Viewconic

Security- und Code-Audit

Anwendung **Kundenportal** · Node.js, Express, TypeScript, SQLite

ERSTELLT FÜR

Muster AG (fiktiv)
Max Beispiel, Leitung IT

IHR ANSPRECHPARTNER

David Kreutzer
d.kreutzer@viewconic.de

Datum	Prüfzeitraum	Version	Einstufung
1. November 2026	**1. November 2026**	**1.0**	**Beispiel, fiktive Daten**

Viewconic GmbH
Ammerseestr. 23
82131 Gauting

d.kreutzer@viewconic.de
www.viewconic.de

AG München, HRB 257528
USt-ID DE331830665
Geschäftsführer: David Kreutzer

Inhalt

1 Management-Summary 2

1.1 Auftrag und Rahmen 2

1.2 Gesamtbewertung 3

1.3 Bewertung nach Bereichen 4

1.4 Die wichtigsten Maßnahmen 4

2 Maßnahmenplan 6

3 Sicherheitsbefunde 8

4 Qualität, Betrieb und Dependencies 25

4.1 Code-Qualität und Wartbarkeit 25

4.2 Komplexität und Umfang 28

4.3 Performance und Caching 31

4.4 Tests 34

4.5 Build, CI/CD und Betrieb 35

4.6 Dependencies 37

5 Automatisierte Prüfung 37

6 Umfang, Methodik und Grenzen 39

A Anhang 41

KAPITEL 1

Management-Summary

1.1 Auftrag und Rahmen

Dieser Report ist ein Beispiel. Die Muster AG und ihr Kundenportal sind erfunden; die Anwendung wurde von Viewconic eigens als Prüfgegenstand geschrieben und enthält absichtlich typische Schwachstellen. Ablauf, Methodik und Layout entsprechen einem echten Code-Audit.

Gegenstand ist das Kundenportal, über das Geschäftskunden Bestellungen und Rechnungen einsehen, Dokumente hochladen und Webhooks für ihre Warenwirtschaft einrichten. Geprüft wurde der vollständige Quellcode samt Git-Historie, Build- und Deployment-Konfiguration auf dem Stand des Hauptzweigs.

Die Prüfung war statisch und rein lesend: automatisierte Werkzeugprüfung auf Secrets, bekannte Schwachstellen, unsichere Muster und Konfigurationsfehler, danach ein KI-gestütztes Review je Prüfbereich. Jeder Befund ab Stufe hoch wurde von einem unabhängigen Prüfer gegengeprüft, bevor er in diesen Report aufgenommen wurde.

Auftraggeber	Muster AG (fiktiv) · Max Beispiel, Leitung IT
Auftragnehmer	Viewconic GmbH · David Kreutzer
Bezug	Beispielreport – fiktive Anwendung
Projekt	Kundenportal · Node.js, Express, TypeScript, SQLite · öffentlich erreichbar
Git-Repository	`muster-kundenportal (Git-Bundle)` · Branch main · Commit `e0e3cde972c5` · 20.10.2026
Umfang	22 Dateien · 309 Zeilen Produktivcode · 30 Zeilen Testcode
Zeitraum	1. November 2026 · Remote, Prüfrechner der Viewconic GmbH
Prüfart	Statisch, nur lesend · Prüftiefe „standard“ · kein Penetrationstest
Prüfgrundlage	Werkzeuge (Secrets, CVE, SAST, Konfiguration) und KI-gestütztes Review entlang OWASP Top 10:2025; Einstufung nach CVSS 3.1, Zuordnung zu CWE
KI-Modell	Claude Opus 5.5 · `claude-opus-5-5` 8 spezialisierte Prüf- und Review-Agenten; jeder relevante Befund von einem unabhängigen Agenten gegengeprüft und validiert

1.2 Gesamtbewertung

GESAMTURTEIL

Ungenügend

Stufe 1 von 4 · 1,50 Punkte

Das Portal ist in diesem Zustand nicht für den Betrieb im Internet geeignet: Jeder kann sich ohne Zugangsdaten Administratorrechte verschaffen und damit auf die Daten aller Kunden zugreifen.

Skala: 4 Sehr gut · 3 Gut · 2 Handlungsbedarf · 1 Ungenügend. Sicherheit zählt 40 %; ist sie ungenügend, ist es auch das Gesamturteil. Skala und Gewichtung im Kapitel Methodik

Das Kundenportal ist schlank gebaut und gut lesbar. Die Sicherheit hängt aber an Prüfungen, die in jeder Route von Hand geschrieben werden – und an mehreren Stellen fehlen. Am schwersten wiegt die Registrierung: Rolle und Kundennummer kommen ungeprüft aus der Anfrage. Wer das weiß, ist in Minuten Administrator.

Dazu kommen Lücken, mit denen angemeldete Kunden fremde Rechnungen, Dateien und Datenbankinhalte lesen, ein Passwort-Zurücksetzen ohne jeden Nachweis und ein im Code hinterlegter Signaturschlüssel. Die meisten Sofortmaßnahmen sind lokale Änderungen. Die kritische und die hohen Lücken lassen sich in etwa drei Entwicklertagen schließen, alle Sofortmaßnahmen zusammen in sieben bis zehn.

Strukturell fehlen ein Test-Gate vor dem Deployment, Tests für die Mandantentrennung und eine Routine für Abhängigkeits-Updates. Ohne diese drei Bausteine entstehen dieselben Fehler bei der nächsten Erweiterung wieder.

1.3 Bewertung nach Bereichen

BEREICH	BEWERTUNG	KURZURTEIL	SEITE
Sicherheit Befunde: 1 K · 5 H · 17 M · 7 N	Ungenügend	Eine kritische Lücke bei der Registrierung und mehrere Wege zu fremden Daten machen die Mandantentrennung wirkungslos.	8 →
Code-Qualität und Wartbarkeit Befunde: 4 M · 1 N	Handlungsbedarf	Der Code ist kurz und gut lesbar, aber Datenzugriff, Fehlerbehandlung und Typisierung sind nicht einheitlich gelöst.	25 →
Komplexität und Umfang Befunde: 1 M · 2 N	Gut	Die Funktionen sind durchweg einfach und frei von Duplikaten; Overengineering gibt es nur in einem klar abgegrenzten Kernmodul.	28 →
Performance und Caching Befunde: 2 M · 2 N	Handlungsbedarf	Der einzige Cache bringt keinen Geschwindigkeitsgewinn, liefert aber veraltete und mandantenfremde Rechnungen.	31 →
Tests Befunde: 1 H	Ungenügend	Tests gibt es nur punktuell; Sicherheits- und Mandantenlogik ist ungetestet.	34 →
Build, CI/CD und Betrieb Befunde: 1 H · 3 M · 3 N	Handlungsbedarf	Das Deployment ist automatisiert, läuft aber ohne Prüfungen direkt in Produktion und auf einer nicht mehr gepflegten Laufzeitumgebung.	35 →
Dependencies Befunde: 1 H · 2 M	Ungenügend	Sechs von neun Laufzeitbibliotheken liegen eine oder mehrere Hauptversionen zurück und sind mit bekannten Schwachstellen belastet; eine Update-Routine fehlt.	37 →

1.4 Die wichtigsten Maßnahmen

SEC-01	`role` bei der Registrierung ignorieren, Kundenzuordnung nur per Einladung	KRITISCH	8
SEC-02	Passwort-Reset nur per befristetem, einmaligem Mail-Link	HOCH	9
SEC-03	Fehler im Reset-Handler abfangen, `email` validieren	HOCH	9
SEC-04	Express aktualisieren (5.x) und Multer ≥ 2.3.0 einspielen	HOCH	10
SEC-05	Fallback-Secret entfernen, JWT-Schlüssel neu erzeugen	HOCH	10

Alle 34 Sofortmaßnahmen und die strukturellen Verbesserungen: Maßnahmenplan, Seite 6

WAS GUT GELÖST IST

- Die Funktionen sind klein und übersichtlich, ohne Code-Duplikate und ohne komplexe Verschachtelung.
- Alle nicht öffentlichen Bereiche sind auf Router-Ebene an eine Anmeldung gebunden, der Admin-Bereich zusätzlich an eine Rollenprüfung; außer der Bestellsuche nutzen alle Abfragen Platzhalter.
- Abhängigkeiten sind exakt versioniert und mit Lockfile festgeschrieben, `.env` ist von der Versionierung ausgeschlossen – eine gute Grundlage für reproduzierbare Builds; die Pipeline nutzt das Lockfile derzeit aber nicht.

KAPITEL 2

Maßnahmenplan

Sofortmaßnahmen schließen konkrete Lücken und sollten vor dem nächsten Release umgesetzt werden. Strukturelle Verbesserungen senken das Risiko dauerhaft und gehören in die Planung.

SOFORTMASSNAHMEN

ID	MASSNAHME	EINSTUFUNG	SEITE
SEC-01	`role` bei der Registrierung ignorieren, Kundenzuordnung nur per Einladung	KRITISCH	8
SEC-02	Passwort-Reset nur per befristetem, einmaligem Mail-Link	HOCH	9
SEC-03	Fehler im Reset-Handler abfangen, `email` validieren	HOCH	9
SEC-04	Express aktualisieren (5.x) und Multer ≥ 2.3.0 einspielen	HOCH	10
SEC-05	Fallback-Secret entfernen, JWT-Schlüssel neu erzeugen	HOCH	10
SEC-06	Rate-Limit und Fehlversuchsbremse am Login, Passwortregel verschärfen	HOCH	11
QA-01	Prüf-Job (Typecheck, Lint, Tests) vor den Deploy schalten	HOCH	35
SEC-07	Lockfile mitliefern und `npm ci` statt `npm install` auf dem Server	MITTEL	11
SEC-08	Download nur per Dokument-ID mit Mandantenfilter, Pfad auf Upload-Verzeichnis begrenzen	MITTEL	12
SEC-09	Parametrisierte Abfrage in `GET /api/orders/search`	MITTEL	13
SEC-10	Mandantenfilter in `GET /api/invoices/:id`, Rechnungs-Cache entfernen	MITTEL	13
SEC-11	Rate-Limit und Adressbestätigung für Registrierung und Reset	MITTEL	14
SEC-12	Webhook-URLs gegen interne Ziele filtern, Antwortinhalt nicht zurückgeben	MITTEL	14
SEC-13	Zertifikatsprüfung und `requireTLS` im SMTP-Transport aktivieren	MITTEL	15
SEC-14	Body-Limit auf 100 kB senken, Upload-Limits in `multer` setzen	MITTEL	16
SEC-15	Uploads mit zufälligem Namen je Kunde ablegen, Größe und Typ begrenzen	MITTEL	16
SEC-16	`errorHandler` auf generische Meldung mit Fehler-ID umstellen	MITTEL	17
SEC-18	Einheitliche Antworten bei Login, Reset und Registrierung	MITTEL	18
SEC-19	nodemailer, axios und lodash aktualisieren, Renovate einschalten	MITTEL	18

ID	MASSNAHME	EINSTUFUNG	SEITE
SEC-20	Request-Body-Log am Login entfernen, alte Logs löschen	MITTEL	19
SEC-21	Secrets aus der Historie widerrufen und rotieren, gitleaks in CI	MITTEL	19
SEC-23	jsonwebtoken 9.x, Algorithmus in `verifyToken` festlegen	MITTEL	20
QA-04	Positionen gebündelt laden, Index auf `order_items.order_id`	MITTEL	31
QA-08	Verschluckte Fehler loggen, Mailversand entkoppeln, async-Handler einheitlich absichern	MITTEL	26
QA-10	`invoiceCache` entfernen, Rechnung mit Mandantenfilter direkt lesen	MITTEL	32
SEC-24	Dockerfile: `USER node`, `.dockerignore`, Multi-Stage mit `npm ci`	NIEDRIG	21
SEC-25	Host-Schlüssel in `known_hosts` hinterlegen, `StrictHostKeyChecking=no` entfernen	NIEDRIG	22
SEC-26	Actions auf SHA pinnen, `permissions: contents: read`, Deploy-Environment	NIEDRIG	22
SEC-27	Token nur noch aus dem Authorization-Header lesen	NIEDRIG	23
SEC-28	CORS-Fallback `true` durch feste Origin-Allowlist ersetzen	NIEDRIG	23
SEC-29	`helmet()` einbinden, Version aus `/health` entfernen	NIEDRIG	24
SEC-30	Passwort-Hash aus der Admin-Nutzerliste entfernen	NIEDRIG	24
QA-14	Wirkungslosen Settings-Endpunkt und Lodash entfernen oder echte Speicherung bauen	NIEDRIG	31
QA-15	Indizes auf `customer_id` (Bestellungen zusätzlich `created_at`) anlegen	NIEDRIG	33

