Pflichtenheft – Company & Location Service

Service-Name:	company-service
Version:	1.2
Stand:	17.02.2026
Geltungsbereich:	Modul/Service zur Verwaltung von Companies (Mandanten) und deren Locations (Standorte) inkl. Hauptstandort-Regel, Standortstatus (OPEN/CLOSED), Audit, Eventing (Outbox – empfohlen) sowie Security-Integration via JWT (Bearer Token) gem. IAM/Auth-Konvention. Wichtig: In V1.2 wird kein Soft-Delete verwendet; Löschungen erfolgen als Hard-Delete über einen orchestrierten Löschprozess (Deletion Workflow) zur Bereinigung von Folgedaten in anderen Services.

1. 0. Änderungshistorie (Auszug)

V1.2

Security: Umstellung auf JWT (Bearer Token) – Rechte/Scopes im Access Token (Pattern A).

Auth-Service signiert JWT selbst; Resource Services validieren Signatur via Issuer-URI/JWKS (kid/Rotation).

Access Token Lifetime: 10 Minuten (kurzlebig); Refresh Token separat (nicht Bestandteil dieses Services).

Mandantentrennung: Tenant-Kontext ausschließlich aus JWT Claim tenant_id (Quelle der Wahrheit).

Bootstrap/Registrierung: Auth-Service darf POST /api/v1/companies ausführen (ohne tenant_id), wobei Company + erste Location atomar angelegt werden (erste Location wird Hauptlocation).

Löschung: Hard-Delete only – orchestrierter Löschprozess (Deletion Workflow) mit technischem Tombstone und Events zur Bereinigung in anderen Services (template-service, person-service, ...).

Idempotenz: Bootstrap-Create und Delete sind idempotent (Idempotency-Key).

2. 1. Einleitung

1.1 Ziel

Ziel ist die zentrale Verwaltung von Company-Stammdaten (Mandant) und Locations (Standorte) innerhalb eines Multi-Mandanten-Systems. Jede Company besitzt mindestens eine Location und genau eine Hauptlocation.

Der Service stellt insbesondere sicher:

beliebig viele Companies (Mandanten) im System

pro Company mindestens 1 Location

pro Company genau 1 Hauptlocation
Hauptlocation kann jederzeit gewechselt werden
Locations können geschlossen werden (CLOSED), aber es muss mindestens eine OPEN Location verbleiben und die Hauptlocation muss immer OPEN sein
Audit-Felder (wer/wann) bei Erstellung und Änderung
Integration via Events (Outbox empfohlen) für lose Kopplung mit anderen Services
Security via JWT/IAM: Mandantentrennung (tenant_id) + Scope-basierte Autorisierung
Hard-Delete über Löschworkflow zur konsistenten Datenbereinigung in abhängigen Services

1.2 Abgrenzung / Nicht-Ziele (MVP)

Nicht Bestandteil des MVP:

Authentifizierung (Login, MFA, Refresh Token-Ausstellung) – erfolgt durch den Auth-Service.
Benutzerverwaltung/Registrierung – User-Service / Auth-Service.
Verwaltung von Kommunikationsdaten (E-Mail, Telefon, Adresse) – Communication-Service.
Vollständiges revisionssicheres Audit-Log (append-only) – optional später (Events liefern Basis).
Volltext-/Geo-Suche über externe Suchsysteme (Elastic/OpenSearch) – optional später.

3. 2. Rahmenbedingungen / Technologie

2.1 Technologie-Stack (MUSS)

Backend: Java 21+ und Spring Boot 4.x
Datenbank: MariaDB (MySQL kompatibel)
API: REST/JSON
IDs: UUID/ULID (String)
Security: OAuth2 Resource Server (JWT) für Bearer Tokens (issuer-uri/jwks-uri konfigurierbar)
OpenAPI/Swagger Dokumentation: vollständig (Definition of Done).

2.2 Zeit- und Datumsstandard (MUSS)

Alle Zeitpunkte in Responses/Requests als UTC (ISO-8601), z. B. 2026-02-12T12:00:00Z.
Serverseitige Audit-Zeitpunkte werden in UTC gesetzt.

4. 3. Fachliche Anforderungen

3.1 Mandantenmodell / Zuordnung (MUSS)

Jeder Company- und Location-Datensatz ist eindeutig einem Mandanten zugeordnet:

companyId ist die Mandanten-ID.
Jede Location gehört genau zu einer Company: location.companyId.
Alle DB-Zugriffe sind mandantenscharf; Cross-Tenant-Zugriff ist ausgeschlossen.

