```
module: 30
title: i18n / Mehrsprachigkeit
phase: 1
status: detail-spec-draft
last_updated: 2026-05-08
session_source: brainstorming-session-2026-05-07-2206.md
related_lps: [LP-9, LP-10, LP-12]
spec_style: requirements-pur
```

Modul 30 — i18n / Mehrsprachigkeit

> ***Detail-Spec gemäß LP-9 + LP-12.*** *Phase 1 nur DE-Locale befüllt, aber die Library wird extrem sauber gebaut, damit Sprache in Phase 2 wirklich kostenlos erweiterbar ist (DeepL-Pipeline für DE→EN/TR/PL).* ***Architektur-Vorschrift AV #4: alle Strings extern, kein hartkodierter Text irgendwo.***

Themenblöcke

```
A. Architektur-Grundsätze (was muss / was darf nicht)
B. Datei-Struktur & Locale-Format
C. Datenmodell (User-Spracheinstellung, Tenant-Default-Sprache)
D. Sprachauflösung (welche Sprache wird wann gewählt)
E. Inhaltliche Trennung: UI-Strings vs. Tenant-Inhalte
F. Outbound-Communication (Email/WhatsApp/Push)
G. Settings-Routen (User + Tenant)
H. Permissions
I. Phase 2 / spätere Features
```

Block A — Architektur-Grundsätze

A.1 — Pflicht-Vorschriften

1. **Kein hartkodierter User-facing Text** im gesamten Codebase. Jeder String, der in der UI sichtbar ist, jede Email-Vorlage, jeder Klartext-Hinweis, jede Validierungs-Fehlermeldung läuft über die i18n-Library.
2. **Keine String-Konkatenation für Sätze** (verhindert Übersetzbarkeit). Alle Sätze als komplette Locale-Keys mit Platzhaltern.
3. **Pluralisierung über ICU-Format** (nicht über if/else im Code), damit andere Sprachen mit anderen Plural-Regeln (z.B. PL hat 3 Plural-Formen) problemlos funktionieren.
4. **Datums-, Zeit- und Zahlenformate** über `Intl`-API mit User-Locale (DE-Format Default), nicht hartkodiert.
5. **Lint-Regel im Build:** harte Strings in JSX/TSX-Komponenten und Server-Code-Pfaden, die nicht in einer Allowlist (technische Logs, IDs, etc.) stehen, blockieren den Build.
6. **Keine direkten Übersetzungen-on-the-fly** in Phase 1 — alle Texte sind statisch in Locale-Files definiert (LP-10: Klartext-Diagnostik = fest formuliert, nicht KI-on-the-fly).

A.2 — Was NICHT i18n-pflichtig ist (bewusst exkludiert)

- **Technische Identifier** (Permission-IDs, Pipeline-Stage-Slugs, Activity-Type-Enums, etc.)
- **Datenbankinhalte** (Custom-Felder-Labels, Tag-Namen, Pipeline-Stage-Display-Names, Form-Texte) — siehe Block E.2

- **System-Logs / Sentry-Fehler** (English Default, technisch nicht user-facing)
- **API-Response-Body-Felder** (technische Schlüsselnamen, nicht user-facing)
- **Email-Adressen, URLs, Domains**

Block B — Datei-Struktur & Locale-Format

B.1 — Datei-Struktur

```
app/i18n/
├── de.json          (Phase 1 – voll befüllt)
├── en.json          (Phase 2 – DeepL-generiert, manuell reviewt)
├── tr.json          (Phase 2)
├── pl.json          (Phase 2)
└── index.ts         (Loader-Logic, Type-Generation)
```

Standard-Format: flache JSON-Strukturen mit Punkt-Notation als Pfad oder verschachtelte Objekte (Implementierungs-Detail). Beispiel-Konvention:

```
{
  "common.save": "Speichern",
  "common.cancel": "Abbrechen",
  "tasks.title": "Aufgaben",
  "tasks.empty_state": "Keine offenen Aufgaben – schöne Sache.",
  "tasks.deadline_overdue_singular": "{count} Aufgabe überfällig",
  "tasks.deadline_overdue_plural": "{count} Aufgaben überfällig"
}
```