STRUKTURELLE VERBESSERUNGEN

ID	MASSNAHME	EINSTUFUNG	SEITE
QA-02	Testbasis mit Testmandanten aufbauen, Mandantentrennung und Auth je Route testen	HOCH	34
SEC-17	Passwort-Hashing auf Argon2id umstellen, Altbestand migrieren	MITTEL	17
SEC-22	Fester Token-Ablauf, Widerruf über Token-Version je Nutzer	MITTEL	20
QA-03	Dependabot/Renovate einrichten, Major-Updates schrittweise nachziehen	MITTEL	36
QA-05	Node-LTS, `npm ci` mit Lockfile, einen einheitlichen Build-Artefakt-Weg	MITTEL	36
QA-06	Datenzugriff je Fachbereich bündeln, Mandant als Pflichtparameter, Migrationen einführen	MITTEL	25
QA-07	Schemaprüfung für alle Routen einführen, JSON-Limit senken	MITTEL	26

ID	MASSNAHME	EINSTUFUNG	SEITE
QA-09	Repository-Basisklasse und Fabrik durch zwei Datenfunktionen ersetzen	MITTEL	29
QA-11	Audit-Log für Anmeldung, Reset und Admin-Aktionen einführen	MITTEL	27
QA-12	Zeilentypen je Tabelle, Eingaben per Schema prüfen, `any` per Lint verbieten	NIEDRIG	27
QA-13	Event-Bus durch direkten Funktionsaufruf mit Fehlerlogging ersetzen	NIEDRIG	30
QA-16	Paging für Listen, Export streamen	NIEDRIG	33

KAPITEL 3

Sicherheitsbefunde

Jeder Befund ist am Code verifiziert und nach CVSS 3.1 eingestuft. Sortiert nach Schwere.

SEC-01

Jeder kann sich selbst als Administrator oder in einem fremden Kundenkonto registrieren

KRITISCH

CVSS 9,1 `AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N` · OWASP A01:2025 Broken Access Control · CWE-269 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/auth.ts:11`
`src/routes/auth.ts:20`
`src/middleware/auth.ts:28`

BEFUND

`POST /api/auth/register` ist ohne Anmeldung erreichbar und übernimmt `role` und `customerId` unverändert aus der Anfrage (`auth.ts:11` , `:20`). Es gibt keine Einladung, keine Prüfung der Kundennummer und keine Bestätigung der E-Mail-Adresse.

RISIKO

Ein anonymer Angreifer registriert sich mit `role: "admin"` und erhält über `/api/admin/*` alle Nutzer samt Passwort-Hashes, alle Kunden und den Export. Alternativ ordnet er sich einer beliebigen Kundennummer zu und liest deren Bestellungen, Rechnungen und Dokumente.

EMPFEHLUNG

Die Rolle serverseitig fest auf `user` setzen und `role` aus der Anfrage ignorieren. Die Zuordnung zu einem Kunden nur über eine einmalige, befristete Einladung erlauben, die ein Administrator (Rolle `admin`) ausstellt, und das Konto erst nach Bestätigung der E-Mail-Adresse aktivieren.

AUSZUG

```
const { email, password, customerId, role } = req.body;
...
  .run(customerId, email, hashPassword(password), role || 'user');
```

SEC-02

Passwort jedes Kontos lässt sich ohne Nachweis zurücksetzen und anschließend erraten

HOCH

CVSS 8,1 `AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H` · OWASP A07:2025 Authentication Failures · CWE-640 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/auth.ts:47`
`src/routes/auth.ts:53`
`src/routes/auth.ts:54`
`src/routes/auth.ts:55`

BEFUND

`POST /api/auth/password-reset` überschreibt das Passwort jedes Kontos sofort mit einem sechsstelligen Zahlencode, ohne Nachweis, dass die Anfrage vom Kontoinhaber stammt (`auth.ts:54`). Der Code entsteht aus `Math.random` (`:53`) und wird entgegen der Antwort (`:55`) nirgends verschickt.

RISIKO

Ein anonymer Angreifer sperrt jeden Nutzer aus, auch Administratoren. Da der Login keine Versuchsbegrenzung hat, kann er danach die eine Million möglichen Codes durchprobieren und das Konto übernehmen; ob nginx das bremst, ist im Repository nicht erkennbar.

EMPFEHLUNG

Das Passwort erst ändern, wenn der Nutzer einen per Mail zugestellten Link bestätigt hat. Dafür ein Token aus `crypto.randomBytes(32)` erzeugen, nur gehasht speichern, auf etwa 30 Minuten befristen, einmalig verwenden und Anfragen je Adresse und IP begrenzen.

AUSZUG

```
const code = String(Math.floor(Math.random() * 1000000)).padStart(6, '0');
db.prepare('UPDATE users SET password_hash = ? WHERE id = ?').run(hashPassword(code), user.id);
```

SEC-03

Ein einziger anonymer Aufruf des Passwort-Resets bringt das Portal zum Absturz

HOCH

CVSS 7,5 `AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H` · OWASP A10:2025 Mishandling of Exceptional Conditions · CWE-248 · Fehlerbehandlung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/auth.ts:47`
`src/routes/auth.ts:49`

BEFUND

`POST /api/auth/password-reset` ist ein `async`-Handler ohne `try/catch`. Ein Body wie `{"email": true}` lässt better-sqlite3 beim Binden des Parameters eine Ausnahme werfen; unter Express 4 wird daraus eine unbehandelte Promise-Ablehnung, die Node.js 16 mit dem Prozess beendet.

RISIKO

Ein anonymer Angreifer legt das Portal mit einer einzigen Anfrage lahm und kann das nach jedem Neustart wiederholen.

EMPFEHLUNG

Den Handler in `try/catch` fassen und Fehler an `next` weitergeben (oder Express 5 nutzen, das abgelehnte Promises selbst behandelt); `email` vorher als String validieren.

SEC-03 · Fortsetzung · Ein einziger anonymer Aufruf des Passwort-Resets bringt das Portal zum Absturz

AUSZUG

```
authRouter.post('/password-reset', async (req, res) => {
  const { email } = req.body;
  const user = db.prepare('SELECT id FROM users WHERE email = ?').get(email) as any;
```

SEC-04

Portal per präparierter Anfrage ohne Anmeldung lahmzulegen (veraltetes Express-Paket)

HOCH

CVSS 7,5 `AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H` · OWASP A03:2025 Software Supply Chain Failures · CWE-1395 · Dependencies · Quelle: osv-scanner · ↑ Bewertung nach Bereichen

FUNDSTELLE

`package.json:17`
`src/server.ts:15`
`package-lock.json (qs 6.7.0)`

BEFUND

Express 4.17.1 bringt body-parser 1.19.0 und qs 6.7.0 mit. Ohne Anmeldung erreichbar sind zwei Wege: Ein einzelner JSON-Request mit gzip-Body und angehängten Bytes bringt body-parser zum Absturz (CVE-2026-111181, `express.json()` läuft für jede Anfrage), und präparierte Query-Strings blockieren den Prozess über qs (CVE-2022-24999).

RISIKO

Ein anonymer Angreifer bringt den Node-Prozess mit einer einzigen präparierten Anfrage zum Absturz oder blockiert ihn per Query-String; das Portal ist dann für alle Kunden nicht erreichbar.

EMPFEHLUNG

Express auf die aktuelle Version 5.x heben (mindestens eine 4.x, die qs ≥ 6.16.1, body-parser ≥ 1.20.9 und path-to-regexp ≥ 0.1.13 auflöst) und Multer auf ≥ 2.3.0; danach `npm audit` ohne Funde auf Laufzeitpaketen als Gate in die Pipeline aufnehmen.

SEC-05

Anmeldetokens sind fälschbar, wenn der im Code hinterlegte Schlüssel greift

HOCH

CVSS 7,4 `AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N` · OWASP A04:2025 Cryptographic Failures · CWE-321 · Kryptografie · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/config.ts:7`
`src/auth/jwt.ts:16`
`src/auth/jwt.ts:20`

BEFUND

Ist `JWT_SECRET` nicht gesetzt, signiert und prüft die Anwendung Tokens mit einem fest im Code stehenden Schlüssel (`config.ts:7`); ein Start ohne Secret wird nicht verhindert. Derselbe Wert stand im ersten Commit auch in der `.env`, ist also allen mit Zugriff auf Code oder Historie bekannt.

RISIKO

Läuft die Produktion mit diesem Wert – das liegt außerhalb des Repositorys –, erzeugt ein anonymer Angreifer ein Token mit `role: "admin"` und beliebiger `customerId` und hat vollen Lese- und Schreibzugriff auf alle Mandanten.

SEC-05 · Fortsetzung · Anmeldetokens sind fälschbar, wenn der im Code hinterlegte Schlüssel greift

EMPFEHLUNG Den Fallback entfernen und den Start abbrechen, wenn `JWT_SECRET` fehlt oder kürzer als 32 Byte ist. Produktiv einen neuen Zufallsschlüssel setzen (z. B. `openssl rand -base64 48`); dadurch werden alle bisherigen Tokens ungültig.

AUSZUG

```
jwtSecret: process.env.JWT_SECRET || '<im Code hinterlegter Wert>',
```

SEC-06

Passwörter lassen sich am Login unbegrenzt durchprobieren

HOCH

CVSS 7,4 `AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N` · OWASP A07:2025 Authentication Failures · CWE-307 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/auth.ts:28`
`src/auth/password.ts:13`
`src/server.ts:19`

BEFUND `POST /api/auth/login` hat weder Rate-Limit noch Sperre oder Verzögerung nach Fehlversuchen (`auth.ts:28`); in der gesamten Anwendung ist keine Begrenzung eingebunden (`server.ts:19`). Als Passwortregel gilt nur eine Mindestlänge von sechs Zeichen (`password.ts:13`).

RISIKO Ein anonymer Angreifer probiert gängige oder aus anderen Lecks bekannte Passwörter gegen Kundenadressen durch und übernimmt Konten mit schwachem Passwort. Ob nginx Anfragen begrenzt, liegt außerhalb des Repositorys; deshalb als schwerer ausnutzbar eingestuft.

EMPFEHLUNG Fehlversuche je Konto und je IP begrenzen, z. B. mit `express-rate-limit` und schrittweiser Verzögerung. Mindestlänge auf 12 Zeichen anheben, Passwörter gegen eine Liste geleakter Passwörter prüfen und für Admin-Konten einen zweiten Faktor vorsehen.

SEC-07

Produktivserver installiert Abhängigkeiten ohne Lockfile bei jedem Deployment

MITTEL

CVSS 6,8 `AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:N` · OWASP A03:2025 Software Supply Chain Failures · CWE-829 · CI/CD und Lieferkette · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`.github/workflows/deploy.yml:21`
`.github/workflows/deploy.yml:22`

BEFUND Die Pipeline überträgt nur `dist` und `package.json` auf den Server, nicht die `package-lock.json`, und führt dort anschließend `npm install --production` aus. Die direkten Pakete sind zwar exakt angegeben, alle transitiven Abhängigkeiten werden auf dem Produktivserver aber bei jedem Deployment neu aufgelöst und ihre Install-Skripte ausgeführt – mit Versionen, die weder gebaut noch getestet wurden.

SEC-07 · Fortsetzung · Produktivserver installiert Abhängigkeiten ohne Lockfile bei jedem Deployment

RISIKO

Wird ein transitives npm-Paket kompromittiert (wie bei mehreren Vorfällen 2025), läuft dessen Schadcode beim nächsten Push auf `main` direkt auf dem Produktivserver mit den Rechten des Deploy-Benutzers und erreicht Datenbank, Uploads und Konfiguration. Voraussetzung ist ein kompromittiertes Paket im Abhängigkeitsbaum zum Deploy-Zeitpunkt, daher hohe Angriffskomplexität.