3.2 IDs (UUID/ULID) & globale Eindeutigkeit (MUSS)

companyId und locationId sind UUID/ULID Strings.
locationId ist global eindeutig, damit die Route /api/v1/location/{locationId} ohne companyId möglich ist.

3.3 Company-Felder (MUSS/SOLL)

MUSS (MVP):

companyId
name, optional displayName
mainLocationId
optional timezone, locale, logoFileRef
Audit: createdAt/By, modifiedAt/By
version (Optimistic Locking)

SOLL (typisch):

nameNormalized (trim + lowercase) für Suche.

3.4 Location-Felder (MUSS/SOLL)

MUSS (MVP):

locationId
companyId
name
status (OPEN/CLOSED)
Close-Meta: closedAt/By, optional closedReason
Audit: createdAt/By, modifiedAt/By
version

SOLL (typisch):

locationCode (pro Company optional eindeutig).
timezone (Fallback auf Company timezone).

3.5 Hauptlocation-Regel (MUSS)

Pro Company existiert genau eine Hauptlocation.
Hauptlocation wird über company.mainLocationId definiert.
Hauptlocation MUSS existieren und OPEN sein.

3.6 Mindestanzahl aktiver Locations (MUSS)

Pro Company muss jederzeit mindestens eine OPEN Location existieren.

Das Schließen der letzten OPEN Location ist nicht erlaubt (409).

3.7 Standort schließen / wieder öffnen (MUSS)

Locations können geschlossen werden (status=CLOSED).

Eine Location darf nicht geschlossen werden, wenn sie aktuelle Hauptlocation ist.

Ein geschlossenes Objekt kann wieder geöffnet werden (status=OPEN).

3.8 Hard-Delete & Löschworkflow (MUSS)

In V1.2 gibt es kein Soft-Delete. Löschungen erfolgen als Hard-Delete. Da andere Services (z. B. template-service, person-service) Folgedaten zur companyId halten können, wird die Löschung als orchestrierter Löschworkflow umgesetzt.

MUSS:

DELETE einer Company startet einen Deletion Workflow (asynchron) und ist idempotent (Idempotency-Key).

Während Deletion in progress gilt die Company fachlich als nicht existent; normale GET/PUT/POST liefern 404.

company-service erzeugt einen technischen Tombstone (DeletionTombstone) zur Prozesssteuerung (kein Soft-Delete der Company).

company-service publiziert CompanyDeletionRequested (Outbox empfohlen).

Abhängige Services löschen ihre Daten zur Company und bestätigen per CompanyDeletionCompleted (idempotent).

Nach Eingang aller erforderlichen Bestätigungen (oder definiertem Timeout/Fehlerpfad) löscht company-service Company + Locations final und publiziert CompanyDeleted.

SOLL:

Timeout/Fehlerbehandlung: Löschworkflow kann in FAILED übergehen; Monitoring/Alarmierung erforderlich.

Optionaler Admin-Status-Endpunkt zur Abfrage des Löschfortschritts (nur mit Admin-Scope).

3.9 Firmenlogo (SOLL)

Company kann ein Logo haben, gespeichert als Referenz logoFileRef (kein Base64 in DB).

Upload/Storage erfolgt außerhalb dieses Services (z. B. File-Service/Object Storage).

3.10 Kommunikation (Abgrenzung, MUSS)

Kommunikationsdaten werden nicht im company-service gespeichert.

Integration erfolgt über Communication-Service via ownerType + ownerId.

3.11 Audit & Versionierung (MUSS)

MUSS:

Audit: createdAt/By, modifiedAt/By.

Close Audit: closedAt/By bei Location Close.

Optimistic Locking über version zur Vermeidung von Überschreibkonflikten, insbesondere bei setMainLocation, close/reopen.

3.12 Domain-Events / Outbox (SOLL, empfohlen)

Outbox Pattern: Domain-Event wird in derselben DB-Transaktion geschrieben (mind. für kritische Events).

Events (minimal):

CompanyCreated, CompanyUpdated, CompanyMainLocationChanged, CompanyDeletionRequested, CompanyDeleted, CompanyDeletionFailed

LocationCreated, LocationUpdated, LocationClosed, LocationReopened

5. 4. Nicht-funktionale Anforderungen

4.1 Sicherheit / Mandantentrennung (JWT/IAM) (MUSS)

Der company-service ist ein OAuth2 Resource Server. Er validiert JWT Access Tokens, die vom Auth-Service ausgestellt und signiert werden. Die Rechte/Scopes sind im Token enthalten (Pattern A).