B.2 — Key-Konventionen

- **Namespacing pro Modul:** `tasks.*`, `crm.contacts.*`, `auth.login.*`, etc.
- **Key-Sprache: Englisch in lowercase + underscore/dot**, damit Code-Lesbarkeit in jeder UI-Sprache gleich
- **Wiederverwendbare Bausteine:** `common.save`, `common.cancel`, `common.delete`, `common.archive`, etc.
- **Klartext-Fehlermeldungen** (LP-10): eigener Namespace `errors.*` mit Lösungs-Hinweis-Sub-Keys (`errors.payment_failed.headline`, `errors.payment_failed.body`, `errors.payment_failed.action_label`)
- **Lange Texte (Email-Templates, mehrzeilige Klartext-Hinweise):** als zusammenhängende Strings, kein Composing aus mehreren Keys

B.3 — Locale-Validierung im Build

- **Vollständigkeits-Check:** alle Keys, die in `de.json` existieren, müssen in jeder anderen befüllten Locale auch existieren (Build-Fail bei Lücken in `en.json` / `tr.json` / `pl.json` ab Phase 2)
- **Keine Waisen-Keys:** Keys, die in keinem Code-Pfad referenziert werden, werden im Build markiert (Warning)
- **Type-Generation:** TypeScript-Types werden aus `de.json` (Master-Locale) generiert; tippt der Code einen falschen Key, gibt es Compile-Error

B.4 — Master-Locale = DE

- Phase 1: nur `de.json` befüllt
- Andere Locale-Files können in Phase 1 leer/teilbefüllt existieren — Fallback auf DE bei fehlendem Key (LP-12: Sprache kostenlos erweiterbar)
- Fallback-Verhalten: bei fehlendem Key in gewünschter Sprache → automatisch DE-Wert; im Dev-Mode mit Debug-Marker, in Prod silent

Block C — Datenmodell

C.1 — User-Spracheinstellung

Bestehendes `User`-Feld (siehe Modul 27):
- `language` (Enum: `de | en | tr | pl`, Default `de` in Phase 1, in Phase 1 effektiv nur `de` wählbar)
- `timezone` (IANA-Zeitzonen-String, Default `Europe/Berlin`)
- `date_format_preference` (Enum: `auto | iso | de | us`, Default `auto` — folgt Locale-Standard)

Setzbar in: `/{tenant}/settings/profile/`

C.2 — Tenant-Default-Sprache

Neues `Tenant`-Feld:
- `default_language` (Enum: `de | en | tr | pl`, Default `de`)
- Anwendung: User ohne explizite Spracheinstellung erbt Tenant-Default; Outbound-Communications an Empfänger ohne hinterlegte Spracheinstellung (z.B. Endkunden-Emails an neue Leads) nutzen Tenant-Default

Setzbar in: `/{tenant}/admin/general/` (erfordert `tenant.admin`)

C.3 — Contact-Spracheinstellung (siehe Modul 1)

Bestehendes `Contact`-Feld:
- `language` (Enum: `de | en`, Default `de` in Phase 1) — für Outbound-Kommunikation an den Kontakt

In Phase 2 erweitert auf `de | en | tr | pl`.

C.4 — Keine separate Locale-DB-Tabelle

Locale-Inhalte sind **statisch in JSON-Files**, nicht in DB. Begründung:
- Performance (kein DB-Roundtrip pro String)
- Versionierung in Git
- Build-Validation möglich
- Ausschließlich Tenant-Inhalte (siehe Block E.2) sind in DB; die werden nicht als Locale-Strings behandelt

Block D — Sprachauflösung

D.1 — Auflösungs-Reihenfolge (UI)

Beim Rendern einer User-Session:

1. Wenn User eingeloggt + `User.language` gesetzt → diese Sprache
2. Sonst: `Tenant.default_language` (falls in Tenant-Kontext)
3. Sonst: Browser-Header `Accept-Language` (für nicht-eingeloggte Routen wie `/login`, `/signup`, `/help`, Booking-Pages, Forms)
4. Sonst: `de` (Plattform-Fallback)

