```
module: 32
title: Feature-Flags
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]
spec_style: requirements-pur
```

Modul 32 — Feature-Flags

> ***Detail-Spec gemäß LP-9.*** *Phase 1 minimal — reines Plattform-Admin-Werkzeug, um Features pro Tenant gezielt zu aktivieren/deaktivieren. Kein Tenant-User-UI in Phase 1. Eng verzahnt mit Modul 37 (Plattform-Admin) und Modul 27 Block F.2.*

Themenblöcke

```
A. Datenmodell (FeatureFlag, FeatureFlagFilter, TenantFeatureFlag)
B. Routes (admin.hivegrid.de/feature-flags)
C. Listen-View und Detail-View
D. Filter-Auswertung (zielgruppen-basiertes Aktivieren)
E. Code-Convention (tenantHasFlag)
F. Flag-Lifecycle (beta → rollout → GA → deprecated)
G. Audit-Logging
H. Permissions
I. Phase 2 Erweiterungen
```

Block A — Datenmodell

FeatureFlag

Globale Definition eines Flags. Lebt **nicht** im Tenant-Scope, kein RLS — gehört der Plattform.

- `id`, `created_at`, `updated_at`, `created_by_admin_user_id`
- `key` (technischer Name, snake_case, eindeutig global, z.B. `beta_new_dashboard`)
- `label` (kurzer Klartext-Name für Admin-UI)
- `description` (Markdown-Freitext: was tut das Flag, welcher Code-Pfad hängt dran, welche Module betroffen)
- `default_value` (Boolean — Wert wenn weder globaler Override noch Tenant-Override existiert)
- `lifecycle_stage` (Enum: `beta | rollout | ga | deprecated`)
- `owner_admin_user_id` (verantwortlicher Plattform-Admin)
- `target_module` (optionales Tag: in welchem Modul lebt der Flag, z.B. `crm`, `automation`, `kwp` — frei textuell)
- `archived_at` (Soft-Delete für GA-/Deprecated-Cleanup-Phase)

Indizes: `key` (unique), `lifecycle_stage`, `archived_at`.

FeatureFlagFilter

Pro Flag eine Liste von Filter-Regeln, die Tenants automatisch matchen. Wenn ein Tenant matcht und kein expliziter `TenantFeatureFlag`-Override existiert, gilt der Filter-Wert.

- `id`, `feature_flag_id`, `created_at`, `created_by_admin_user_id`
- `name` (Klartext: *„Tenants mit > 50 MA"*)
- `priority` (Integer — bei mehreren matchenden Filtern gewinnt höchste Priority)
- `value` (Boolean — was der Filter setzt)
- `criteria` (JSONB — Filter-Bedingung, siehe unten)
- `active` (Boolean — Filter ein-/ausschaltbar ohne Löschen)

Filter-Kriterien (JSONB-Schema, kombinierbar mit AND):
- `tenant_size` (Enum-Match: `1-10 | 11-25 | 26-50 | 51-100 | 100+`)
- `tenant_industry` (Enum-Match: `shk | ...`)
- `tenant_status` (Match auf `trial | active | past_due | frozen | suspended`)
- `subscription_product_id` (Match auf bestimmten Plan)
- `tenant_created_after` / `tenant_created_before` (Datum)
- `tenant_id_in[]` (explizite Liste — quasi „diese 5 Test-Tenants")
- `tenant_id_not_in[]`

Hinweis: Filter-Kriterien sind in Phase 1 bewusst klein gehalten. Erweiterungen in Phase 2 (Region, MRR-Schwelle, Custom-Tags).

TenantFeatureFlag (Override pro Tenant)

Expliziter Override eines Flags für einen Tenant. Hat **immer Vorrang** vor Filter und Default.

- `id`, `feature_flag_id`, `tenant_id`, `created_at`, `updated_at`
- `value` (Boolean)
- `set_by_admin_user_id`
- `note` (optionaler Klartext: *„Manuell aktiviert für Geerkens-Pilot"*)
- `expires_at` (optional — automatischer Rück-Reset auf Filter/Default)

Constraint: unique pro `(feature_flag_id, tenant_id)`.

Indizes: `(tenant_id, feature_flag_id)` für schnelles Lookup, `expires_at` für Cron-basiertes Aufräumen.