MUSS:

Signaturprüfung via issuer-uri oder jwks-uri (kid/Rotation). Tokens ohne verifizierbare Signatur werden abgewiesen (401).

Audience-Check: Claim aud muss company-service enthalten (strict).

Mandantenkontext ausschließlich aus tenant_id (Quelle der Wahrheit).

Scopes über Claim scope (String) oder scp (Array).

Pflicht-Claims (MUSS):

iss, sub, aud, exp, iat, jti

tenant_id (für alle tenant-scharfen Endpoints). Ausnahme: Bootstrap-Create (siehe 4.1.4).

Scopes (MUSS, minimal):

company:read – Lesen/Listen im eigenen Tenant (Company/Locations).

company:write – Ändern im eigenen Tenant (Company/Locations, inkl. reopen).

company:admin – administrative Aktionen (close, mainLocation, delete).

company:create – Bootstrap: Company anlegen (Registrierung/Provisionierung).

4.1.1 Spezialfall: Location-Routen ohne companyId (MUSS)

Bei /api/v1/location/{locationId} MUSS serverseitig geprüft werden:

location.companyId == tenant_id aus JWT.

zusätzlich Scope-Prüfung im Company-Kontext.

4.1.2 Token-Laufzeiten (MUSS/SOLL)

Access Token Lifetime: 10 Minuten (gegeben).

Resource Services akzeptieren abgelaufene Tokens nicht (401).

SOLL: Clock-Skew (kleines Zeitfenster) konfigurierbar.

KANN: Sofort-Sperre (Kill-Switch) via tokensValidAfter (z. B. bei Incident) als spätere Erweiterung.

4.1.3 Tenant-Isolation (MUSS)

Der Service akzeptiert keine tenantId/companyId aus Body/Query als Steuerungsfeld.

Cross-Tenant Zugriff ist ausgeschlossen (403).

4.1.4 Bootstrap: Company-Anlage bei Registrierung (MUSS)

Beim Start/bei Registrierung im Auth-Service darf eine Company angelegt werden. Dazu gilt:

Endpoint: POST /api/v1/companies ist ohne tenant_id zulässig (Bootstrap-Flow).
Der Call MUSS mit Scope company:create erfolgen.
sub sollte den Service-Principal des Auth-Service enthalten (z. B. auth-service).
Company + erste Location werden atomar angelegt; die erste Location wird automatisch Hauptlocation.
Idempotenz ist verpflichtend (Idempotency-Key), um Retries in der Registrierung abzufangen.

4.2 Logging/Tracing (SOLL)

Jede Anfrage loggt correlationId/requestId, tenant_id (falls vorhanden), Endpoint, Dauer, Statuscode.
Keine sensiblen Daten im Klartext loggen (z. B. Tokens).

4.3 Caching (SOLL)

Siehe Abschnitt 8 (Caching).

4.4 Zuverlässigkeit des Löschworkflows (MUSS/SOLL)

MUSS:

Event-Delivery wird als at-least-once angenommen; alle Consumer müssen idempotent sein.
Outbox Pattern wird für CompanyDeletionRequested und CompanyDeleted empfohlen.
DELETE ist idempotent (Idempotency-Key) und liefert wiederholbar denselben Prozesszustand zurück.

SOLL:

Retry + Dead-Letter für Löschbestätigungen/Events; Monitoring/Alarmierung bei FAILED oder Timeout.

6. 5. Datenmodell (logisch)

5.1 Entity: Company

Kernfelder:

companyId (UUID/ULID, String)
name, optional displayName
mainLocationId (UUID/ULID, String)
optional timezone, locale, logoFileRef
Audit: createdAt/By, modifiedAt/By
version

Indizes (MUSS/SOLL):

MUSS: PK(companyId)
SOLL: (nameNormalized) für Suche

5.2 Entity: Location

Kernfelder:

locationId (UUID/ULID, String)
companyId (FK logisch)
name, optional locationCode, optional timezone
status (OPEN/CLOSED)
Close: closedAt/By, optional closedReason
Audit: createdAt/By, modifiedAt/By
version

Indizes (MUSS/SOLL):

MUSS: PK(locationId)
SOLL: (companyId, status) (z. B. für „mindestens eine OPEN“)
SOLL: (companyId, nameNormalized)
SOLL: Unique (companyId, locationCode) falls locationCode genutzt

5.3 Technisch: DeletionTombstone (MUSS)

Der DeletionTombstone ist ein technisches Objekt zur Steuerung des Löschworkflows. Er ersetzt kein Soft-Delete der Company, sondern markiert den Prozesszustand, bis die Löschung finalisiert ist.