D.2 — Auflösungs-Reihenfolge (Outbound an Empfänger)

Beim Versand einer Email/WhatsApp/Push-Notification an einen Empfänger:

1. Wenn Empfänger ein User ist + `User.language` gesetzt → diese
2. Wenn Empfänger ein Contact ist + `Contact.language` gesetzt → diese
3. Sonst: `Tenant.default_language`

4. Sonst: `de`

D.3 — Sprache pro Request festlegen

- Server-Loader setzt Sprache am Anfang jedes Requests basierend auf D.1
- Components erhalten Sprache über Context/Hook (z.B. `useTranslation` / `t(key)`)
- SSR rendert sofort korrekt ohne Flash-of-DE

D.4 — Sprache wechseln

- User wechselt Sprache in `/settings/profile/` → sofort wirksam, Reload-Free
- Cookie/Session speichert Sprache (für Konsistenz nach Logout/Re-Login: Wert kommt nach Login direkt aus `User.language` aus DB)
- Non-Logged-In: Sprache via Header-Switcher in Public-Pages (z.B. Help-Center, Booking-Page) änderbar; Auswahl in Cookie für Session

---

Block E — Inhaltliche Trennung

E.1 — UI-Strings (i18n-pflichtig)

Alles, was im Code als Text steht und in der UI gerendert wird:
- Button-Labels, Form-Labels, Headlines, Body-Texte
- Validierungs-Fehlermeldungen
- Klartext-Hinweise (LP-10)
- Email-Template-Texte (Plattform-eigene Templates: Welcome, Passwort-Reset, Einladung, Notifications, etc.)
- Push-Notification-Texte
- Help-Center-Artikel-Titel und Default-Inhalt (siehe Modul 38, Phase 1 nur DE)

E.2 — Tenant-Inhalte (NICHT i18n-pflichtig)

Daten, die der Tenant selbst eingibt — werden in der **Tenant-Sprache** gespeichert und so angezeigt, wie eingegeben:
- Custom-Felder-Labels und -Optionen (Modul 1, Block F)
- Tag-Namen (Modul 1, Block H)
- Pipeline-Stage-Namen (Modul 3, Modul 4)
- Form-Builder-Inhalte: Form-Titel, Feld-Labels, Hilfetexte (Modul 5)
- Booking-Page-Beschreibungen (Modul 6)
- Deal-/Projekt-Templates
- Notizen, Beschreibungen
- Email-Marketing-Kampagnen-Inhalte (Modul 10)
- Automation-Regel-Namen
- HR-Hub-Ankündigungen (Modul 14)

Begründung: Diese Inhalte sind tenant-eigene Geschäftsdaten. Multi-Sprach-Support für Tenant-Inhalte (z.B. Pipeline-Stage in DE + EN) ist explizit Phase-2-Thema (siehe I.).

Tenant-Sprache als Dateneingabe-Sprache: Der Tenant gibt Custom-Felder/Tags/Pipeline-Stages in der Sprache ein, in der seine Mitarbeiter arbeiten. Wenn ein Tenant mehrsprachige Teams hat, ist das in Phase 1 ein offener Punkt — Workaround: Eingabe in DE oder EN, andere lesen mit.

E.3 — System-Daten (DE-fix)

Plattform-eigene Default-Daten, die beim Tenant-Anlegen geseedet werden (SHK-Template-Rollen, SHK-Custom-Felder, Default-Pipeline-Stages, Default-Tags) sind in Phase 1 **fix in DE**. Phase 2: Templates haben übersetzte Versionen pro Locale, beim Seeding wird die passende Version genommen je nach `Tenant.default_language`.

