SkillPilot Glossary

Shared Vocabulary for Curricula, Skill Graphs,
Learner Runtime, Coach Adapters, and Quality Governance

Purpose of This Glossary

This document defines the core terms SkillPilot coins and relies on, and draws clear boundaries between closely related concepts. It is the shared vocabulary for curriculum authors, Curriculum Champions, developers, and AI coach adapters.

As of 11 November 2026 · License: Apache-2.0 (technical concept reference; see LICENSING.md).

The linked normative document always takes precedence over this summary.

How to read this page

- **Definitions are short on purpose.** Each entry states what the term means and, where it matters, what it is *not*. The linked document stays the normative source; this page never overrides it.
- **DE** gives the German term used in the app, in German documentation, and by the learning coach. Field names such as `contains`, `requires`, or `applicability` are code identifiers and stay English in both languages.
- **Terms are grouped by layer**, from curriculum sources up to quality governance, because most misunderstandings come from mixing layers rather than from a single word.

Two rules explain a large part of the vocabulary below:

- **Goals are stable, structure is contextual.** A goal is a learnable, assessable competence with a stable identity; years, phases, tracks, and jurisdictions are context around it.
- **As much semantics as necessary, as little ontology as possible.** A new distinction is introduced only when it solves a concrete authoring, migration, validation, projection, or runtime problem.

Contents

How to read this page 2

1 Curricula, skill graphs, and provenance 4

2 Inside the skill graph 5

3 Node types (learning modes) 9

4 Structure layers and projection 10

5 Learner runtime 14

6 Coach, session, and identity 17

7 Quality, review, and governance 19

8 Distinctions that are easy to get wrong 20

Maintenance 21

1 Curricula, skill graphs, and provenance

Curriculum

DE: Curriculum, Lehrplan — see Whitepaper

The external, authority-defined body of rules SkillPilot docks onto: a state syllabus, a module handbook, or a standard such as CEFR. The curriculum remains the **normative source**; Skill Pilot does not create standards, it makes existing ones operational.

Not the same as the skill graph derived from it. In the quality dashboard, "Curriculum" is used more narrowly for one canonical skill landscape file.

Skill graph

DE: Skill-Graph — see Graph Definition

The versioned, machine-readable **operative model** derived from a curriculum: goals connected by `contains` and `requires`, together with their semantics, validity conditions, and frontier logic. The skill graph is what SkillPilot builds, validates, and navigates; the curriculum stays the normative source above it.

Skill landscape

DE: Skill-Landschaft — see Graph Definition

One delimited, published instance of a skill graph: a single subject-and-school-type unit identified by a `landscapeId`, owned by champions, rated with a maturity level, and shipped as a release package. A skill landscape may be serialized into several locale files that share its ID.

Use *skill graph* when the point is the model, *skill landscape* when the point is **which** one.

Canonical landscape

DE: kanonische Landschaft — see Canonical Gymnasium Rollout Policy

A skill landscape maintained once for a whole school type and reused by all jurisdictions, instead of one separately authored landscape per state. Jurisdiction-specific differences are expressed through applicability and composition views, not by copying goals.

Root curriculum and curriculum manifest

DE: Wurzelcurriculum, Curriculum-Manifest — see Levels of Personalization

A root curriculum is a skill landscape that is selectable in the UI and valid for champion registration. The set of root curricula is declared explicitly in curricula/curriculum_manifest.json; CI fails if the manifest and the computed set of roots disagree.

Source snapshot

DE: Quellen-Snapshot — see Curriculum Quality

A captured, parseable extract of an official curriculum document, kept in the repository together with an official HTTP(S) link to the original. Coverage claims are only ever proven against the snapshots present in the repository.

Source goal and source registry

DE: Originalziel, Source-Ziel; Source-Registry — see Curriculum Quality

A source goal is a learning objective as written in the original curriculum. The source registry records which source goals exist and how their **atomic source closure** maps onto canonical goals. Coverage is checked in both directions: every visible atomic goal needs source evidence, and every registered source goal must map completely into the view.

`sourceRef` vs `resourceLinks`

DE: Herkunftsnachweis vs. Lernressourcen — see Source And Resource Links

`sourceRef`, mappings, and provenance answer "Why does this goal exist?" Official curriculum references remain evidence, not optional study material.

The Content Integration Architecture puts optional external learning materials in separate packages that reference goal IDs; the learner selects packages independently of the curriculum. `resourceLinks` remains the compatibility field for existing resources and the separately governed SkillPilot-owned goal visualizations. Direct external didactic links in canonical goals are migration debt, not the target authoring model.

Learning material must never be smuggled in as provenance, and provenance is not a study link.