EMPFEHLUNG

Auf dem Server nicht mehr installieren: im CI-Job mit `npm ci --omit=dev` ein vollständiges Artefakt (inkl. `node_modules`) bauen und dieses übertragen, oder mindestens `package-lock.json` mitkopieren und auf dem Server `npm ci --omit=dev --ignore-scripts` verwenden (native Module wie better-sqlite3 dann gezielt bauen). Auch im CI `npm ci` statt `npm install` nutzen.

AUSZUG

```
rsync -e "ssh -i key ..." -r dist package.json deploy@portal.muster-ag.example:/srv/portal
ssh -i key ... "cd /srv/portal && npm install --production && pm2 restart portal"
```

SEC-08

Beliebige Dateien des Servers und Dokumente anderer Kunden herunterladbar

MITTEL

CVSS 6,5 `AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` · OWASP A01:2025 Broken Access Control · CWE-22 · Eingaben und Dateien · Quelle: semgrep · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/documents.ts:36`
`src/routes/documents.ts:37`
`src/routes/documents.ts:41`

BEFUND

`GET /api/documents/download` setzt den Query-Parameter `file` per `path.join` an das Upload-Verzeichnis und liefert die Datei aus. Es wird weder geprüft, dass der Pfad im Upload-Verzeichnis bleibt (`../` wird aufgelöst), noch, dass die Datei zum Kunden des Nutzers gehört; die Tabelle `documents` wird gar nicht befragt.

RISIKO

Ein angemeldeter Nutzer lädt mit `?file=../data/portal.db` die komplette Datenbank aller Kunden (inklusive Passwort-Hashes), mit `?file=../.env` die Server-Geheimnisse oder jede andere Datei, die der Prozess lesen darf. Ohne Traversal genügt ein bekannter oder erratener Dateiname, um Dokumente anderer Kunden zu laden.

EMPFEHLUNG

Downloads über die Dokument-ID statt über den Dateinamen anbieten: Datensatz mit `WHERE id = ? AND customer_id = ?` laden und nur den gespeicherten Namen ausliefern. Zusätzlich den aufgelösten Pfad gegen das Upload-Verzeichnis prüfen (`path.resolve` und `startsWith(uploadDir + path.sep)`) oder `res.download(name, { root: uploadDir })` verwenden.

AUSZUG

```
const file = String(req.query.file || '');
const fullPath = path.join(config.uploadDir, file);
...
res.download(fullPath);
```

SEC-09

Bestellsuche erlaubt das Auslesen der gesamten Datenbank

MITTEL

CVSS 6,5 `AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` · OWASP A05:2025 Injection · CWE-89 · Injection · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/orders.ts:22`
`src/routes/orders.ts:23`
`src/routes/orders.ts:24`

BEFUND

`GET /api/orders/search` baut die SQL-Abfrage per String-Verkettung aus den Query-Parametern `q` und `status`. Der Mandantenfilter steht zwar in der Abfrage, lässt sich über ein eingeschleustes `UNION SELECT` aber umgehen. Datenbankfehler gibt die zentrale Fehlerbehandlung im Klartext an den Aufrufer zurück, was das Ausprobieren erleichtert.

RISIKO

Ein angemeldeter Nutzer liest über `status` oder `q` beliebige Tabellen aus, etwa alle Nutzer mit E-Mail, Rolle und Passwort-Hash sowie Bestellungen und Rechnungen aller Kunden. Schreibende Abfragen sind hier nicht möglich, weil die Bibliothek nur eine einzelne lesende Anweisung ausführt; mit den ausgelesenen Hashes lassen sich aber Konten übernehmen.

EMPFEHLUNG

Platzhalter verwenden: `... WHERE customer_id = ? AND status = ? AND reference LIKE ?` mit `.all(customerId, status, '%' + q + '%')`. `status` zusätzlich gegen die Liste erlaubter Werte prüfen.

AUSZUG

```
const sql =
  `SELECT * FROM orders WHERE customer_id = ${req.user!.customerId} ` +
  `AND status = '${status}' AND reference LIKE '%${q}%'`;
res.json(db.prepare(sql).all());
```

SEC-10

Rechnungen anderer Kunden per ID abrufbar

MITTEL

CVSS 6,5 `AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N` · OWASP A01:2025 Broken Access Control · CWE-639 · Autorisierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/invoices.ts:20`
`src/routes/invoices.ts:24`
`src/routes/invoices.ts:28`
`src/services/cache.ts:5`

BEFUND

`GET /api/invoices/:id` lädt die Rechnung nur über die ID, ohne Filter auf `customer_id` des angemeldeten Nutzers. Zusätzlich liegt die Rechnung zehn Minuten im `invoiceCache`, dessen Schlüssel nur die Rechnungs-ID ist; ein Cache-Treffer wird ohne jede Prüfung ausgeliefert. Die übrigen Rechnungsrouten (`/`, `/:id/pdf`, `/:id/paid`) filtern dagegen korrekt nach Kunde.

RISIKO

Ein angemeldeter Nutzer eines beliebigen Kunden zählt die fortlaufenden Rechnungs-IDs hoch und liest Rechnungsnummer, Betrag, Zahlstatus und Bestellbezug aller anderen Kunden.

SEC-10 · Fortsetzung · Rechnungen anderer Kunden per ID abrufbar

EMPFEHLUNG Die Abfrage auf `WHERE id = ? AND customer_id = ?` umstellen, wie in `/:id/pdf`. Den `invoiceCache` entfernen (siehe Befund zum Rechnungs-Cache); falls er bleiben muss, Schlüssel `${customerId}:${id}` und Invalidierung bei `POST /:id/paid`; besser die Besitzprüfung zentral in eine Funktion `getInvoiceForCustomer` legen.

AUSZUG

```
const cached = invoiceCache.get(id);
if (cached) { return res.json(cached); }
const invoice = db.prepare('SELECT * FROM invoices WHERE id = ?').get(id);
```

SEC-11

Registrierung verschickt unbegrenzt Mails an beliebige Adressen

MITTEL

CVSS 6,5 `AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:L` · OWASP A06:2025 Insecure Design · CWE-799 · Missbrauchsschutz · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE
`src/routes/auth.ts:9`
`src/routes/auth.ts:21`

BEFUND Jede anonyme Registrierung löst sofort eine Willkommensmail an die angegebene Adresse aus (`auth.ts:21`), ohne Rate-Limit, Captcha oder Bestätigung der Adresse.

RISIKO Ein Angreifer lässt über das Portal massenhaft Mails an fremde Adressen senden und füllt die Nutzertabelle. Das gefährdet den Ruf der Absenderdomain (Sperrlisten), verbraucht das Mailkontingent und kann den regulären Versand blockieren.

EMPFEHLUNG Registrierungen je IP und Zeitfenster begrenzen und Konten erst nach Bestätigung der Adresse aktivieren; mit der Einladungslösung aus dem Befund zur Selbstregistrierung entfällt der offene Endpunkt ganz. Den Passwort-Reset ebenso begrenzen.

SEC-12

Webhook-Test ruft beliebige Adressen auf und zeigt die Antwort

MITTEL

CVSS 6,3 `AV:N/AC:H/PR:L/UI:N/S:C/C:H/I:N/A:N` · OWASP A01:2025 Broken Access Control · CWE-918 · Externe Aufrufe · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE
`src/routes/webhooks.ts:26`
`src/routes/webhooks.ts:28`
`src/routes/webhooks.ts:29`

BEFUND `POST /api/webhooks/test` ruft die URL aus dem Request-Body ohne Prüfung per `axios.get` vom Server aus auf und gibt Statuscode und vollständigen Antwortinhalt zurück. Interne Adressen (`localhost`, private Netze, Link-Local) werden nicht ausgeschlossen, Weiterleitungen werden gefolgt. Auch `POST /api/webhooks` speichert URLs ungeprüft.

SEC-12 · Fortsetzung · Webhook-Test ruft beliebige Adressen auf und zeigt die Antwort

RISIKO

Ein angemeldeter Nutzer nutzt den Server als Proxy in das interne Netz und liest Dienste, die nur von dort erreichbar sind (z. B. Verwaltungsoberflächen auf dem Server oder, bei Cloud-Betrieb, den Metadatendienst mit Zugangsdaten). Wie schwer das wiegt, hängt vom Netz hinter dem Server ab, das nicht Teil des Repositorys ist; daher als schwerer ausnutzbar eingestuft.

EMPFEHLUNG

URLs beim Anlegen und vor jedem Aufruf validieren: nur `https`, Hostnamen auflösen und private, Loopback- und Link-Local-Adressen ablehnen (z. B. mit einer Bibliothek wie `ssrf-req-filter` als Agent), Weiterleitungen abschalten (`maxRedirects: 0`). Im Test nur Statuscode und Antwortzeit zurückgeben, nicht den Inhalt.

AUSZUG

```
const { url } = req.body;
const response = await axios.get(url, { timeout: 5000 });
res.json({ status: response.status, body: response.data });
```

SEC-13

Mailversand akzeptiert unverschlüsselte und gefälschte SMTP-Verbindungen

MITTEL

CVSS 5,9 `AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N` · OWASP A04:2025 Cryptographic Failures · CWE-295 · Transportverschlüsselung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/services/mail.ts:7`
`src/services/mail.ts:9`

BEFUND

Der SMTP-Transport deaktiviert mit `tls: { rejectUnauthorized: false }` die Zertifikatsprüfung. Zusätzlich ist bei `secure: false` auf Port 587 STARTTLS nicht erzwungen (`requireTLS` fehlt), sodass die Verbindung auch unverschlüsselt aufgebaut wird, wenn der Server bzw. ein Angreifer STARTTLS nicht anbietet.

RISIKO

Wer sich in den Netzweg zwischen Portal-Server und SMTP-Server bringen kann (z. B. über DNS-Manipulation oder ein kompromittiertes Netzsegment), liest die SMTP-Zugangsdaten mit und kann anschließend im Namen der Muster AG Mails versenden; Mailinhalte und Empfängeradressen werden ebenfalls offengelegt.

EMPFEHLUNG

`rejectUnauthorized: false` entfernen und `requireTLS: true` setzen (oder `secure: true` auf Port 465). Bei einem internen Mailserver mit eigenem Zertifikat dessen CA über `tls.ca` hinterlegen statt die Prüfung abzuschalten.

AUSZUG

```
secure: false,
auth: { user: config.smtp.user, pass: config.smtp.pass },
tls: { rejectUnauthorized: false },
```

SEC-14

Sehr große Anfragen und Uploads können den Dienst lahmlegen

MITTEL

CVSS 5,9 `AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H` · OWASP A02:2025 Security Misconfiguration · CWE-770 · Ressourcenbegrenzung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/server.ts:15`
`src/routes/auth.ts:29`
`src/routes/documents.ts:13`

BEFUND

Der JSON-Parser akzeptiert global bis zu 50 MB je Anfrage, auch an anonymen Endpunkten wie `POST /api/auth/login`, der den gesamten Body zusätzlich ins Log schreibt. Der Datei-Upload (`multer`) hat keinerlei Grenzen für Dateigröße oder -anzahl. Ob nginx die Anfragegröße begrenzt (`client_max_body_size`), ist im Repository nicht ersichtlich (Annahme: nicht oder großzügig konfiguriert).

RISIKO

Ein anonymer Angreifer kann mit wenigen parallelen 50-MB-Anfragen Speicher und CPU des einzelnen Node-Prozesses erschöpfen und das Portal für alle Kunden unerreichbar machen; angemeldete Nutzer können zusätzlich den Datenträger mit Uploads füllen. Ist nginx restriktiv konfiguriert, entfällt ein Großteil des Risikos.

EMPFEHLUNG

Das JSON-Limit global auf einen realistischen Wert setzen (z. B. `express.json({ limit: '100kb' })`), für `multer` `limits: { fileSize, files }` festlegen und die Größe zusätzlich in nginx begrenzen. Den Request-Body nicht mehr loggen.