Kernfelder:

deletionId (UUID, String)

companyId (UUID/ULID, String)

state (IN_PROGRESS | FAILED | COMPLETED)

requestedAtUtc, requestedBySub

optional correlationId

optional idempotencyKey

SOLL: Tracking der erforderlichen/erhaltenen Acks (z. B. serviceName-Liste).

5.4 Optional: OutboxEvent (SOLL)

eventId, eventType, occurredAtUtc, companyId, optional locationId, actorSubjectId, payloadJson, status, retryCount

7. 6. REST-API Spezifikation (inkl. Zweck)

6.1 Company

POST /api/v1/companies

Zweck: Company anlegen (inkl. initialer Location als Hauptlocation). Bootstrap erlaubt (ohne tenant_id). Scope: company:create. Idempotent via Idempotency-Key.

GET /api/v1/companies/{companyId}

Zweck: Company laden. Scope: company:read. Regel: companyId == tenant_id. Während Deletion in progress: 404.

PUT /api/v1/companies/{companyId}

Zweck: Company ändern. Scope: company:write. Regel: companyId == tenant_id.

PUT /api/v1/companies/{companyId}/main-location

Zweck: Hauptlocation wechseln. Scope: company:admin. Regeln: Ziel-Location muss zur Company gehören und OPEN sein.

PUT /api/v1/companies/{companyId}/logo

Zweck: Logo-Referenz setzen. Scope: company:write.

DELETE /api/v1/companies/{companyId}/logo

Zweck: Logo entfernen. Scope: company:write.

DELETE /api/v1/companies/{companyId}

Zweck: Hard-Delete einer Company über Deletion Workflow (asynchron). Scope: company:admin. Regel: companyId == tenant_id. Antwort: 202 Accepted + Deletion-Status. Während Deletion in progress gilt Company fachlich als nicht existent (404). Idempotent via Idempotency-Key.

GET /api/v1/companies/{companyId}/deletion-status (SOLL)

Zweck: Admin-Statusabfrage Löschworkflow. Scope: company:admin. Nur im eigenen Tenant. Liefert state/progress/startedAt.

GET /api/v1/companies (SOLL)

Zweck: Companies listen. Scope: company:read. Paging/Sort.

6.2 Locations (company-bezogen, empfohlen für UI)

GET /api/v1/companies/{companyId}/locations (SOLL)

Zweck: Locations einer Company listen. Scope: company:read. Query (SOLL): status, nameContains, Paging/Sort.

POST /api/v1/companies/{companyId}/locations (SOLL)

Zweck: Location anlegen (default OPEN). Scope: company:write.

6.3 Location (ID-basiert)

GET /api/v1/location/{locationId}

Zweck: Location laden. Scope: company:read. MUSS: serverseitiger Tenant-Check (location.companyId == tenant_id).

PUT /api/v1/location/{locationId}

Zweck: Location ändern. Scope: company:write. MUSS: Tenant-Check.

POST /api/v1/location/{locationId}/close

Zweck: Location schließen (status=CLOSED). Scope: company:admin. Regeln: nicht Hauptlocation; nicht letzte OPEN Location; Tenant-Check.

POST /api/v1/location/{locationId}/reopen

Zweck: Location wieder öffnen (status=OPEN). Scope: company:write. Tenant-Check.

DELETE /api/v1/location/{locationId} (KANN)

Zweck: Hard-Delete einer Location. Scope: company:admin. Regeln: nicht Hauptlocation; nicht letzte OPEN Location; Tenant-Check. Hinweis: Falls andere Services locationId referenzieren, ist ein LocationDeleted-Event vorzusehen (KANN).

