HIVEGRID

ENGINEERING HANDBOOK

Entwicklungs-Regeln & Infrastruktur

Verbindliche Standards für die gemeinsame Entwicklung

0 Warnings | TypeScript strict | Hohe Test-Coverage | Pflicht-E2E | Accessibility AA | EU-Stack | ~80 Quality-Gates

Produkt: HiveGrid 2.0 – B2B-SaaS für CRM & projektbasiertes Arbeiten

Codebasis: Git-Downstream des ToolCTRL-Boilerplates (pnpm-Monorepo)

Stand: 14. Juni 2026

Grundsatz: Regeln sind durch automatische Gates erzwungen, nicht durch guten Willen. Was rot ist, wird nicht gemerged.

HIVEGRID

Inhalt

1. Was HiveGrid Cloud ist (Produktkontext)
2. Codebasis-Modell: Git-Downstream & Package-Namen
3. Grundprinzipien
4. Code-Qualität & Linting (0 Warnings)
5. TypeScript & Typsicherheit
6. Formatierung (Prettier)
7. Unit- & Integrationstests + Coverage
8. End-to-End-Tests (Playwright)
9. Accessibility (a11y)
10. Quality-Gates: Commit → Push → Merge → Deploy
11. CI/CD-Pipeline
12. Infrastruktur & Hosting
13. Architektur-Konventionen (inkl. EU-Stack)
14. Onboarding-Checkliste

1. Was HiveGrid Cloud ist

HiveGrid 2.0 ist ein B2B-SaaS für Kontaktmanagement (CRM) und projektbasiertes Arbeiten – mit konsequent europäischem Datenschutz- und Hosting-Anspruch.

- **Kern-Domäne:** Kontakte/Leads → projektbasierte Pipelines (Vertrieb, HR-Onboarding, Kundenbetreuung) → Auftragsabwicklung.
- **Datenmodell:** typisierte First-Class-Tabellen (z. B. `Kontakt`, `Projekt`) statt generischer Entity-Builder.
- **EU-Stack (nicht verhandelbar):** AI über EU-Provider (Scaleway, Mistral, Aleph Alpha; keine US-Provider), EU-Hosting auf Hetzner/Coolify, Auth über E-Mail+Passwort / Magic-Link / TOTP (kein Apple/Google/Microsoft-OAuth), Per-Tenant-Envelope-Encryption, Postgres mit Row-Level-Security als Defense-in-Depth.
- **Bauweise:** aufgeteilt in 176 Milestones (Pflichtenheft v1.1), Phase 0 → Fundament → Wertschöpfung.

> **Begriffsklärung**
> **HiveGrid Cloud** ist die SaaS-Anwendung (dieses Repo, in aktiver Entwicklung). Die Marketing-Website `hivegrid.de` ist ein separates Projekt auf einem anderen Host – nicht zu verwechseln.

2. Codebasis-Modell: Git-Downstream & Package-Namen

HiveGrid Cloud ist **kein klassischer Fork**, sondern ein **Git-Downstream** des ToolCTRL-Boilerplates: Verbesserungen aus dem Boilerplate werden per `git merge upstream/main` übernommen. Damit das funktioniert, bleiben Package-Namen und Konfiguration absichtlich kompatibel.

> **Wichtig zu den Package-Namen (häufiges Missverständnis)**
> Es gibt **kein flächiges Rename** auf `@hivegrid/*`. Geteilte Boilerplate-Pakete behalten `@toolctrl/*` – sonst brechen Upstream-Merges. Entscheidung dokumentiert in `docs/ARCHITECTURE-DECISIONS.md` und `AGENTS.md`.

Scope	Wofür	Beispiele
`@toolctrl/*`	Alle vom Boilerplate geteilten Pakete (bleiben so, für Mergebarkeit)	`@toolctrl/web`, `@toolctrl/api`, `@toolctrl/db`, `@toolctrl/foundation`, `@toolctrl/ui-app`, …
`@hivegrid/*`	**Nur** genuin neue HiveGrid-Pakete, die es upstream nie gibt	`@hivegrid/activity` (Activity-Stream-Service)

Mapping-Hinweis: Wo ältere Docs/Pläne `packages/ui-hivegrid` oder `@hivegrid/foundation` sagen, ist real `packages/ui-app` bzw. `@toolctrl/foundation` gemeint. Ein späterer Naming-Drift-Fix ist in den ADRs als Option notiert, aber bewusst noch nicht umgesetzt.