AUSZUG

```
app.use(express.json({ limit: '50mb' }));
```

SEC-15

Uploads überschreiben Dokumente anderer Kunden und sind unbegrenzt groß

MITTEL

CVSS 5,4 `AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:L` · OWASP A06:2025 Insecure Design · CWE-434 · Eingaben und Dateien · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/documents.ts:10`
`src/routes/documents.ts:11`
`src/routes/documents.ts:13`

BEFUND

Alle Kunden speichern in dasselbe Verzeichnis, der Dateiname auf dem Server ist der vom Browser übermittelte Originalname. `multer` ist ohne `limits` und ohne `fileFilter` konfiguriert, Größe, Anzahl und Dateityp sind also nicht begrenzt. Ein Ausbruch aus dem Verzeichnis beim Upload ist nicht möglich, weil der Upload-Parser Pfadanteile aus dem Namen entfernt.

RISIKO

Ein angemeldeter Nutzer lädt eine Datei mit dem Namen eines Dokuments eines anderen Kunden hoch (z. B. `Vertrag.pdf`) und ersetzt es; der andere Kunde erhält beim Download die untergeschobene Datei. Große oder viele Uploads können den Speicherplatz des Servers füllen.

EMPFEHLUNG

Dateien unter einem serverseitig erzeugten Namen (z. B. UUID) in einem Unterordner je Kunde ablegen und den Originalnamen nur in der Datenbank führen. In `multer` `limits` (`fileSize`, `files`) und einen `fileFilter` für die erlaubten Dateitypen setzen.

SEC-15 · Fortsetzung · Uploads überschreiben Dokumente anderer Kunden und sind unbegrenzt groß

AUSZUG

```
destination: (_req, _file, cb) ⇒ cb(null, config.uploadDir),
filename: (_req, file, cb) ⇒ cb(null, file.originalname),
...
const upload = multer({ storage });
```

SEC-16

Fehlermeldungen geben Stacktraces und interne Details an Clients heraus

MITTEL

CVSS 5,3 `AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N` · OWASP A10:2025 Mishandling of Exceptional Conditions · CWE-209 · Fehlerbehandlung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/middleware/error.ts:5`
`src/middleware/error.ts:7`
`src/server.ts:26`

BEFUND

Der zentrale Fehlerbehandler sendet bei jedem Fehler `err.message` und den vollständigen `err.stack` als JSON an den Aufrufer, unabhängig von `NODE_ENV`. Er greift für alle Routen, auch für anonyme: Bereits ungültiges JSON an `POST /api/auth/login` löst einen Parserfehler aus, der mit Stacktrace beantwortet wird; ebenso Datenbankfehler (z. B. aus `POST /api/auth/register` oder der Bestellsuche).

RISIKO

Ein anonymer Angreifer erfährt ohne Aufwand Installationspfade, eingesetzte Bibliotheken samt Versionen und – bei Datenbankfehlern – Tabellen- und Spaltennamen sowie SQL-Fehlermeldungen. Das erleichtert gezielte Angriffe, insbesondere das Ausnutzen der SQL-Injection in der Bestellsuche über Fehlermeldungen.

EMPFEHLUNG

In `errorHandler` nur eine generische Meldung und eine Fehler-ID zurückgeben, Details ausschließlich serverseitig loggen. Den Statuscode aus dem Fehler übernehmen (`err.status`, z. B. 400/413 bei Parserfehlern) statt pauschal 500.

AUSZUG

```
res.status(500).json({
  error: err.message,
  stack: err.stack,
  path: req.path,
});
```

SEC-17

Passwörter sind mit MD5 ohne Salt gespeichert

MITTEL

CVSS 5,3 `AV:N/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N` · OWASP A04:2025 Cryptographic Failures · CWE-916 · Kryptografie · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/auth/password.ts:5`
`src/auth/password.ts:9`
`src/routes/admin.ts:10`

SEC-17 · Fortsetzung · Passwörter sind mit MD5 ohne Salt gespeichert

BEFUND `hashPassword` bildet einen einfachen MD5-Hash ohne Salt (`password.ts:5`), der Vergleich ist nicht zeitkonstant (`:9`). Der Admin-Endpunkt `/api/admin/users` gibt die Hashes per `SELECT *` mit aus (`admin.ts:10`).

RISIKO Wer an die Hashes gelangt – über den Admin-Endpunkt, ein Backup oder eine Injection –, ermittelt die meisten Passwörter per Nachschlagetabelle oder Grafikkarte in kurzer Zeit und kann sie auch bei anderen Diensten der Nutzer ausprobieren.

EMPFEHLUNG Auf Argon2id oder bcrypt umstellen. Bestehende Konten ohne Zwangs-Reset migrieren: den MD5-Hash einmalig in Argon2id einpacken und beim nächsten Login durch einen direkten Hash ersetzen. Hashes nie über die API ausgeben.

AUSZUG

```
return crypto.createHash('md5').update(password).digest('hex');
```

SEC-18

Registrierte E-Mail-Adressen lassen sich von außen abfragen

MITTEL

CVSS 5,3 `AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N` · OWASP A07:2025 Authentication Failures · CWE-204 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/auth.ts:33`
`src/routes/auth.ts:36`
`src/routes/auth.ts:51`
`src/middleware/error.ts:6`

BEFUND Der Login meldet „Unbekannte E-Mail-Adresse“ bzw. „Falsches Passwort“ (`auth.ts:33` , `:36`), der Passwort-Reset antwortet bei unbekannter Adresse mit 404 (`:51`). Bei der Registrierung einer vorhandenen Adresse gibt die Fehlerbehandlung die Datenbankmeldung zur Eindeutigkeit aus (`error.ts:6`).

RISIKO Ein anonymer Angreifer prüft Adresslisten und erfährt, wer Kunde der Muster AG ist. Das erleichtert gezieltes Phishing und die Angriffe auf Login und Passwort-Reset.

EMPFEHLUNG Den Login immer mit derselben Meldung beantworten („E-Mail oder Passwort falsch“), den Passwort-Reset immer mit 200 und neutralem Text. Bei der Registrierung keine Datenbankmeldung ausgeben; mit E-Mail-Bestätigung erfährt nur der Inhaber, dass die Adresse schon existiert.

SEC-19

Weitere Laufzeitbibliotheken mit bekannten Schwachstellen

MITTEL

CVSS 5,3 `AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L` · OWASP A03:2025 Software Supply Chain Failures · CWE-1395 · Dependencies · Quelle: osv-scanner · ↑ Bewertung nach Bereichen

FUNDSTELLE

`package.json:14`
`package.json:19`
`package.json:22`

SEC-19 · Fortsetzung · Weitere Laufzeitbibliotheken mit bekannten Schwachstellen

BEFUND	nodemailer 6.4.11, axios 0.21.1 und lodash 4.17.15 haben zusammen über fünfzig veröffentlichte Advisories. Erreichbar sind vor allem die Lastangriffe im Adressparser von nodemailer (die E-Mail-Adresse aus der Registrierung geht ungeprüft an `sendMail`) und die Proxy- und Weiterleitungsschwächen von axios im Webhook-Test. Die Befehlsinjektion in nodemailer (CVE-2020-7769) betrifft nur den Sendmail-Transport, der hier nicht genutzt wird.
RISIKO	Ein anonymer Angreifer kann über präparierte Adressen bei der Registrierung den Prozess spürbar ausbremsen; die axios-Schwächen verstärken die Webhook-Lücke.
EMPFEHLUNG	nodemailer auf ≥ 10.1.4, axios auf ≥ 1.18.0 und lodash auf 4.18 oder neuer aktualisieren; Dependabot oder Renovate für Laufzeitpakete einschalten.

SEC-20

Klartext-Passwörter landen bei jedem Login im Log

MITTEL

CVSS 5,1 `AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:L/A:N` · OWASP A09:2025 Security Logging and Alerting Failures · CWE-532 · Protokollierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE	`src/routes/auth.ts:29`
BEFUND	Der Login schreibt den kompletten Request-Body per `console.log` ins Log (`auth.ts:29`), also E-Mail-Adresse und Passwort im Klartext – auch bei Fehlversuchen, die oft Passwörter anderer Dienste enthalten.
RISIKO	Wer Zugriff auf Server-Logs, pm2-Logdateien oder einen angebundenen Logdienst hat, erhält gültige Zugangsdaten aller Nutzer, die sich angemeldet haben, einschließlich Administratoren.
EMPFEHLUNG	Die Zeile entfernen und nur ein strukturiertes Ereignis ohne Passwort protokollieren (Ergebnis, Nutzer-ID, IP). Vorhandene Logs löschen und betroffene Nutzer zum Passwortwechsel auffordern.
AUSZUG	`console.log('Login-Versuch', req.body);`

SEC-21

Zugangsdaten aus der ersten Version sind weiterhin in der Git-Historie lesbar

MITTEL

CVSS 4,9 `AV:N/AC:H/PR:L/UI:N/S:C/C:L/I:L/A:N` · OWASP A02:2025 Security Misconfiguration · CWE-540 · Geheimnisse · Quelle: gitleaks · ↑ Bewertung nach Bereichen

FUNDSTELLE	`.env:2 (Commit 8ba4686)`
	`.env:4 (Commit 8ba4686)`
	`.env:5 (Commit 8ba4686)`
	`.env:6 (Commit 8ba4686)`

SEC-21 · Fortsetzung · Zugangsdaten aus der ersten Version sind weiterhin in der Git-Historie lesbar

BEFUND

Der erste Commit (8ba4686, 12.03.2026) enthielt eine `.env` mit JWT-Schlüssel, SMTP-Benutzer und -Passwort sowie dem API-Schlüssel eines Geodienstes. Die Datei wurde im Folgecommit gelöscht, ist in der Historie aber lesbar; die automatische Secret-Suche meldet nur den API-Schlüssel, die übrigen Werte haben wir bei der Durchsicht gefunden.

RISIKO

Wer Lesezugriff auf das Repository oder einen Klon hat, kann Mails im Namen des Portals versenden und den Geodienst auf Kosten der Muster AG nutzen. Ob die Werte noch gültig sind, ist von außen nicht prüfbar; der JWT-Schlüssel ist im Befund zum Fallback-Schlüssel bewertet.

EMPFEHLUNG

SMTP-Passwort und Geodienst-Schlüssel beim jeweiligen Anbieter widerrufen und neu ausstellen, den JWT-Schlüssel wie im Befund zum Fallback-Schlüssel neu erzeugen – das Löschen der Datei genügt nicht. Danach die Historie mit `git filter-repo` bereinigen und eine Secret-Suche als Pre-Commit-Hook und in der CI einbinden.

SEC-22

Anmeldetokens bleiben unbegrenzt gültig und lassen sich nicht widerrufen

MITTEL

CVSS 4,8 `AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N` · OWASP A07:2025 Authentication Failures · CWE-613 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/auth/jwt.ts:13`
`src/auth/jwt.ts:20`
`src/middleware/auth.ts:20`
`src/middleware/auth.ts:28`

BEFUND

Ein Ablauf wird nur gesetzt, wenn `JWT_EXPIRES_IN` konfiguriert ist (`jwt.ts:13`); sonst gilt ein Token unbegrenzt. Es gibt weder Logout noch Sperrliste, und Rolle und Kundenzuordnung stammen nur aus dem Token (`middleware/auth.ts:20` , `:28`), nicht aus der Datenbank.

RISIKO

Ein einmal abgeflossenes Token bleibt dauerhaft nutzbar. Ein entzogenes Admin-Recht oder ein gelöschter Nutzer wirkt erst mit Ablauf des Tokens – ohne gesetzte Variable nie.

EMPFEHLUNG

Den Ablauf im Code fest vorgeben (z. B. 15–60 Minuten mit Refresh-Token) und in `verifyToken` `algorithms: ['HS256']` und `maxAge` setzen. Für Widerruf eine Token-Version je Nutzer speichern und in `requireAuth` zusammen mit Rolle und Status aus der Datenbank prüfen.