Auswertungs-Reihenfolge (deterministisch)

1. Existiert `TenantFeatureFlag` für diesen Tenant + Flag → dessen `value` gilt
2. Sonst: alle aktiven `FeatureFlagFilter` durchlaufen (sortiert nach `priority` absteigend), erster Match liefert `value`
3. Sonst: `FeatureFlag.default_value`

Block B — Routes

Ausschließlich auf `admin.hivegrid.de` (Plattform-Admin-Subdomain, siehe Modul 37). **Keine** Routen unter `app.hivegrid.de/{tenant}/...` in Phase 1.

```
admin.hivegrid.de/feature-flags/                 → Flag-Liste
admin.hivegrid.de/feature-flags/new              → Neues Flag anlegen
admin.hivegrid.de/feature-flags/:id              → Flag-Detail (inkl. Filter, Tenant-Overrides,
Audit-History)
admin.hivegrid.de/feature-flags/:id/edit         → Stammdaten editieren
admin.hivegrid.de/feature-flags/:id/filters/new  → Neuen Filter anlegen
admin.hivegrid.de/feature-flags/:id/filters/:fid → Filter editieren
admin.hivegrid.de/tenants/:tenantId/flags        → Pro-Tenant-Sicht aller Flag-Auswertungen
(siehe Modul 37)
```

Permissions: Alle Routen erfordern `admin.feature_flags.*` (Plattform-Admin-Permission, siehe Modul 37).

Block C — Listen-View und Detail-View

Flag-Liste (`admin.hivegrid.de/feature-flags/`)

Zu zeigende Daten pro Eintrag:
- `key`, `label`
- `lifecycle_stage`
- `default_value`
- Anzahl aktiver Filter
- Anzahl Tenant-Overrides
- `owner_admin_user_id`
- `target_module`
- Letzte Änderung (Datum + Admin-User)

Benötigte UI-Elemente:
- Tabellarische Listen-Anzeige
- Filter-Funktionen: nach `lifecycle_stage`, `target_module`, Owner
- Suche über `key` und `label`
- Sortierung pro Spalte
- Toggle „Archivierte anzeigen"
- Add-Funktion für neues Flag

Operationen:
- Detail öffnen
- Lifecycle-Stage-Wechsel (Quick-Action)
- Archivieren / Reaktivieren

Flag-Detail (`admin.hivegrid.de/feature-flags/:id`)

Zu zeigende Daten:
- Stammdaten (alle Felder aus `FeatureFlag`)
- Liste der `FeatureFlagFilter` mit Priority, Value, Criteria-Klartext, aktiv/inaktiv
- Liste der `TenantFeatureFlag`-Overrides mit Tenant-Name, Value, Note, gesetzt von, Expires-At
- Effektive-Auswertung-Statistik (Anzahl Tenants pro Endwert: *X Tenants haben Flag aktiv, Y nicht*)
- Audit-History (siehe Block G)

Benötigte UI-Elemente:
- Sektionen für: Stammdaten, Filter, Tenant-Overrides, Audit
- Inline-Edit für Stammdaten
- Filter-Editor (siehe Block D)
- Tenant-Override-Editor: Tenant-Suche (Auto-Complete) + Boolean-Wert + Note + optional Expires-At
- Bulk-Override-Funktion: mehrere Tenants gleichzeitig auf gleichen Wert setzen (z.B. *„diese 12 Tenants alle auf true"*)

- Lifecycle-Stage-Wechsel mit Bestätigungs-Dialog
- Archivieren-Funktion (mit Klartext-Hinweis bei aktiven Tenant-Overrides)

Operationen:
- Stammdaten editieren (`label` , `description` , `default_value` , `lifecycle_stage` , `owner` , `target_module`)
- `key` ist nach Anlage **nicht editierbar** (Code-Referenz-Stabilität)
- Filter anlegen/editieren/löschen/de-/aktivieren
- Tenant-Override anlegen/editieren/löschen
- Audit-History betrachten

Block D — Filter-Auswertung

Filter-Editor

Benötigte UI-Elemente:
- Eingabe für `name` , `priority` , `value` (Toggle)
- Criteria-Builder: pro Criteria-Schlüssel ein Eingabe-Element (Dropdown für Enums, Multi-Select wo passend, Datums-Picker für Created-Vor/Nach, Tenant-Auto-Complete für `tenant_id_in[]`)
- Live-Vorschau: *„Aktuell würden X Tenants matchen"* mit Klick auf Liste der gematchten Tenants
- Aktivieren/Deaktivieren-Toggle

Operationen:
- Criteria sind AND-verknüpft. OR über mehrere Filter-Einträge mit unterschiedlichen Kriterien lösbar.
- Live-Vorschau läuft auf Read-Replica gegen aktuelle Tenant-Tabelle.