8. 7. DTOs (Beispiele)

7.1 Create Company (mit initialer Location) – Request

```
{
  "name": "InnoLogic GmbH",
  "displayName": "InnoLogic",
  "timezone": "Europe/Berlin",
  "locale": "de-DE",
  "logoFileRef": "file_abc123",
  "initialLocation": {
    "name": "Bremen HQ",
    "locationCode": "HB-01",
    "timezone": "Europe/Berlin"
  }
}
```

7.2 Delete Company – Response (202 Accepted)

```
{
  "companyId": "01J3Z4Z8Q9F1K2M3N4P5R6S7T8",
  "deletionId": "a5f7d1b0-4c7f-4cc7-9e1f-0c2c2b7d9c4a",
  "state": "IN_PROGRESS",
  "startedAtUtc": "2026-02-12T12:34:00Z"
}
```

7.3 JWT Claims – Beispiel (Tenant Call)

```
{
  "iss": "https://auth.example.local",
  "sub": "user_123",
  "aud": ["company-service"],
  "exp": 1760000000,
  "iat": 1759999400,
  "jti": "01J...TOKEN...",
  "tenant_id": "01J3Z4Z8Q9F1K2M3N4P5R6S7T8",
  "scope": "company:read company:write"
}
```

7.4 JWT Claims – Beispiel (Bootstrap Create Company)

```
{
  "iss": "https://auth.example.local",
  "sub": "auth-service",
  "aud": ["company-service"],
  "exp": 1760000000,
  "iat": 1759999400,
  "jti": "01J...TOKEN...",
  "scope": "company:create"
}
```

9. 8. Caching (SOLL)

Ziele:

Reduzierung DB-Last bei häufigen Reads (Company, Locations).

Kurze TTLs, klare Invalidation-Strategie.

TTL (Beispiel):

Company: 60s
Locations: 60s

Invalidation (MUSS, falls Caching aktiv):

Nach PUT Company: Company-Cache invalidieren.
Nach Location-Änderungen: Locations-Cache invalidieren.
Nach Set Main Location: Company-Cache invalidieren.
Bei Delete Company: alle Tenant-bezogenen Caches invalidieren.

10.9. Fehlercodes (MUSS)

400 Bad Request – Validierungsfehler (DTO).

401 Unauthorized – fehlendes/ungültiges JWT (Signatur, exp, aud, iss).

403 Forbidden – fehlender Scope oder Cross-Tenant Zugriff (tenant_id Mismatch).

404 Not Found – Objekt existiert nicht (oder Deletion in progress).

409 Conflict – fachliche Regel verletzt (z. B. letzte OPEN Location).

202 Accepted – Delete Workflow gestartet (asynchron).

500 Internal Server Error – unerwarteter Fehler.

11.10. Pagination & Sort (SOLL)

Standard für Listen-Endpunkte:

page (default 0), size (default 50, max z. B. 200)

sort (z. B. name,createdAt; default: name)

12.11. Testing (MUSS/SOLL)

Unit Tests: Services (Regeln: Hauptlocation, letzte OPEN Location).

Integration Tests: REST + DB (Testcontainers MariaDB empfohlen).

Security Tests: JWT-Validierung (aud/iss/exp, Signatur/JWKS), Scope-Prüfung, tenant_id Isolation.

Deletion Workflow Tests: Idempotenz (Create/Delete), Eventing (Requested/Completed/Deleted), Timeout/Failed-Pfad.

13.12. MUSS/SOLL/KANN Übersicht

Bereich	MUSS	SOLL	KANN
Security	JWT Resource Server, JWKS Signaturprüfung, aud strict, tenant_id Isolation, Scopes company:*	Clock skew; Monitoring	Kill-Switch tokensValidAfter
Löschung	Hard-Delete only via Deletion Workflow, Tombstone, Events, Idempotency-Key	Timeout/Failed-Handling; Deletion-Status	LocationDeleted-Event oder Workflow
Fachliche Regeln	Hauptlocation, min. 1 OPEN Location, Close/Reopen	Suche/Filter	-
Eventing	Events bei Changes (mind. kritisch)	Outbox Pattern	-

14.13. Besonderheiten / Sonderfälle (MUSS)

Location-Routen ohne companyId: Tenant-Check immer über DB (location.companyId == tenant_id).

In-flight Requests während Delete: Mutationen müssen DeletionTombstone prüfen und Requests ablehnen (404/409).

Event-Duplikate: Consumer müssen idempotent sein (at-least-once).

Abhängige Services: Bei CompanyDeletionRequested müssen Folgedaten gelöscht werden und CompanyDeletionCompleted gesendet werden.

Timeout/Fehler: Bei ausbleibenden Acks muss der Workflow in FAILED gehen (Monitoring/Alarm).

Idempotenz: Bootstrap-Create und Delete müssen Retries sauber abfangen (Idempotency-Key).