SEC-23

Token-Bibliothek mit bekannten Schwächen bei der Signaturprüfung

MITTEL

CVSS 4,8 `AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N` · OWASP A03:2025 Software Supply Chain Failures · CWE-347 · Dependencies · Quelle: osv-scanner · ↑ Bewertung nach Bereichen

FUNDSTELLE

`package.json:18`
`src/auth/jwt.ts:20`

SEC-23 · Fortsetzung · Token-Bibliothek mit bekannten Schwächen bei der Signaturprüfung

BEFUND jsonwebtoken 8.5.1 ist von CVE-2022-23539, CVE-2022-23540 und CVE-2022-23541 betroffen. `verifyToken` ruft `jwt.verify` ohne feste Algorithmus-Liste auf. Weil immer ein Secret gesetzt ist, ist der bekannte Umgehungsweg derzeit nicht direkt nutzbar.

RISIKO Ändert sich die Schlüsselkonfiguration (z. B. leeres Secret oder Umstieg auf Schlüsselpaare), kann ein Angreifer selbst signierte Tokens unterschieben und sich als beliebiger Nutzer anmelden.

EMPFEHLUNG jsonwebtoken auf 9.0.x aktualisieren und in `verifyToken` `algorithms: ['HS256']` festlegen.

SEC-24

Container würde als root mit dem gesamten Arbeitsverzeichnis laufen

NIEDRIG

CVSS 3,9 `AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:L/A:L` · OWASP A02:2025 Security Misconfiguration · CWE-250 · Container · Quelle: semgrep, trivy · ↑ Bewertung nach Bereichen

FUNDSTELLE

```
Dockerfile:1
Dockerfile:4
Dockerfile:5
Dockerfile:9
```

BEFUND Das Dockerfile setzt keinen `USER`, die Anwendung läuft daher als root. `COPY . .` übernimmt ohne `.dockerignore` den kompletten Build-Kontext, bei einem lokalen Build also auch eine vorhandene `.env`, `data/` und `uploads/` in eine Image-Schicht; `npm install` installiert zudem Entwicklungsabhängigkeiten ins Image. Laut Pipeline wird produktiv per rsync/pm2 deployt, das Dockerfile ist dort derzeit offenbar nicht im Einsatz.

RISIKO Falls das Image eingesetzt wird, erhält ein Angreifer, der Code im Container ausführen oder über die Pfadfehler beim Dokumenten-Up- und -Download Dateien schreiben bzw. lesen kann, root-Rechte im Container und damit Zugriff auf alle Dateien inklusive Anwendungscode. Geheimnisse aus einer mitkopierten `.env` wären für jeden mit Zugriff auf das Image lesbar.

EMPFEHLUNG Multi-Stage-Build mit `npm ci` im Build-Schritt und `npm ci --omit=dev` im Laufzeit-Image, am Ende `USER node`; eine `.dockerignore` mit `.env`, `data/`, `uploads/`, `.git` anlegen. Basis-Image auf eine unterstützte Node-Version mit fester Version oder Digest umstellen und einen `HEALTHCHECK` ergänzen. Wird das Dockerfile nicht genutzt, entfernen.

AUSZUG

```
FROM node:16
WORKDIR /app
COPY . .
RUN npm install && npm run build
ENV NODE_ENV=production
CMD ["npm", "start"]
```

SEC-25

Deploy-Verbindung prüft die Identität des Produktivservers nicht

NIEDRIG

CVSS 3,7 `AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N` · OWASP A08:2025 Software or Data Integrity Failures · CWE-295 · CI/CD · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`.github/workflows/deploy.yml:21`
`.github/workflows/deploy.yml:22`

BEFUND

`rsync` und `ssh` laufen mit `-o StrictHostKeyChecking=no`, der Host-Schlüssel des Servers wird also nie geprüft. Der private Deploy-Schlüssel wird zudem als Datei `key` im Arbeitsverzeichnis abgelegt und nicht wieder entfernt.

RISIKO

Kann ein Angreifer die Verbindung des Runners umleiten (z. B. über DNS), gibt sich sein Server als Produktivserver aus und erhält den kompilierten Anwendungscode; das Deployment erreicht den echten Server dann nicht. Den privaten Schlüssel selbst erhält er dadurch nicht.

EMPFEHLUNG

Den Host-Schlüssel des Servers als Secret oder Variable hinterlegen und vor dem Deploy in `~/.ssh/known_hosts` schreiben, `StrictHostKeyChecking=no` entfernen. Den Schlüssel per `ssh-agent` laden statt als Datei im Workspace abzulegen.

AUSZUG

```
echo "${{ secrets.DEPLOY_KEY }}" > key
chmod 600 key
rsync -e "ssh -i key -o StrictHostKeyChecking=no" ...
```

SEC-26

Deploy-Pipeline nutzt veränderliche Action-Versionen und unbegrenzte Rechte

NIEDRIG

CVSS 3,3 `AV:N/AC:H/PR:H/UI:N/S:U/C:L/I:L/A:N` · OWASP A03:2025 Software Supply Chain Failures · CWE-1357 · CI/CD · Quelle: semgrep · ↑ Bewertung nach Bereichen

FUNDSTELLE

`.github/workflows/deploy.yml:8`
`.github/workflows/deploy.yml:11`
`.github/workflows/deploy.yml:12`

BEFUND

`actions/checkout@v3` und `actions/setup-node@v3` sind über verschiebbare Tags statt über Commit-Hashes eingebunden; der Workflow definiert keinen `permissions:`-Block, die Rechte des `GITHUB_TOKEN` hängen damit von der Repository-Einstellung ab (Annahme: Standard). Jeder Push auf `main` geht ohne Prüfschritt (Tests, Freigabe über `environment`) direkt in Produktion; ob Branch-Schutz besteht, ist im Repository nicht ersichtlich.

RISIKO

Wird ein Tag einer eingebundenen Action umgehängt, läuft fremder Code im selben Job, in dem der Deploy-Schlüssel auf dem Runner liegt. Das Risiko ist bei den von GitHub selbst gepflegten Actions gering, setzt Kontrolle über deren Repository voraus und ist daher als Härtung eingestuft.

EMPFEHLUNG

Actions auf vollständige Commit-SHA pinnen (Aktualisierung über Dependabot), `permissions: contents: read` setzen und den Deploy-Schritt in ein GitHub-`environment` mit Freigabe und vorgeschaltetem Test-Job verschieben. Branch-Schutz für `main` sicherstellen.

SEC-26 · Fortsetzung · Deploy-Pipeline nutzt veränderliche Action-Versionen und unbegrenzte Rechte

AUSZUG

```
- uses: actions/checkout@v3
- uses: actions/setup-node@v3
```

SEC-27

Anmeldetokens werden auch in der URL angenommen

NIEDRIG

CVSS 3,1 `AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N` · OWASP A07:2025 Authentication Failures · CWE-598 · Authentifizierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/middleware/auth.ts:15`

BEFUND

`requireAuth` akzeptiert das Token außer im `Authorization`-Header auch als Query-Parameter `?token=` (`middleware/auth.ts:15`).

RISIKO

URLs landen in Zugriffslogs von nginx und Proxys, im Browserverlauf und im `Referer` an Dritte. Wer dort ein Token findet, handelt im Namen des Nutzers, bei fehlendem Ablauf dauerhaft.

EMPFEHLUNG

Das Token nur aus dem `Authorization`-Header lesen. Für Downloads, die ohne Header auskommen müssen, kurzlebige, einmal nutzbare Download-Links ausstellen.

AUSZUG

```
const token = header.startsWith('Bearer ') ? header.slice(7) : req.query.token;
```

SEC-28

CORS erlaubt ohne Konfiguration Zugriffe von beliebigen Webseiten

NIEDRIG

CVSS 3,1 `AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N` · OWASP A02:2025 Security Misconfiguration · CWE-942 · CORS · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/server.ts:14`
`src/config.ts:9`

BEFUND

Ist `CORS_ORIGIN` nicht gesetzt, spiegelt die CORS-Konfiguration (`origin: true`) jede anfragende Herkunft zurück und erlaubt dabei Credentials. Da die Anmeldung derzeit ausschließlich über einen Bearer-Token im Header bzw. Query-Parameter erfolgt und keine Cookies verwendet werden, kann eine fremde Seite damit aktuell keine Daten im Namen eines Nutzers abrufen. Ob `CORS_ORIGIN` in Produktion gesetzt ist, ist nicht ersichtlich (Annahme; `.env.example` sieht es vor).

RISIKO

Kein unmittelbarer Angriffspfad. Sobald künftig Cookies oder andere automatisch mitgesendete Anmeldedaten eingeführt werden, könnte jede Webseite, die ein angemeldeter Nutzer besucht, Bestellungen und Rechnungen auslesen.

EMPFEHLUNG

Ohne gesetzte Variable keinen Fallback auf `true` verwenden, sondern den Start abbrechen oder eine feste Allowlist (`origin: ['https://portal.muster-ag.example']`) nutzen; `credentials: true` nur setzen, wenn tatsächlich benötigt.

SEC-28 · Fortsetzung · CORS erlaubt ohne Konfiguration Zugriffe von beliebigen Webseiten

AUSZUG

```
app.use(cors({ origin: config.corsOrigin || true, credentials: true }));
```

SEC-29

Sicherheits-Header fehlen, Server verrät Technik und Version

NIEDRIG

CVSS 3,1 `AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N` · OWASP A02:2025 Security Misconfiguration · CWE-693 · Security-Header · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/server.ts:12`
`src/server.ts:17`

BEFUND

Die Anwendung setzt keine Sicherheits-Header (kein `helmet`, kein HSTS, kein `X-Content-Type-Options`, keine Frame-Optionen) und lässt den Standard-Header `X-Powered-By: Express` aktiv; `GET /health` gibt anonym die Anwendungsversion aus. Ob nginx diese Header ergänzt, ist im Repository nicht ersichtlich (Annahme).

RISIKO

Einzeln kein direkter Angriffspfad, da die API JSON liefert und Dateien als Download ausgibt. Fehlendes HSTS erleichtert jedoch Downgrade-Angriffe auf die Verbindung des Browsers, und Technik- und Versionsangaben erleichtern die Auswahl passender Exploits.

EMPFEHLUNG

`helmet()` einbinden bzw. `app.disable('x-powered-by')` setzen, HSTS und `X-Content-Type-Options: nosniff` in Anwendung oder nginx sicherstellen und die Version aus `/health` entfernen.

AUSZUG

```
app.get('/health', (_req, res) => res.json({ ok: true, version: process.env.npm_package_version
}));
```

SEC-30

Admin-Nutzerliste gibt Passwort-Hashes aus

NIEDRIG

CVSS 2,7 `AV:N/AC:L/PR:H/UI:N/S:U/C:L/I:N/A:N` · OWASP A01:2025 Broken Access Control · CWE-200 · Autorisierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/admin.ts:10`

BEFUND

`GET /api/admin/users` liefert `SELECT * FROM users`, also auch die Spalte `password_hash`. Für die Verwaltung der Nutzer wird der Hash nicht gebraucht.

RISIKO

Wer Admin-Rechte hat oder ein Admin-Token erlangt, erhält die Hashes aller Konten und kann sie offline knacken; zusammen mit der offenen Registrierung als Admin ist das ohne Vorwissen möglich.

EMPFEHLUNG

Nur die benötigten Spalten ausliefern (`SELECT id, customer_id, email, role, created_at FROM users`); generell `SELECT *` in API-Antworten vermeiden.

AUSZUG

```
res.json(db.prepare('SELECT * FROM users').all());
```

KAPITEL 4

Qualität, Betrieb und Dependencies

Jeder Bereich ist auf vier Stufen bewertet, seine Befunde folgen direkt darunter. Qualitätsbefunde sind priorisiert, aber nicht nach CVSS bewertet.

4.1 Code-Qualität und Wartbarkeit

■■□□ Handlungsbedarf

SQL steht direkt in den Request-Handlern, und für Bestellungen gibt es zwei parallele Zugriffswege. Deshalb muss die Mandantenprüfung an jeder Stelle von Hand mitgeschrieben werden – genau dort entstehen die Lücken dieses Reports. Fehler beim Mailversand werden verschluckt oder hinterlassen halbe Zustände. Der strikte TypeScript-Modus ist aktiv, wird durch untypisierte Datenbankzeilen aber weitgehend umgangen.