Block F — Outbound-Communication

F.1 — Email

- **Plattform-eigene Templates** (Welcome, Passwort-Reset, MA-Einladung, Notification-Digest, etc.): pro Locale-File ein Template-Set; bei Versand wird Empfänger-Sprache aufgelöst (D.2) und passendes Template gerendert
- **Tenant-eigene Email-Marketing-Kampagnen** (Modul 10): Tenant erstellt Inhalt selbst, in Tenant-Sprache; in Phase 1 keine Multi-Sprach-Variante pro Kampagne (Phase 2 möglich: Variante pro Locale + Empfänger-Auflösung)
- **Plattform-Footer in Email** (Impressum-/AGB-Links, „Powered by HiveGrid"): über Locale-Files

F.2 — WhatsApp (Phase 2 — Modul 11)

- Analog Email: Plattform-Templates über i18n-Library, Tenant-Templates eingegeben in Tenant-Sprache
- WhatsApp-Business-Templates müssen pro Sprache bei Meta einzeln approved werden — Spec-Detail in Modul 11

F.3 — Push (Modul 22)

- Plattform-Push-Texte: über Locale-Files
- Empfänger-Sprache via D.2 aufgelöst
- Title + Body separat übersetzt; Deep-Link bleibt sprach-unabhängig

F.4 — In-App-Notifications (Modul 22)

- Über Locale-Files; Sprache des **lesenden** Users (D.1), nicht des erzeugenden Users
- Beispiel: User A (DE) erstellt Task für User B (EN) → User B sieht Notification auf EN

Block G — Settings-Routen

G.1 — User-Profile-Settings (`/{tenant}/settings/profile/`)

Spracheinstellung:
- Auswahl-Komponente für Sprache (Phase 1: nur DE auswählbar — andere Sprachen in der UI sichtbar als „Coming soon", Phase 2 freigeschaltet)
- Auswahl-Komponente für Zeitzone
- Auswahl-Komponente für Datums-Format-Präferenz
- Sofortige Anwendung ohne Reload

Permission: keine eigene — jeder eingeloggte User kann sein eigenes Profil bearbeiten.

G.2 — Tenant-General-Settings (`/{tenant}/admin/general/`)

Tenant-Default-Sprache:
- Auswahl-Komponente für `Tenant.default_language`
- Klartext-Hinweis: *„Diese Sprache wird verwendet für: Outbound-Emails an Empfänger ohne eigene Spracheinstellung, Default für neue User, System-Texte für nicht-eingeloggte Besucher (Booking-Pages, Forms)."*

Permission: `tenant.admin`.

G.3 — Help-Center (Modul 38) und Public-Pages

- URL-Struktur enthält Sprachsegment für öffentliche Seiten: `/help/{lang}/{slug}` (Phase 1 effektiv nur `/help/de/...`)

- Booking-Pages und Forms erben Tenant-Default-Sprache; Endkunde sieht Sprach-Switcher (Phase 2 mehr Optionen)

Block H — Permissions

Permission-IDs

i18n hat **keine eigenen** Permissions. Anwendungen:
- User-Spracheinstellung in `/settings/profile/` → kein Permission-Gate (eigenes Profil)
- Tenant-Default-Sprache in `/admin/general/` → `tenant.admin`

Klartext-Fehler

- User wählt Sprache, die in Phase 1 noch nicht befüllt ist: Fallback auf DE silent in Prod; im Dev-Mode Hinweis *„Lokalisierung für [Sprache] ist noch nicht vollständig — Fallback auf Deutsch."*
- Tenant-Admin wechselt `default_language` zu nicht-befüllter Sprache: Klartext-Warnung *„Die Sprache [Sprache] ist noch nicht vollständig verfügbar. Bestehende User behalten ihre Sprache; neue System-Texte fallen auf Deutsch zurück."*

Block I — Klartext-Hinweise (zentral)

- Sprache erfolgreich gewechselt: *„Sprache geändert."*
- Fehlende Locale: siehe oben
- Browser-Header zeigt unbekannte Sprache: silent Fallback DE

Phase 2 / spätere Features

I.1 — DeepL-Pipeline (DE → EN/TR/PL)

- Build-Step: alle DE-Keys werden via DeepL-API automatisch übersetzt
- Generierte Locale-Files werden manuell reviewt (Glossary für Fachbegriffe SHK, Plattform-Begriffe wie „Tenant", „Workspace")
- Glossary in Repo gepflegt (CSV oder JSON)
- Re-Translation nur für geänderte Keys (Diff-basiert)
- DeepL-API-Quota: monatlich, in CI/CD getrackt