Auswertungs-Reihenfolge bei Konflikten

- Mehrere Filter matchen einen Tenant → Filter mit höchster `priority` gewinnt
- Bei gleicher Priority: chronologisch ältester Filter gewinnt (deterministisch)
- Tenant-Override hat **immer** Vorrang vor jedem Filter

Klartext-Hinweise

- Filter ohne Criteria: *„Dieser Filter matcht aktuell alle Tenants — sicher gewollt? [Bestätigen]"*
- Filter überschneidet sich mit höher-priorisiertem Filter komplett: *„Dieser Filter ist effektiv wirkungslos, weil Filter ‚X' mit höherer Priority dieselbe Menge matcht."*

Block E — Code-Convention

Lookup-Funktion

```
await tenantHasFlag(tenantId, 'beta_new_dashboard')
```

Verhalten:
- Liefert Boolean
- Wendet Auswertungs-Reihenfolge aus Block A an
- Wirft, wenn `key` im Code referenziert wird, aber kein `FeatureFlag` -Eintrag existiert (Schema-Drift-Schutz)
- Ergebnis wird Request-scoped gecacht (kein wiederholtes DB-Roundtrip pro Render)

Code-Pattern

```
if (await tenantHasFlag(tenantId, 'beta_new_dashboard')) {
  // neuer Code-Pfad
} else {
  // alter Code-Pfad
}
```

Ausschließlich auf Boolean-Werte in Phase 1. Keine String-/Numeric-Variants. Phase 2 ggf. Multi-Variant.

Ohne Tenant-Kontext

- Hintergrund-Jobs / Worker, die ohne Tenant laufen, dürfen Flag-Lookup nur mit explizitem Tenant-Argument verwenden
- Plattform-globale Flags (ohne Tenant-Bezug) sind in Phase 1 **nicht vorgesehen** — wenn nötig, einfach als `FeatureFlag` ohne Filter und mit `default_value` arbeiten und über separaten globalen Helper aufrufen

---

Block F — Flag-Lifecycle

Stages

Stage	Bedeutung	Default-Filter-Empfehlung
`beta`	Neu, einzeln aktiviert für Test-Tenants	`default_value=false`, einzelne Tenant-Overrides oder enger Filter
`rollout`	Breit aktiviert über Filter, Beobachtungs-Phase	`default_value=false`, breite Filter (z.B. *„Tenants > 50 MA"*)
`ga`	Always-On, Code-Cleanup geplant	`default_value=true`, Filter werden überflüssig
`deprecated`	Abkündigung, Code wird entfernt	`default_value=false`, Tenants kriegen Klartext-Hinweis im Audit

Lifecycle-Wechsel

Workflow:
- Plattform-Admin wählt neuen Stage im Detail-View
- Bestätigungs-Dialog mit Klartext-Beschreibung der Konsequenzen
- Audit-Eintrag wird geschrieben

Edge-Cases:
- Wechsel auf `ga` mit aktiven Filter-Regeln, die `false` setzen: Klartext-Warnung *„Filter ‚X' setzt das Flag auf false — bei GA wird default_value=true ignoriert, falls Filter höher priorisiert sind."*
- Wechsel auf `deprecated`: Klartext-Hinweis an Owner-Admin: *„Bitte Code-Cleanup einplanen — Owner: [Name]"*

Archivierung

- Nur für `ga` und `deprecated` Flags möglich
- Aktive Tenant-Overrides werden bei Archivierung **nicht** automatisch gelöscht (Audit-Spur bleibt)
- Klartext beim Archivieren mit aktiven Overrides: *„Es gibt noch X Tenant-Overrides — Archivierung verbirgt das Flag aus der Standard-Liste, Overrides bleiben funktional. Sicher fortfahren?"*

Block G — Audit-Logging

Pro Flag-Änderung wird ein Audit-Eintrag geschrieben (in Modul 21 oder dediziert in `AdminAuditLog`-Tabelle, siehe Modul 37).