QA-06

Datenbankzugriff steckt direkt in den Routen, zwei Zugriffsstile nebeneinander

■■□□ MITTEL

Architektur · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/orders.ts:12`
`src/routes/orders.ts:31`
`src/routes/invoices.ts:24`
`src/routes/admin.ts:10`
`src/db.ts:10`

BEFUND

Alle Router enthalten SQL direkt im Request-Handler; eine Schicht für Fachlogik gibt es nicht. Gleichzeitig nutzen zwei Bestell-Endpunkte ein generisches Repository, die übrigen Endpunkte derselben Datei rohes SQL – innerhalb von `orders.ts` gibt es also zwei Arten, Bestellungen zu lesen. Das Schema wird beim Start per `CREATE TABLE IF NOT EXISTS` angelegt; Migrationen für spätere Schemaänderungen gibt es nicht.

RISIKO

Regeln wie die Mandantenprüfung müssen in jeder Abfrage einzeln von Hand mitgeschrieben werden und fehlen deshalb stellenweise; Änderungen an einer Tabelle betreffen verstreute Stellen. Spaltenänderungen an einer bestehenden Produktionsdatenbank sind ohne Migrationen nur manuell möglich.

EMPFEHLUNG

Je Fachbereich ein schlankes Datenmodul mit benannten Funktionen anlegen (z. B. `findOrdersForCustomer(customerId)`, `findInvoice(id, customerId)`), in dem der Mandantenfilter Pflichtparameter ist; die Router rufen nur noch diese Funktionen auf. Für das Schema ein einfaches Migrationswerkzeug einführen (nummerierte SQL-Dateien plus `user_version` von SQLite genügen).

AUSZUG

```
const orders = db
  .prepare('SELECT * FROM orders WHERE customer_id = ? ORDER BY created_at DESC')
  .all(customerId) as any[];
...
const order = repositories.get('order').findById(Number(req.params.id));
```

QA-07

Eingaben werden nirgends an der Schnittstelle geprüft

MITTEL

OWASP A06:2025 Insecure Design · CWE-20 · Eingabevalidierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/server.ts:15`
`src/routes/auth.ts:11`
`src/routes/webhooks.ts:14`
`src/routes/admin.ts:19`

BEFUND

Keine Route prüft Request-Body oder Query gegen ein Schema; Werte werden direkt destrukturiert und an Datenbank, Mailversand oder HTTP-Client weitergegeben. Der JSON-Parser akzeptiert Bodies bis 50 MB, auch auf den anonymen Routen. `POST /api/admin/settings` verschmilzt den Body per `_.merge` mit Standardwerten, speichert das Ergebnis aber nicht.

RISIKO

Fehlende Typ- und Feldprüfung ist die gemeinsame Ursache der Befunde zu Registrierung, Webhooks und Suche und begünstigt neue Lücken bei jeder Erweiterung. Große JSON-Bodies blockieren den Prozess beim Parsen.

EMPFEHLUNG

An jeder Route ein Schema (z. B. zod oder express-validator) mit erlaubten Feldern, Typen und Längen durchsetzen und unbekannte Felder verwerfen. Das JSON-Limit auf einen realistischen Wert wie `100kb` senken; die funktionslose Route `/api/admin/settings` entfernen oder ohne `_.merge` mit fremden Daten umsetzen.

AUSZUG

```
app.use(express.json({ limit: '50mb' }));
```

QA-08

Fehler beim Mailversand gehen verloren oder hinterlassen halbe Zustände

MITTEL

CWE-755 · Fehlerbehandlung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/core/EventBus.ts:19`
`src/core/EventBus.ts:30`
`src/routes/auth.ts:18`
`src/routes/auth.ts:21`
`src/routes/auth.ts:47`

BEFUND

Der Event-Bus verwirft jeden Fehler seiner Handler mit `.catch(() => undefined)`: Scheitert die Stornierungsmail, gibt es weder Log noch Wiederholung. Bei der Registrierung wird der Nutzer erst gespeichert, dann die Willkommensmail abgewartet; scheitert der Versand, erhält der Client einen Fehler 500, obwohl das Konto existiert. Der Reset-Handler ist `async`, reicht Fehler aber nicht an `next` weiter (siehe Sicherheitsbefund zum Absturz über den Passwort-Reset).

RISIKO

Kunden erhalten Stornobestätigungen nicht, ohne dass es jemand bemerkt; Registrierungen hängen bei Mailstörungen in einem inkonsistenten Zustand, der nur durch manuelles Eingreifen in der Datenbank auflösbar ist.

QA-08 · Fortsetzung · Fehler beim Mailversand gehen verloren oder hinterlassen halbe Zustände

EMPFEHLUNG Im Event-Bus bzw. im späteren direkten Aufruf Fehler mindestens protokollieren (`console.error` bzw. ein Logger mit Bestell-ID). Mailversand bei der Registrierung vom Request entkoppeln (nach erfolgreichem Insert senden, Fehler loggen statt 500) und alle `async` -Handler einheitlich über einen Wrapper wie `asyncHandler(fn)` an `next` anbinden oder auf Express 5 wechseln, das abgelehnte Promises selbst weiterreicht.

AUSZUG

```
for (const handler of this.handlers[event] || []) {
  Promise.resolve(handler(payload)).catch(() => undefined);
}
```

QA-11

Sicherheitsrelevante Ereignisse werden nicht protokolliert

MITTEL

OWASP A09:2025 Security Logging and Alerting Failures · CWE-778 · Protokollierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE
`src/routes/auth.ts:28`
`src/routes/auth.ts:47`
`src/routes/admin.ts:17`
`src/routes/admin.ts:23`

BEFUND Erfolgreiche und fehlgeschlagene Logins, Passwort-Resets, Registrierungen mit Rolle und Admin-Aktionen wie der Nutzerexport (`admin.ts:23`) hinterlassen keinen Protokolleintrag. Der einzige Log-Aufruf im Login ist die Klartextausgabe des Request-Bodys.

RISIKO Angriffe auf Konten oder ein Datenabfluss über den Export fallen nicht auf und lassen sich im Nachhinein nicht nachvollziehen; das erschwert auch eine Meldung nach Art. 33 DSGVO.

EMPFEHLUNG Ein strukturiertes Audit-Log (z. B. mit `pino`) für Anmeldung, Reset, Registrierung, Rollenänderung, Export und Admin-Aufrufe mit Zeit, Nutzer-ID, Mandant und IP einführen, getrennt aufbewahren und bei Auffälligkeiten wie gehäuften Fehlversuchen alarmieren.

QA-12

Datenbankzeilen und Eingaben sind untypisiert, der strikte TypeScript-Modus wird umgangen

NIEDRIG

Lesbarkeit und Typisierung · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE
`src/routes/auth.ts:11`
`src/routes/auth.ts:31`
`src/routes/orders.ts:14`
`src/routes/invoices.ts:35`
`src/core/EventBus.ts:27`
und 2 weitere Stellen gleicher Art

QA-12 · Fortsetzung · Datenbankzeilen und Eingaben sind untypisiert, der strikte TypeScript-Modus wird umgangen

BEFUND	`tsconfig.json` aktiviert `strict`, Ergebnisse aus der Datenbank werden aber fast überall mit `as any` bzw. `as any[]` weiterverwendet, und Request-Bodies werden ohne Schema destrukturiert. Die Typprüfung erkennt damit weder Tippfehler in Spaltennamen noch fehlende oder falsch typisierte Felder in Anfragen.
RISIKO	Fehler wie ein falscher Spaltenname oder ein String statt einer Zahl fallen erst zur Laufzeit auf; die Sicherheit, die der strikte Modus verspricht, ist faktisch nicht vorhanden.
EMPFEHLUNG	Für jede Tabelle ein Zeilen-Interface anlegen (wie `Order` in `OrderRepository.ts`) und Abfragen damit typisieren (`db.prepare<[number], Invoice>(...)`). Request-Bodies mit einer Schema-Bibliothek wie `zod` prüfen und daraus die Typen ableiten; ESLint-Regel `@typescript-eslint/no-explicit-any` aktivieren.
AUSZUG	`const user = db.prepare('SELECT * FROM users WHERE email = ?').get(email) as any;`

4.2 Komplexität und Umfang

■■■□ Gut

Keine Funktion erreicht eine erhöhte zyklomatische Komplexität, und es gibt weder überlange Funktionen noch lange Parameterlisten. Für Bestellungen gibt es dennoch ein generisches Repository mit Fabrik und einen eigenen Event-Bus, obwohl jeweils nur ein Anwendungsfall existiert. Beides lässt sich ohne fachlichen Verlust durch wenige direkte Funktionen ersetzen; das spart rund 100 der knapp 600 Quelltextzeilen, etwa ein Sechstel.

KENNZAHLEN ZU KOMPLEXITÄT UND UMFANG

Gemessen mit lizard über den Produktivcode, ohne Tests, Abhängigkeiten und generierten Code. Die zyklomatische Komplexität (CCN) zählt die unabhängigen Pfade durch eine Funktion.

KENNZAHL	RICHTWERT	TYPESCRIPT	GESAMT
Funktionen		48	48
Zeilen Code (NLOC)		309	309
Komplexität Mittel	≤ 5	1,9	1,9
Komplexität 90. Perzentil	≤ 10	3	3
Komplexität Maximum	≤ 20	7	7
Funktionen mit CCN > 10	< 5 %	0 0,0 %	0 0,0 %
Funktionen mit CCN > 20	< 1 %	0 0,0 %	0 0,0 %
Funktionen > 60 Zeilen	< 2 %	0 0,0 %	0 0,0 %
Funktionen > 5 Parameter	< 1 %	0 0,0 %	0 0,0 %
Duplizierter Code	< 5 %		0,0 %
Testanteil (Testzeilen an allen Zeilen)	≥ 30 %		8,8 %

QA-09

Generische Repository-Schicht mit Fabrik für zwei einfache Abfragen

MITTEL

Komplexität und Umfang · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/core/AbstractRepository.ts:12`
`src/core/AbstractRepository.ts:20`
`src/core/AbstractRepository.ts:41`
`src/core/OrderRepository.ts:11`
`src/core/RepositoryFactory.ts:8`
und 3 weitere Stellen gleicher Art

BEFUND

Für Bestellungen gibt es eine abstrakte Basisklasse, eine einzige Ableitung und eine Fabrik mit Registry und Instanz-Cache, die genau ein Repository (`'order'`) kennt. Genutzt werden davon nur `findById` und `update` in `orders.ts` ; `findAll` und `map` werden nirgends aufgerufen. Gegenprobe: keine zweite Implementierung, kein Test, der das Repository ersetzt, keine weitere Datenbank – alle übrigen Tabellen werden direkt per SQL gelesen.

QA-09 · Fortsetzung · Generische Repository-Schicht mit Fabrik für zwei einfache Abfragen

RISIKO

Die Schicht verdoppelt die Zugriffswege für Bestellungen und verlangt beim Lesen drei Indirektionen für eine Abfrage per Primärschlüssel. Das ungenutzte `findAll` setzt Sortierung und Limit ungeprüft in SQL ein und wäre bei künftiger Verwendung mit Benutzereingaben ein Einfallstor; toter Code dieser Art wird leicht unbedacht wiederverwendet.

EMPFEHLUNG

Die drei Dateien durch zwei benannte Funktionen ersetzen, z. B. `findOrderForCustomer(id, customerId)` und `setOrderStatus(id, status)` mit je einem vorbereiteten Statement – geschätzt 89 Zeilen → 10–15 Zeilen, Einsparung etwa 75–80 Zeilen. Der Mandantenfilter gehört dabei direkt in die Abfrage statt in den Vergleich im Router.