I.2 — Chat-Auto-Übersetzung (KI-Killer-Feature)

- Team-Chat (Modul 13): User schreibt in seiner Sprache, andere User sehen automatisch in ihrer Sprache übersetzt
- Beispiel: PL-Monteur schreibt Polnisch, Büro liest Deutsch
- Pro Nachricht: Original gespeichert, Übersetzung on-the-fly per LLM (LP-13: EU-Provider Mistral/Scaleway)
- Toggle pro User: „Nachrichten in meine Sprache übersetzen" (Default an, in mehrsprachigen Tenants)
- Kennzeichnung: Übersetzung als solche markiert mit Original-Anzeige auf Klick

I.3 — Multi-Sprach-Tenant-Inhalte

- Custom-Felder-Labels mit Übersetzungen pro Locale (Tenant-Admin pflegt mehrere Sprachversionen)
- Pipeline-Stage-Namen mit Übersetzungen

- Form-Builder-Forms mit Übersetzungs-Modus für Endkunden-Forms
- Email-Marketing-Kampagnen-Varianten pro Locale, Empfänger-Auflösung wie F.1

I.4 — Sprache pro Endkunden-Form

- Endkunde wählt Sprache auf öffentlicher Form-Page
- Form-Builder erlaubt Übersetzungen der Feld-Labels und Hilfetexte

I.5 — Help-Center mehrsprachig

- Plattform-Help-Center mit voll lokalisierten Artikeln (Modul 38)
- DeepL für Erst-Übersetzung, manueller Review

I.6 — Datenformate erweitert

- Lokale Telefon-Format-Anzeige (E.164 + lokales Display)
- Lokale Adress-Formate (US-Adressen vs. DACH-Adressen)
- Lokale Währungsanzeigen pro User-Präferenz

I.7 — Sprache für KI-Outputs (LLM-Gateway)

- LLM-Antworten werden in User-Sprache erzeugt (Prompt-Suffix mit Sprach-Anweisung)
- RAG-Suche kann sprachübergreifend matchen (Multi-Lingual-Embeddings)

Architektur-Vorschriften (Querschnitt)

- **AV #4 (LP-12) — Alle Strings extern:** kein hartkodierter Text, Lint-Regel im Build (Block A.1)
- **LP-9 — Spec-Tiefe:** Architektur-Vorschrift hier auf Modul-Ebene konkret beschrieben
- **LP-10 — Klartext-Diagnostik:** alle Fehler-Texte in `errors.*`-Namespace mit Headline + Body + Action-Label
- **LP-12 — Sprache kostenlos erweiterbar:** Library so sauber gebaut, dass Phase 2 nur Locale-Files + DeepL-Pipeline benötigt, kein Code-Refactor
- **LP-13 — EU-Stack:** DeepL ist EU-Anbieter, kompatibel; KI-Übersetzungen Phase 2 ausschließlich über EU-LLM-Gateway

---

Detail-Spec-Status: DRAFT

Erste vollständige Detail-Spec für Modul 30. **Vor Bau:** Verifizierung der Lint-Regel-Strenge mit Noah (Allowlist für technische Strings). Bei Implementierung dann konkrete i18n-Library-Wahl (z.B. `react-i18next` aus altem Codebase weiternutzen oder Alternative), Build-Pipeline für DeepL Phase 2 separat speccen.

Offene Detail-Punkte (nicht-blockierend):
- Konkrete i18n-Library-Wahl (Implementierungs-Detail)
- Lint-Regel-Allowlist-Spec
- DeepL-Glossary-Format (Phase 2)
- Konkretes Cookie-Schema für Non-Logged-In-Sprach-Persistierung

Abhängige Module:
- Alle UI-Module (jedes Modul liefert seine Locale-Keys)
- Modul 22 (Notifications) — Empfänger-Sprach-Auflösung
- Modul 10 (Email-Marketing) — Phase 2 Multi-Sprach-Kampagnen
- Modul 13 (Chat) — Phase 2 Auto-Übersetzung
- Modul 38 (Help-Center) — Phase 2 Multi-Sprach-Artikel