Neu in HiveGrid: `@hivegrid/activity` – zentraler Activity-Stream/Änderungs-Historie, hängt nur an `@toolctrl/db`. Weitere HiveGrid-spezifische Pakete folgen demselben Muster.

Node-Version: HiveGrid nutzt `v22.12.0` (`.nvmrc`) – neuer als der Boilerplate. Vor der Arbeit `nvm use` ausführen.

3. Grundprinzipien

- **Linting:** Kompromisslos – `--max-warnings=0`. Eine Warning bricht den Build.
- **Typsicherheit:** TypeScript `strict`. `any`/Type-Escapes nur über eine gedeckelte Baseline.
- **Tests:** Sehr hohe Coverage auf kritischen Pfaden (Auth, Billing, Tenant-Isolation ~99 %), erzwungen pro Verzeichnis.
- **End-to-End:** Pflicht. Kritische Flows müssen als Playwright-Specs existieren; ein Gate prüft wörtliche Test-Inhalte.
- **Accessibility:** WCAG 2.2 AA als Gate (Farbkontrast, Mobile-Overflow).
- **EU-Datenschutz by design:** RLS, Envelope-Encryption und EU-only-Provider sind Architektur-Vorgaben, kein Add-on.
- **Automatisierung statt Disziplin:** ~80 `verify:*`-Gates erzwingen die Regeln in Pre-Commit-Hooks und CI.

> **Leitsatz**
> Fehler nach links schieben: Was lokal in Sekunden gefangen werden kann, läuft im Pre-Commit-Hook. CI ist Sicherheitsnetz, nicht erster Prüfer.

4. Code-Qualität & Linting

ESLint mit `@typescript-eslint`, `import`, `jsx-a11y`, `react-hooks` (Basis `plugin:jsx-a11y/recommended`). Konfiguration identisch zum Boilerplate.

> **Die wichtigste Regel**
> `pnpm run lint` = `eslint --max-warnings=0 --ext .tsx,.ts .` — **Null Warnings.** Es gibt keine „gelben" Builds.

Harte Regeln (Error-Level)

Regel	Wirkung
`no-debugger`	Kein `debugger`.
`no-console` (Server)	In Server-Code (`*.server.ts`, `apps/api`, `apps/worker`, `packages/*/src`, Service-/DB-Layer) verboten; in UI/Tests/Skripten erlaubt.
`no-restricted-imports`	Kein direkter `@prisma/client` (nur über `@toolctrl/db`); keine direkten `src/*`-Imports zwischen Packages; `foundation` / `view-models` ohne React/UI; UI-Packages ohne DB.
`no-restricted-syntax` (Phase-C-UI)	Keine hartkodierten nutzersichtbaren Strings in `ui-app` / `ui-subframe` / `ui-mock` – alles über i18n-Keys.

Accessibility-Lint & Escape-Hatch-Baseline

jsx-a11y-Regeln „monoton fallend": bei 0 Verstößen werden sie auf `error` hochgestuft und können nie wieder regredieren. Type-/Lint-Ausnahmen sind gedeckelt (`verify:eslint-baseline`): max. **262** `eslint-disable`, **35** `@ts-ignore`, **43** `@ts-expect-error`. Neue brauchen Baseline-Erhöhung + Review.

Datei-Limits: `apps/api/src/app.ts` ≤ 450 Zeilen, API-Route-Module ≤ 500 Zeilen (`verify:source-health`).

5. TypeScript & Typsicherheit

- `strict: true` (inkl. `noImplicitAny`, `strictNullChecks` …). `any` praktisch verboten.
- Typecheck ist Pflicht nach Routen-Änderungen und läuft als Pre-Push-Hook.

> **React-Router-v7-Regel**
> Routen-Typen **immer** aus `./+types/[routeName]` (nie relativ). Fehlen Typen → `pnpm run typecheck`, nie den Importpfad „reparieren".

6. Formatierung (Prettier)

Einstellung	Wert	Einstellung	Wert
printWidth	160	semi	true
singleQuote	false (Double Quotes)	tabWidth	2 (Spaces)
trailingComma	es5	arrowParens	always
endOfLine	lf	Plugin	prettier-plugin-tailwindcss

7. Unit- & Integrationstests + Coverage

Vitest + Testing Library (jsdom), Coverage-Provider v8. Schwellen identisch zum Boilerplate und **pro Verzeichnis** erzwungen (`verify:test-coverage` gegen `scripts/coverage-targets.json`).