AUSZUG

```
type RepositoryName = 'order';
...
export const repositories = new RepositoryFactory().register('order', OrderRepository);
```

QA-13

Eigener Event-Bus für genau ein Ereignis mit genau einem Empfänger

NIEDRIG

Komplexität und Umfang · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/core/EventBus.ts:6`
`src/core/EventBus.ts:10`
`src/core/EventBus.ts:26`
`src/routes/orders.ts:45`

BEFUND

`EventBus.ts` implementiert einen typisierten Publish/Subscribe-Mechanismus (32 Zeilen) für ein einziges Ereignis (`order.cancelled`), das genau einen, in derselben Datei registrierten Empfänger hat und nur an einer Stelle ausgelöst wird. Gegenprobe: keine weiteren Ereignisse, keine Webhook-Auslieferung über den Bus (die Tabelle `webhooks` wird nicht angebunden), keine prozessübergreifende Zustellung, keine Tests.

RISIKO

Der Umweg verschleiert, was beim Stornieren tatsächlich passiert (Mailversand), und ist der Grund, warum Fehler dabei unbemerkt verworfen werden (siehe Befund zu verlorenen Fehlern beim Mailversand). Ohne zweiten Anwendungsfall bringt die Abstraktion keinen Nutzen.

EMPFEHLUNG

Den Empfänger als Funktion `notifyOrderCancelled(orderId, customerId)` in `services/mail.ts` bzw. einem Bestellmodul ablegen und in `orders.ts` direkt (ohne `await` , mit `.catch(logError)`) aufrufen – geschätzt 32 Zeilen → 10–14 Zeilen, Einsparung etwa 18–22 Zeilen. Einen Bus erst einführen, wenn tatsächlich Webhooks oder weitere Empfänger hinzukommen.

AUSZUG

```
interface EventMap {
  'order.cancelled': { orderId: number; customerId: number };
}
```

QA-14

Einstellungs-Endpunkt ohne Wirkung zieht eine eigene Bibliothek nach sich

NIEDRIG

Komplexität und Umfang · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/admin.ts:2`
`src/routes/admin.ts:17`
`package.json:19`

BEFUND

`POST /api/admin/settings` führt Standardwerte und Request-Body mit `lodash.merge` zusammen und gibt das Ergebnis zurück, speichert aber nichts; `invoicePrefix` und `reminderDays` werden nirgends im Code gelesen. Lodash wird ausschließlich für diesen Aufruf eingebunden. Ob ein Frontend den Endpunkt als Vorschau nutzt, ist aus dem Repository nicht ersichtlich; die bekannten Schwachstellen von lodash sind im Befund zu weiteren Laufzeitbibliotheken bewertet.

RISIKO

Administratoren gehen davon aus, Einstellungen geändert zu haben, die keine Wirkung haben; die zusätzliche Abhängigkeit muss ohne Nutzen gepflegt werden.

EMPFEHLUNG

Endpunkt und Lodash entfernen, oder – falls Einstellungen gebraucht werden – eine Tabelle `settings` anlegen und die Werte mit Objekt-Spread (`{ ...defaults, ...validated }`) nach Schemaprüfung speichern. Einsparung etwa 5–6 Zeilen und eine Laufzeitabhängigkeit.

AUSZUG

```
adminRouter.post('/settings', (req, res) ⇒ {
  const defaults = { invoicePrefix: 'RE-', reminderDays: 14 };
  const settings = _.merge({}, defaults, req.body);
  res.json(settings);
});
```

4.3 Performance und Caching

Handlungsbedarf

Der Rechnungs-Cache ist ohne Kunden-ID im Schlüssel angelegt und wird beim Bezahlt-Markieren nicht invalidiert. Die Quelle, ein lokaler SQLite-Zugriff per Primärschlüssel, ist ohnehin schnell, deshalb sollte er entfallen. Die Bestellübersicht fragt Positionen einzeln je Bestellung ab, und auf den Mandantenspalten fehlen Indizes. Bei den heutigen Datenmengen fällt das noch nicht auf.

QA-04

Bestellübersicht stellt je Bestellung eine eigene Datenbankabfrage

MITTEL

Performance und Caching · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/routes/orders.ts:12`
`src/routes/orders.ts:15`
`src/routes/orders.ts:16`
`src/db.ts:32`

QA-04 · Fortsetzung · Bestellübersicht stellt je Bestellung eine eigene Datenbankabfrage

BEFUND

`GET /api/orders` lädt alle Bestellungen eines Kunden und fragt danach in einer Schleife die Positionen jeder Bestellung einzeln ab (N+1-Muster); das Statement wird dabei in jedem Durchlauf neu vorbereitet. Auf `order_items.order_id` liegt kein Index, SQLite legt für Fremdschlüssel keinen automatisch an – jede dieser Einzelabfragen durchsucht also die gesamte Positionstabelle.

RISIKO

Die Antwortzeit wächst mit Bestellanzahl mal Positionsanzahl; bei Kunden mit langer Historie wird die Übersicht spürbar langsam und blockiert, weil better-sqlite3 synchron arbeitet, in dieser Zeit alle anderen Anfragen des Prozesses (Einschätzung, nicht gemessen).

EMPFEHLUNG

Positionen in einer Abfrage holen (`SELECT * FROM order_items WHERE order_id IN (SELECT id FROM orders WHERE customer_id = ?)`) und im Code nach `order_id` gruppieren, Statements einmal auf Modulebene vorbereiten und `CREATE INDEX order_items_order_id ON order_items(order_id)` ergänzen.

AUSZUG

```
for (const order of orders) {
  order.items = db.prepare('SELECT * FROM order_items WHERE order_id = ?').all(order.id);
}
```

QA-10

Rechnungs-Cache liefert veraltete Zahlungsstände und umgeht jede Mandantenprüfung

MITTEL

CWE-524 · Performance und Caching · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`src/services/cache.ts:5`
`src/config.ts:16`
`src/routes/invoices.ts:20`
`src/routes/invoices.ts:28`
`src/routes/invoices.ts:42`

BEFUND

Der einzige Daten-Cache (`invoiceCache` , im Prozessspeicher, unbegrenzt) speichert vollständige Rechnungen unter der Rechnungs-ID, ohne Kunden-ID im Schlüssel, und wird beim Bezahlt-Markieren nicht invalidiert. Die Quelle ist ein Primärschlüssel-Zugriff auf eine lokale SQLite-Datei und ohnehin schnell. Weil der Cache vor der Datenbankabfrage steht, schließt eine Korrektur nur der SQL-Abfrage die Lücke aus SEC-10 nicht.

RISIKO

Nach `POST /api/invoices/:id/paid` zeigt die Detailansicht bis zu zehn Minuten weiter „unbezahlt“; ein einmal zwischengespeicherter Eintrag wird jedem angemeldeten Nutzer ausgeliefert, unabhängig vom Mandanten.

EMPFEHLUNG

Den Cache ersatzlos entfernen und die Rechnung direkt mit `WHERE id = ? AND customer_id = ?` lesen; `node-cache` und `invoiceCacheTtl` entfallen. Falls später ein Cache nötig wird, die Kunden-ID in den Schlüssel aufnehmen, bei jeder Änderung invalidieren und `maxKeys` setzen.

QA-10 · Fortsetzung · Rechnungs-Cache liefert veraltete Zahlungsstände und umgeht jede Mandantenprüfung

AUSZUG

```
const cached = invoiceCache.get(id);
if (cached) {
  return res.json(cached);
}
const invoice = db.prepare('SELECT * FROM invoices WHERE id = ?').get(id);
```

QA-15

Keine Indizes auf den Mandantenspalten, über die jede Liste gefiltert wird

NIEDRIG

Performance und Caching · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

src/db.ts:26
src/db.ts:41
src/db.ts:50
src/db.ts:57
src/routes/invoices.ts:13
und 1 weitere Stelle gleicher Art

BEFUND

Das Schema legt außer den Primärschlüsseln und `users.email` keine Indizes an. Alle Kundenlisten (Bestellungen, Rechnungen, Dokumente, Webhooks) filtern über `customer_id` und müssen deshalb die jeweilige Tabelle vollständig durchsuchen.

RISIKO

Bei kleinen Datenmengen unauffällig; mit wachsender Kundenzahl werden alle Listenansichten gleichmäßig langsamer, obwohl jeder Kunde nur einen kleinen Teil der Daten sieht (Einschätzung, nicht gemessen).

EMPFEHLUNG

Indizes auf `customer_id` in `orders`, `invoices`, `documents`, `webhooks` und `users` anlegen, für `orders` kombiniert mit `created_at` (`(customer_id, created_at)`), passend zur Sortierung der Übersicht – am besten als erste Migration.

AUSZUG

```
CREATE TABLE IF NOT EXISTS invoices (
  id INTEGER PRIMARY KEY,
  customer_id INTEGER NOT NULL REFERENCES customers(id),
```

QA-16

Listen und Admin-Export laden alle Datensätze auf einmal

NIEDRIG

Performance und Caching · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

src/routes/orders.ts:13
src/routes/invoices.ts:13
src/routes/documents.ts:19
src/routes/admin.ts:10
src/routes/admin.ts:24

QA-16 · Fortsetzung · Listen und Admin-Export laden alle Datensätze auf einmal

BEFUND	Keine der Listenrouten kennt Paging oder ein Limit; die Admin-Routen `/users`, `/customers` und `/export` laden jeweils die ganze Tabelle mit `.all()` in den Speicher und serialisieren sie in eine einzige JSON-Antwort.
RISIKO	Speicherbedarf und Antwortzeit wachsen linear mit dem Datenbestand; ein Export über alle Nutzer blockiert den synchron arbeitenden Prozess für die Dauer der Abfrage (Einschätzung, nicht gemessen).
EMPFEHLUNG	Für Kundenlisten `limit` / `offset` oder besser Keyset-Paging (`WHERE id < ? ORDER BY id DESC LIMIT 50`) einführen; den Export mit `stmt.iterate()` als CSV- oder NDJSON-Stream ausgeben.
AUSZUG	

```
adminRouter.get('/users', (_req, res) => {
  res.json(db.prepare('SELECT * FROM users').all());
});
```

4.4 Tests

■□□□ Ungenügend

Geprüft werden lediglich die Passwortfunktionen und ob die Bestellliste eine Anmeldung verlangt. Für Rechnungen, Dokumente, Webhooks, Admin-Bereich, Registrierung und Passwort-Zurücksetzen fehlen Tests ebenso wie Testdaten mit mehreren Mandanten. Die Jest-Konfiguration für TypeScript fehlt, und die Tests laufen in keiner Pipeline.

QA-02

Kritische Abläufe sind durch Tests praktisch nicht abgesichert

■■■□ HOCH

Tests · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE	`tests/orders.test.ts:7`
	`tests/password.test.ts:3`
	`package.json:11`
	`tsconfig.json:11`

BEFUND	Es gibt zwei Testdateien: Eine prüft, dass `GET /api/orders` eine Anmeldung verlangt, die andere die Passwortfunktionen. Für Registrierung, Passwort-Zurücksetzen, Rechnungen, Dokumente, Webhooks und den Admin-Bereich gibt es keinen Test, und kein Test prüft die Mandantentrennung. Eine Jest-Konfiguration für TypeScript fehlt (`ts-jest` ist installiert, aber nicht eingetragen), der Bestellungstest läuft gegen die echte Datenbankdatei.
RISIKO	Fehler in der Zugriffs- und Mandantenlogik – die in diesem Audit mehrfach gefunden wurden – fallen erst in Produktion auf; jede Änderung an einer Route ist ein Blindflug. Ob die vorhandenen Tests in der jetzigen Konfiguration überhaupt laufen, ist statisch nicht gesichert.
EMPFEHLUNG	Jest mit `preset: 'ts-jest'` konfigurieren und Tests gegen eine In-Memory-SQLite-Datenbank mit zwei Testmandanten laufen lassen (`DB_FILE=:memory:` plus Seed). Zuerst Integrationstests für Mandantentrennung je Route, Rollenprüfung im Admin-Bereich, Anmeldung und Passwort-Zurücksetzen schreiben; danach jede Behebung aus diesem Report mit einem Regressionstest absichern.