Geloggt wird:
- Flag-Erstellung/-Änderung mit Diff (`label` , `description` , `default_value` , `lifecycle_stage` , `target_module` , `owner`)
- Filter-Erstellung/-Änderung/-Löschung mit Diff
- Tenant-Override-Erstellung/-Änderung/-Löschung mit Diff
- Lifecycle-Stage-Wechsel
- Archivierung/Reaktivierung

Pro Eintrag:
- Admin-User (Akteur)
- Zeitstempel
- Vorher/Nachher-Diff
- Optionaler Note-Text vom Admin (z.B. *„Geerkens-Pilot-Aktivierung"*)

Sichtbarkeit:
- Im Detail-View des Flags als Audit-History-Sektion
- In Modul 37 als Cross-Tenant-Audit-Log durchsuchbar

Block H — Permissions

Permission-IDs (Plattform-Admin-Scope, siehe Modul 37)

Permission	Erlaubt
`admin.feature_flags.read`	Flag-Liste + Detail einsehen
`admin.feature_flags.write`	Flags anlegen/editieren, Filter und Tenant-Overrides verwalten
`admin.feature_flags.lifecycle`	Lifecycle-Stage wechseln, archivieren

Bundle-Mapping

Bundle	Atomare Permissions
`admin.feature_flags.none`	(keine)
`admin.feature_flags.viewer`	`read`
`admin.feature_flags.operator`	`read` + `write`
`admin.feature_flags.owner`	`read` + `write` + `lifecycle`

Tenant-User-Sicht (Phase 1)

- **Keine Tenant-User-Routen** für Feature-Flags in Phase 1
- Tenants sehen aktive Flags nur indirekt: Features sind sichtbar oder nicht
- Klartext-Hinweise für nicht-aktivierte Features kommen aus dem jeweiligen Modul (Subscription-Gate-Logik aus Modul 27 Block F.1, nicht aus Feature-Flags)

Block I — Klartext-Hinweise / -Fehler

- Lookup auf nicht-existierenden `key` im Code: *„Feature-Flag ‚beta_xy' ist nicht definiert — bitte im Admin-UI anlegen oder Code-Referenz korrigieren."* (wirft im Dev/Staging, loggt nur im Prod)
- Tenant-Override anlegen für inaktiven/archivierten Tenant: *„Tenant ist im Status ‚deletion_pending' — Override anlegen?"* mit Bestätigung
- Filter-Criteria führt zu 0 matchenden Tenants: dezenter Hinweis *„Aktuell matcht kein Tenant"* (kein Block, ist erlaubt — z.B. Filter für zukünftige Tenants)
- Override mit `expires_at` in der Vergangenheit: Klartext beim Anlegen *„Expires-At liegt in der Vergangenheit — Override wird beim nächsten Cron-Lauf entfernt."*

Block J — Phase 2 / spätere Features

- **User-spezifische Flags** — Flag pro `(tenant_id, user_id)` statt nur pro Tenant
- **Tenant-self-service-Flags** — Tenant-Admin sieht in `/{tenant}/admin/labs/` Toggle *„Beta-Features anzeigen"* und kann ausgewählte Flags selbst aktivieren (Whitelist im `FeatureFlag` Definition: `tenant_self_service_enabled` Boolean)
- **Multi-Variant-Flags** — String-/Numeric-Werte (A/B/C-Tests, nicht nur Boolean)
- **Rollout-Prozentsatz** — *„aktiviere für 25% der Tenants"* mit deterministischem Hash
- **Erweiterte Filter-Kriterien** — Region, MRR-Schwelle, Custom-Tags am Tenant
- **Schedule-basierte Aktivierung** — *„aktiviere Flag X automatisch ab 1.6.2026"*
- **Telemetrie pro Flag** — wie viele User haben ein Flag-getriggertes Feature genutzt (Verbindung zu Analytics)

Block K — Architektur-Vorschriften (Querschnitt)

- **AV #6 — Audit-Trail-Default:** alle Flag-Änderungen geloggt (Block G)
- **AV #7 — Feature-Flag-Hülle:** dieses Modul **ist** die Implementierung dieser Architektur-Vorschrift; alle neuen Features in Phase 1 werden mit Flag eingeführt
- **LP-10 — Klartext-Diagnostik:** alle Fehler-Cases mit menschenlesbarem Hinweis (Block I)
- **LP-9 — Spec-Tiefe:** keine UI-Layouts, nur Anforderungen

Detail-Spec-Status: DRAFT

Erste Detail-Spec für Modul 32. Bewusst kompakt gehalten — Modul ist klein, Phase 1 minimal.

Offene Detail-Punkte (nicht-blockierend):
- Konkrete Permission-IDs final mit Modul 37 abstimmen
- Caching-Strategie pro Request (in-memory vs. Redis-shared) im Implementierungs-Schritt
- Schema-Drift-Check: ob bei Code-Deploy unbekannte Flag-Keys automatisch geseeded werden sollen (Implementierungs-Detail)
- Subframe-UI-Designs separat