Runtime package and ontology package

DE: Runtime-Paket, Ontologie-Paket — see Dual Curriculum Package Releases

The two equivalent release artifacts of one curriculum release: a JSON runtime package consumed by the application, and an FWU-style ontology package for semantic interchange. Both are produced from the same reviewed release unit.

2 Inside the skill graph

Goal

DE: Lernziel — see Graph Definition

A node of the skill graph: a learnable, assessable competence, usually phrased as "Die lernende Person kann …". Every goal has a stable UUID `id`, a `title`, and a positive `weight`.

A year, semester, module, phase, or track is **not** a goal.

Atomic goal

DE: atomares Lernziel — see Graph Definition

A goal with no `contains` children. Atomic goals are the assessable units: mastery is recorded on them, the frontier is computed over them, and in the target state all authored `requires` edges connect atomic goals.

The classification is derived from the graph structure, not authored. Stored classification metadata must agree with the derived one.

Cluster goal

DE: Cluster-Ziel — see Graph Definition

A goal with at least one `contains` child. Clusters serve navigation, filtering, and aggregated progress. Their satisfaction is derived from their atomic descendants rather than recorded directly.

Atomic basis

DE: atomare Basis — see Graph Definition

Written Atoms(g): the goal itself if it is atomic, otherwise its atomic `contains` descendants. It is the set whose mastery witnesses satisfaction of a cluster.

`contains`

DE: enthält — see Graph Definition

The authored hierarchy relation C: (p, c) means parent p directly contains child c in **content** terms. Its arrow points from parent to child. `contains` must be acyclic but allows multiple parents (polyhierarchy).

Ancestor/descendant relationships are derived through paths of one or more `contains` edges. `contains` is not a prerequisite and not a program placement.

`requires`

DE: setzt voraus — see Graph Definition

The authored prerequisite relation R: (g, p) means goal g directly requires prerequisite p. Its arrow points from the dependent goal to its prerequisite; p must be satisfied before g is approached. It must be acyclic. Under the binding target state, all authored `requires` edges connect **atomic** goals ($R \subseteq A \times A$); existing prerequisites on clusters are evaluated via `contains` inheritance and atomic expansion during migration.

Both `contains` and `requires` record direct relationships; derived relationships are computed rather than modeled as extra relation types.

Inherited prerequisites

DE: vererbte Voraussetzungen — see Graph Definition

Additional prerequisites derived from the `requires` of a goal's `contains` ancestors, excluding pairs already authored on the goal itself: $R_{\text{inherited}} = R_{\text{eff}} \setminus R$. With multiple parents, all ancestor paths contribute. Inheritance through containment is distinct from following a chain of prerequisites.

Effective prerequisites

DE: effektive Voraussetzungen — see Graph Definition

The derived prerequisite relation $R_{\text{eff}} = R \cup R_{\text{inherited}}$, combining a goal's own `requires` with prerequisites inherited from its `contains` ancestors and expanded onto atomic basis goals (R_{at}). In a fully atomic-authored prerequisite graph, authored and effective atomic prerequisites coincide.

Prerequisite reachability

DE: Erreichbarkeit von Voraussetzungen — see Graph Definition

Goals connected by one or more edges of the specified prerequisite relation, written R^+ or R^+_{at}. This transitive closure includes single edges and longer chains. A learning-flow view uses the inverse direction and labels it explicitly.

Weight

DE: Gewicht — see Graph Definition and AGENTS.md

A strictly positive number expressing a goal's share in progress. Convention in the canonical landscapes: atomic goals carry `weight = 1`, clusters carry the number of their *unique* atomic descendants, so a goal reachable through several parents is counted once.

Goal ID and `shortKey`

DE: Lernziel-ID, Kurzschlüssel — see Graph Definition

`id` is the canonical UUID identity of a goal; it may be shown to learners and coaches and is not an access credential. `shortKey` is an optional stable ASCII identifier for exports, APIs, and cross-layer references; it is unique per logical landscape and never replaces `id`.

Graph validity

DE: Graphgültigkeit — see Graph Definition

A graph is valid iff IDs are unique, `contains`, `requires`, and the effective atomic prerequisite relation are acyclic, and both minimality rules hold:

- **Local minimality:** a prerequisite already inherited from a `contains` ancestor must not be restated on the child node.
- **Transitive minimality:** an edge must not be present if the same prerequisite already follows from other effective prerequisite paths.

Motivation anchor

DE: Motivationsanker — see Context & Views

An atomic orientation goal that opens a didactic route by answering "why this subject or topic?", for example `Warum Physik? - Weltverständnis & Zukunft`. Route checks measure connectivity back to such an anchor.