Kritischer Pfad	Statements	Branches
API Auth	99 %	92 %
Billing	99 %	97 %
Tenant-Isolation	99 %	99 %
Session / Permissions	99 %	98 %
Stripe	95 %	90 %
Basis-Targets (z. B. apps/api)	70 %	60 %

Test-Dichte-Gate (Pre-Commit): Mindestverhältnis Tests/Quelldateien je Workspace (z. B. `apps/api` ≥ 0,18; App-Routes ≥ 0,05; `packages/brand` / `config` ≥ 0,50). Neue HiveGrid-Pakete (z. B. `@hivegrid/activity`) brauchen eigene Tests.

Konventionen: `*.test.ts(x)` / `*.spec.ts`; Boundary `*boundary.test.ts`; a11y `*.a11y.test.tsx` / `*.axe.test.tsx`.

8. End-to-End-Tests (Playwright)

Verzeichnis `tests/e2e` (~39 Specs). Chromium, optional Mobile-Matrix (Pixel 5 + iPhone 13). Seriell, Timeout 120 s, Traces bei Fehlern. E2E laufen nächtlich gegen die Live-Staging-Umgebung; Basis ist ein deterministischer Readiness-Tenant (`seed:e2e-readiness`).

a11y-Coverage-Gate (pinnt Tests wörtlich)

`verify:a11y-coverage` verlangt ≥ 12 Component-a11y-Tests und ≥ 7 E2E-a11y-Specs und prüft wörtliche Snippets kritischer Flows (z. B. `expectNoCriticalA11yViolations(` , `expectNoHorizontalOverflow(`).

> **Nach E2E-Refactors**
> Das Gate matcht **literale Strings**. Tests umbenennen/umbauen → `verify:a11y-coverage` erneut laufen lassen.

RR7 Single-Fetch-Interceptor-Contract

Form-POST-Interceptoren müssen das `.data` -Ziel matchen und **turbo-stream** zurückgeben (kein rohes JSON), inkl. `X-Remix-Response: yes` . Helfer in `tests/e2e/utils/navigation.ts` .

Pflicht-Flows (Auszug): Registration/Onboarding, Golden-Path, Subscription-Smoke, Plugin-Settings/-Lifecycle, Workflow-Builder+Run, Members, Audit, Profile-Security, Privacy, API-Keys, Mobile-Shell.

9. Accessibility

jsx-a11y-Lint · `verify:color-contrast` (WCAG 2.2 AA) · axe-core-Suite (`test:a11y`) · Mobile-Overflow-/Violation-Checks in E2E · Lighthouse-a11y als Warnung.

10. Quality-Gates: Commit → Push → Merge → Deploy

Beim Commit lefthook pre-commit · parallel

lint-staged (ESLint auf geänderten Dateien) · `verify-secrets` · `verify-eslint-baseline` · `verify-test-density` .

Beim Push lefthook pre-push · seriell

`typecheck` · `verify-hardcoded-strings` (i18n). Notfall-Bypass nur mit `LEFTHOOK=0` , nie auf `main` .

Beim Merge (CI) blockiert

Jobs `quality` + `api` + `worker` + `database` + `docker-smoke` müssen grün sein (Typecheck, Lint, Tests, Build, Bundle-Size, a11y, ~18+ Baseline-Gates).

Beim Deploy blockiert

Preflight-Gates (`deploy-readiness` , `docker-baseline` , `docker-non-root`). Produktions-Promote erfordert `backup_confirmed=true` .

11. CI/CD-Pipeline

Zwei parallele Welten mit gleichen Gates: GitHub Actions und **Gitea** (self-hosted Runner). Eigene Gates sichern die Parität. `ci.yml`-Jobs: `quality`, `api`, `worker`, `database`, `docker-smoke`.

Die ~80 `verify:*`-Gates sind **Baseline-/Anti-Regressions-Gates**. Gruppen: Security/Auth · Mandantentrennung/Boundaries · Docker/Deploy · i18n/a11y · API/Specs · CI-Struktur · Docs.

> **Lighthouse & SEO**
> Performance ≥ 95 % ist hart gegated; a11y/Best-Practices/PWA nur Warnung. **SEO ist bewusst nicht gegated** (B2B-SaaS).

