgen; eine doppelte Nummer quittiert die API mit 409. Die Oberfläche verwendet die Muster `INC-#####`, `MAS-####`, `AST-#####`, `RSK-####`, `LIF-#####` und `WEA-#####`. Ausnahme sind die Schwachstellen: fehlt dort `nummer`, vergibt die API die nächste freie `WEA-#####` (Abschnitt 9) – ein Scanner kennt den Nummernkreis nicht.
- **Schema:** Die benötigten Spalten (`sync`, `synctime`, bei Maßnahmen zusätzlich `Ticketnummer` und `abteilung_id`, bei Assets und Risiken `abteilung_id`) legt die API beim ersten Aufruf selbst an. Ein manuelles Datenbank-Update ist nicht nötig.
- **Stammdaten der Assets:** Kategorie, Status und Klassifizierung kommen aus Tabellen, die die Administration pflegt. Ein Client sollte sie über `/api/v1/assets/werte` abfragen, statt sie fest zu hinterlegen – siehe Abschnitt 6. Die Kritikalität setzt kein Client; die API berechnet sie aus Schutzbedarf und Wert. Für Risiken gilt dasselbe über `/api/v1/risiken/werte`, dort kommen die Score-Grenzen hinzu (Abschnitt 7), und für Lieferanten über `/api/v1/lieferanten/werte` samt der Gewichte aus `vendor_gewichte` (Abschnitt 8). Für Schwachstellen nennt `/api/v1/schwachstellen/werte` die Status- und Quellenlisten sowie die Grenzen der Schweregrade (Abschnitt 9).
- **Protokollierung:** Jeder schreibende Zugriff erscheint im Aktivitäten-Feed der Anwendung („API: angelegt", „API: Sync-Status 1"). Als Benutzer wird 0 geführt, da kein angemeldeter Benutzer beteiligt ist.
- **Beschreibungstexte:** Bei Maßnahmen wird `beschreibung` als HTML gespeichert und in der Oberfläche gerendert. Die API bereinigt den Text serverseitig; Skript-Inhalte werden entfernt.
- **Schlüsselwechsel:** Für jedes anbindende System einen eigenen Schlüssel mit sprechender Beschreibung anlegen. Dann lässt sich eine einzelne Anbindung abschalten, ohne die übrigen zu stören.
- **Abgeschaltete Module:** Schaltet ein Administrator ein Modul unter Administration → Module aus, ist auch sein API-Bereich gesperrt. Die Aufrufe enden dann mit **403** und der Antwort `{"success": false, "error": "… Dieses Modul wurde deaktiviert."}` – der Schlüssel ist in Ordnung, das Modul nicht erreichbar. Erfasste Daten bleiben erhalten und sind nach dem Wiedereinschalten unverändert abrufbar.