QA-02 · Fortsetzung · Kritische Abläufe sind durch Tests praktisch nicht abgesichert

AUSZUG

```
it('liefert die Bestellungen des eigenen Kunden', async () => {
  const res = await request(app).get('/api/orders').set('Authorization', `Bearer ${token}`);
  expect(res.status).toBe(200);
  expect(Array.isArray(res.body)).toBe(true);
});
```

4.5 Build, CI/CD und Betrieb

■■□□ Handlungsbedarf

Jeder Push auf den Hauptzweig wird gebaut und sofort ausgeliefert, ohne Tests, Linter oder einen Pull-Request-Check als Gate. Node.js 16 erhält keine Updates mehr, und weil das Lockfile nicht auf den Server kopiert wird, löst dieser die Abhängigkeiten bei jedem Deploy neu auf. Neben dem tatsächlich genutzten Deploy-Weg existiert ein nicht verwendetes Dockerfile. Die sicherheitsrelevanten Punkte sind als SEC-07 und SEC-24 bis SEC-26 bewertet.

QA-01

Jeder Push auf main geht ohne Tests oder Prüfungen in Produktion

■■■□ HOCH

Build und CI/CD · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`.github/workflows/deploy.yml:4`
`.github/workflows/deploy.yml:15`
`.github/workflows/deploy.yml:16`
`package.json:7`

BEFUND

Die einzige Pipeline baut bei jedem Push auf `main` und kopiert das Ergebnis direkt auf den Produktionsserver. Tests (`npm test`) laufen nicht, ein Linter ist weder konfiguriert noch als Skript vorhanden, und es gibt keine Pipeline für Pull Requests. Einziges Gate ist der TypeScript-Compiler im Build.

RISIKO

Ein fehlerhafter Commit erreicht ohne Vorwarnung das öffentliche Kundenportal; Rückfälle bei bereits behobenen Fehlern werden nicht erkannt. Ob Branch-Schutz oder Reviews auf GitHub-Seite eingerichtet sind, ist aus dem Repository nicht ersichtlich (Annahme: nein).

EMPFEHLUNG

Einen eigenen Prüf-Job für Pull Requests und `main` anlegen (`npm ci`, `tsc --noEmit`, Linter, `npm test`) und den Deploy-Job per `needs:` davon abhängig machen. ESLint mit `@typescript-eslint` einführen und Branch-Schutz mit Pflicht-Check für `main` aktivieren.

AUSZUG

```
on:
  push:
    branches: [main]
...
      - run: npm install
      - run: npm run build
```

QA-03

Abhängigkeiten werden nicht gepflegt, die meisten liegen Hauptversionen zurück

MITTEL

Build und CI/CD · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`package.json:13`
`package.json:24`

BEFUND

Die direkten Abhängigkeiten sind auf Stände von 2019 bis 2022 festgeschrieben (u. a. Express 4.17.1, axios 0.21.1, jsonwebtoken 8.5.1, multer 1.4.2, nodemailer 6.4.11, better-sqlite3 7.6.2) und liegen teils mehrere Hauptversionen hinter dem aktuellen Stand. Ein automatischer Update-Prozess (Dependabot, Renovate) ist nicht eingerichtet. Die daraus folgenden bekannten Schwachstellen sind als Sicherheitsbefunde separat bewertet.

RISIKO

Je länger Updates liegen bleiben, desto größer und riskanter wird der Sprung; Sicherheitskorrekturen lassen sich dann nicht kurzfristig einspielen, weil zuerst Breaking Changes mehrerer Versionen abgearbeitet werden müssen.

EMPFEHLUNG

Dependabot oder Renovate mit wöchentlichen, gruppierten Updates einrichten und Updates nur über den Prüf-Job aus CI mergen. Die Hauptversionssprünge einzeln und testgestützt nachziehen, beginnend mit den Paketen mit bekannten Schwachstellen.

AUSZUG

```
"axios": "0.21.1",
"better-sqlite3": "7.6.2",
"express": "4.17.1",
"jsonwebtoken": "8.5.1",
```

QA-05

Builds sind nicht reproduzierbar und laufen auf einer nicht mehr gepflegten Node-Version

MITTEL

Build und CI/CD · Quelle: Review · ↑ Bewertung nach Bereichen

FUNDSTELLE

`.github/workflows/deploy.yml:14`
`.github/workflows/deploy.yml:15`
`.github/workflows/deploy.yml:21`
`.github/workflows/deploy.yml:22`
`Dockerfile:1`
und 1 weitere Stelle gleicher Art

BEFUND

CI und Dockerfile verwenden Node 16, das seit September 2023 keine Sicherheitsupdates mehr erhält. Die Pipeline nutzt `npm install` statt `npm ci`, und auf den Server gehen nur `dist` und `package.json`, nicht `package-lock.json` – dort werden die indirekten Abhängigkeiten bei jedem Deploy neu aufgelöst. Daneben liegt ein ungenutztes Dockerfile ohne `.dockerignore` (`COPY . .` übernimmt auch eine lokale `.env`).

RISIKO

Produktion kann andere Paketversionen enthalten als die getestete Version, Fehler sind dann nicht nachstellbar. Zwei unterschiedliche, nicht abgeglichene Betriebswege (pm2 per rsync und Docker) erschweren Wartung und Fehlersuche.

QA-05 · Fortsetzung · Builds sind nicht reproduzierbar und laufen auf einer nicht mehr gepflegten Node-Version

EMPFEHLUNG Auf eine aktuelle LTS-Version von Node wechseln und sie in `.nvmrc` bzw. `engines` festhalten; in CI und auf dem Server `npm ci` mit mitgeliefertem Lockfile verwenden. Einen Betriebsweg festlegen – z. B. das Docker-Image in CI bauen (mehrstufig, mit `.dockerignore`, nur Produktionsabhängigkeiten) und genau dieses Artefakt ausliefern.

AUSZUG

```
rsync ... -r dist package.json deploy@portal.muster-ag.example:/srv/portal
ssh ... "cd /srv/portal && npm install --production && pm2 restart portal"
```

4.6 Dependencies

■□□□ Ungenügend

Betroffen sind unter anderem Express, jsonwebtoken, Multer, axios, lodash und nodemailer. Ohne Anmeldung erreichbar sind vor allem ein Absturz über den JSON-Parser (body-parser) und ein Lastangriff über den Query-Parser von Express; die übrigen Advisories verstärken bestehende Lücken. Lizenzen sind unkritisch, das Lockfile ist vollständig, es gibt keine fremden Paketquellen. Die Einzelbefunde sind als SEC-04, SEC-19 und SEC-23 bewertet.

KAPITEL 5

Automatisierte Prüfung

Werkzeuge finden zuverlässig, was sich nach Mustern erkennen lässt. Relevante Funde wurden am Code verifiziert, Fehlalarme aussortiert. Die Rohergebnisse liegen bei.

WERKZEUG	PRÜFUNG	ROH	RELEVANT	BEFUNDE
gitleaks	Secrets in der gesamten Git-Historie	1	1	SEC-21
osv-scanner	Abhängigkeiten gegen CVE/OSV, Lizenzen	83	3	SEC-04, SEC-19, SEC-23
semgrep	SAST, Regelsätze OWASP Top 10 und Security-Audit	6	4	SEC-08, SEC-24, SEC-26
trivy	Konfiguration: Dockerfiles, IaC	2	2	SEC-24
lizard	Zyklomatische Komplexität, Funktionslänge, Parameter, Duplikate	48	0	–

AKTUALITÄT VON FRAMEWORK UND LAUFZEIT

KOMPONENTE	IM EINSATZ	AKTUELL	EINSCHÄTZUNG
Node.js (Laufzeit)	16	26 LTS	seit 2023 ohne Sicherheitsupdates

KOMPONENTE	IM EINSATZ	AKTUELL	EINSCHÄTZUNG
axios	0.21.1	1.21.0	1 Hauptversion zurück
better-sqlite3	7.6.2	13.0.3	6 Hauptversionen zurück
express	4.17.1	5.3.0	1 Hauptversion zurück
jsonwebtoken	8.5.1	9.0.3	1 Hauptversion zurück
multer	1.4.2	2.5.0	1 Hauptversion zurück
nodemailer	6.4.11	10.3.1	4 Hauptversionen zurück

HERKUNFT UND PINNING DER DEPENDENCIES

- Lockfile vollständig, keine fremden Paketquellen, keine eigene .npmrc
- Install-Skripte in drei Paketen (better-sqlite3, fsevents, nodemailer) – üblich, aber auf dem Server ohne Lockfile ausgeführt
- GitHub Actions über veränderliche Tags eingebunden, Token-Rechte nicht begrenzt

LIZENZEN

Übersicht ohne rechtliche Bewertung.

MIT	389	BSD-2-Clause	1
ISC	32	CC-BY-4.0	1
BSD-3-Clause	18	CC0-1.0 OR MIT	1
Apache-2.0	7	MIT OR WTFPL	1
Apache-2.0 OR BSD-2-Clause OR MIT	1	UNKNOWN	5

KAPITEL 6

Umfang, Methodik und Grenzen

Statische Prüfung des Quellcodes einschließlich Git-Historie, ausschließlich lesend. Es wurde kein Code ausgeführt und auf keine laufende Umgebung zugegriffen.

GEGENSTAND	Kundenportal · muster-kundenportal (Git-Bundle)
COMMIT	`e0e3cde972c5` (main, 20.10.2026)
ZEITRAUM	1. November 2026
AUFWAND	1 Personentag

GEPRÜFTE BEREICHE

BEREICH	ABDECKUNG	EINSCHÄTZUNG
Authentifizierung und Sessions	● Vollständig	
Autorisierung und Mandantentrennung	● Vollständig	
Injection, Eingaben, externe Aufrufe	● Vollständig	
Geheimnisse und Kryptografie	● Vollständig	inklusive gesamter Git-Historie
Protokollierung und Missbrauchsschutz	● Vollständig	
Konfiguration, Container und CI/CD	● Vollständig	Server und nginx nicht im Repository
Code-Qualität, Komplexität, Performance	● Vollständig	statisch, ohne Laufzeitmessung

SO ENTSTEHT DIE GESAMTBEWERTUNG

Jeder Bereich wird auf vier Stufen bewertet: Sehr gut (4), Gut (3), Handlungsbedarf (2), Ungenügend (1). Die Gesamtbewertung ist der gewichtete Durchschnitt; Sicherheit zählt 40 %, die übrigen Bereiche teilen sich 60 %. Ab 3,25 Punkten gilt Sehr gut, ab 2,5 Gut, ab 1,75 Handlungsbedarf, darunter Ungenügend. Ist die Sicherheit ungenügend, ist es die Gesamtbewertung ebenfalls.

Die Sicherheit ergibt sich aus den Befunden: Ungenügend bei einem kritischen Befund, Handlungsbedarf ab zwei hohen oder mehr als zehn mittleren Befunden, Sehr gut ohne hohe und mit höchstens drei mittleren Befunden, sonst Gut. Die übrigen Bereiche bewertet das Review anhand der Befunde und der Qualitätsbewertung.

SKALA DER EINSTUFUNG

EINSTUFUNG	CVSS	BEDEUTUNG
KRITISCH	9,0–10,0	Ausnutzbar mit schwerem Schaden, sofort handeln
HOCH	7,0–8,9	Erhebliches Risiko, kurzfristig beheben
MITTEL	4,0–6,9	Risiko unter Bedingungen, planbar beheben
NIEDRIG	0,1–3,9	Geringes Risiko oder Härtung

GRENZEN DES AUDITS

Das Audit ist eine zeitlich begrenzte, statische Stichprobe. Es ersetzt keinen Penetrationstest und ist kein Nachweis dafür, dass keine weiteren Schwachstellen bestehen.

Bewertet wurde der Stand des angegebenen Commits. Konfiguration und Daten der Laufzeitumgebung waren nicht Gegenstand.

NICHT IM UMFANG

- Laufende Umgebung, Server, nginx-Konfiguration und Datenbankinhalte
- Einstellungen außerhalb des Repositorys (Branch-Schutz, Secrets in GitHub)
- Laufzeitmessungen – Performance ist statisch beurteilt

Für Rückfragen stehe ich gern zur Verfügung.

David Kreutzer
Geschäftsführer, Viewconic GmbH

Anhang

DATEI	INHALT
`Rohdaten_Beispiel.zip`	Rohergebnisse aller Werkzeuge (JSON, CSV) und Übersicht
`Befunde_Beispiel.zip`	Alle Befunde als JSON-Dateien