12. Infrastruktur & Hosting

HiveGrid Cloud läuft im gemeinsamen Hetzner-Estate (EU/Deutschland), zentral über Coolify gesteuert. Detail-Inventar im separaten Dokument „Server-Struktur". Das Wesentliche:

- **Control-Plane:** Coolify v4 auf der `control`-VM (Dashboard `saas-ctrl.com`) steuert alle Hosts per SSH.
- **Staging:** Ein gemeinsamer Host `staging-1` trägt mehrere Produkt-Stagings nebeneinander – nicht eine VM pro Produkt.
- **Produktion:** Laut ADR-009 erhält jedes SaaS-Produkt ein **eigenes Hetzner-Projekt** für Prod (eigene DB/App). Das HiveGrid-Cloud-Prod-Projekt wird bei Reife angelegt; die SaaS-App ist aktuell in Entwicklung und noch nicht deployt.
- **Zentrale Dienste (geteilt über WireGuard-Mesh, auf `node1`):** Sentry (Errors), Loki (Logs), Meilisearch (Suche, per-Produkt-Scoped-Key), ClamAV (Virenscan), mCaptcha; Prometheus + Grafana auf `control`.
- **Services pro App-Stack:** web (8080), api (8080), worker ×5 Queue-Gruppen (general/mail/import/webhook/plugin), Postgres 16, PgBouncer (Transaction-Pooling), Redis 7, Authentik (Admin-SSO).
- **Deploy:** build-once-deploy-many über die Gitea-Registry; Coolify zieht das fertige Image und startet die Container neu (~30 s). Rollback = altes SHA-Image neu deployen.

> **EU-Hosting**
> Compute, Daten und Object-Storage (S3) liegen in der EU – auf Hetzner (Deutschland), orchestriert über Coolify. Das stützt die EU-Datenschutz-Vorgaben aus Abschnitt 13.

13. Architektur-Konventionen

- **Upstream-Disziplin:** Geteilten Code so lassen, dass `git merge upstream/main` sauber bleibt. Neues HiveGrid-Verhalten in `@hivegrid/*`-Paketen oder klar abgegrenzten Modulen.
- **Package-Grenzen:** Immer öffentliche Paketnamen importieren, nie fremdes `src/*`. Server-DB nur über `@toolctrl/db/client.server`, Env über `@toolctrl/config`, Queues über `@toolctrl/queue`.
- **EU-Stack erzwingen:** Keine US-AI-Provider (nur EU-Provider: Scaleway, Mistral, Aleph Alpha); kein Social-OAuth (nur E-Mail/Passwort, Magic-Link, TOTP); Per-Tenant-Envelope-Encryption; Postgres-RLS als zusätzliche Mandanten-Grenze.
- **URLs:** host-aware Helfer (`href()`, `appPath` / `adminPath` / `apiPath`), keine handgebauten Pfade.
- **Secrets:** nie in UI-Code; nur über `@toolctrl/config` / Server-Boundaries. Tenant-Filter bei jedem Datenzugriff erhalten.
- **RR7:** Typen aus `./+types/...`; `<Outlet/>` in Layouts; `typecheck` nach Routen-Änderungen.
- **Doku mitführen:** bei Feature-/Provider-/Worker-Änderungen `docs/feature-manifests/*.json`; bei mutierenden Routen `docs/security/csrf-inventory.md`; bei Rollen/Permissions `docs/roles-matrix.md`. HiveGrid-Fortschritt in `docs/PROGRESS-TRACKER.md`, Entscheidungen in `docs/ARCHITECTURE-DECISIONS.md`.

14. Onboarding-Checkliste

Vor dem ersten Commit

1. `nvm use` (Node `v22.12.0`), dann `pnpm install` – installiert die Hooks automatisch.
2. `AGENTS.md` + `docs/ARCHITECTURE-DECISIONS.md` lesen (Package-Modell, EU-Stack, Milestone-Struktur).
3. `pnpm run dev` starten – generiert die RR7-Routen-Typen.
4. `pnpm run typecheck` & `pnpm run lint` sauber (0 Warnings); vor Push `pnpm test` grün.
5. Für E2E: `seed:e2e-readiness`, `.env.e2e-readiness` sourcen, dann `verify:a11y-coverage` + Playwright.

Goldene Regel: Wenn ein Gate rot ist, ist die Lösung der Code – nicht das Umgehen des Gates.

Stand 14. Juni 2026. Quellen: HiveGrid-Cloud `package.json`, `.eslintrc`, `.prettierrc`, `tsconfig.json`, `vitest.config.ts`, `playwright.config.ts`, `lefthook.yml`, `scripts/coverage-targets.json`, `AGENTS.md`, `docs/ARCHITECTURE-DECISIONS.md` sowie das `infra`-Repository. Konfiguration ist – bis auf Node-Version und HiveGrid-spezifische Pakete/Docs – identisch zum ToolCTRL-Boilerplate.