Terminal autonomy goal

DE: terminales Autonomieziel — see Context & Views

An atomic goal at which the learner performs independently, typically an exam-mode exercise goal such as the atomic tasks under `Übungen Q2`. It closes a didactic route.

Route coverage

DE: Routenabdeckung — see Context & Views

The property that every route-relevant atomic goal lies on at least one prerequisite path from a motivation anchor to a terminal autonomy goal. The intended learning direction runs from orientation through understanding (and parallel memorization where needed) to independent application.

3 Node types (learning modes)

Pedagogical goal kinds are orthogonal to the structural atomic/cluster split. See Node Types.

Understanding

DE: Verstehen

The default type: conceptual or procedural competence, coached and practised in dialogue. A goal with no special exam or memorization fields is an Understanding node; orientation goals are marked explicitly and do not test subject knowledge.

Memorize

DE: Sich merken

A spaced-repetition goal for facts, vocabulary, notation, or selected formulas. It is mastered only when no required card is due **and** every required card has passed an unaided verified recall test.

Exam

DE: Selbstständig Probleme lösen, Prüfungsknoten

A goal carrying `examData`: an independent assessment or practice task with a model solution and scoring scheme. `requires` gates access to the task and drives targeted remediation after an unsuccessful attempt.

`examData` and scoring

DE: Prüfungsdaten, Bewertung — see Node Types

Task content, solution content, and a scoring object with `maxPoints`, `passingPoints`, and per-step points: total $= \min(\sum \text{step points}, \text{maxPoints})$, passed iff total $\geq$ passingPoints. Only passing writes mastery (`1.0`); an unsuccessful attempt is explained in dialogue without altering stored learner state, so the task can be retried.

Exam Mode

DE: Prüfungsmodus — see Node Types

The runtime mode in which the AI coach acts as a strict, neutral exam supervisor: the task is delivered verbatim, no hints are given during work, one complete submission is graded step by step, and only afterwards does the coaching role resume.

SRS and SM-2

DE: Kartei, Wiederholungsplanung — see Node Types

Spaced-repetition scheduling following the SuperMemo-2 algorithm. Each card stores `interval`, `repetition`, easiness factor `ef`, and `nextReview`. A card is **due** when `nextReview <= now` or when it has no state yet.

Verified recall

DE: geprüfter Abruf — see Node Types

The hard recall gate on memorization cards: the coach shows only the prompt, the learner answers unaided, and only afterwards does the coach retrieve the expected answer and persist `passed` or `failed`. Ordinary SRS practice in the Cockpit never substitutes for this check.

4 Structure layers and projection

Layer model

DE: Schichtenmodell — see General Goal System and Migration

SkillPilot separates five layers that older legacy graphs sometimes mixed:

Layer	Holds	Example
Goal layer	assessable competences and content clusters	”lineare Funktionen verknüpfen”
Program layer	structural units	`Jahrgang 8`, `E-Phase`, module, track, exam
Placement layer	links from goals into program units	”this goal belongs to Q1”
Competency-axis layer	taxonomy entries	`K1 Mathematisch argumentieren`
Composition-view layer	scope-specific learner-facing trees	`DE-BY / Gymnasium / SekII / Mathematik / LK`

Program unit

DE: Programmeinheit — see View Projection and Goal Placement

An explicit structural container (year, semester, module, phase, track, exam section) forming an ordered tree via `parentUnitId` and `order`. A content goal must never be reused as a stand-in for a program unit.

Learning unit

DE: Lerneinheit — see Learning Units and Optional Planning (§12)

A recursively composed educational whole with a stated learning outcome, motivational introduction, constituent goals or subunits, and a reviewed completion rule (`LearningUnit`). A year, semester, or arbitrary navigation cluster does not become a learning unit solely by grouping goals.

Each target has one direct acquisition owner in the resolved unit hierarchy; parent units aggregate descendants. Completion follows from required children and the unit's own goal-type obligations without a stored unit-mastery result. An open parent exam does not reopen completed child competencies, and passing a parent exam never passes protected child exams or cards. Effective competence remains global, and individual recognition can precede complete unit completion; it is only sound inside units that satisfy the validity rules R1–R5 of 11 November 2026 (Graph Definition §13.3). This revised concept is not yet implemented in the runtime; see also Learning Unit Progression.

Goal placement

DE: Zielplatzierung — see View Projection and Goal Placement

An entry attaching an unchanged goal to a program unit under a reviewed context (`primary`, `secondary`, `assessed`). A placement never re-parents the content `contains` tree and never creates a duplicate occurrence in the default tree.

Competency axis

DE: Kompetenzachse — see View Projection and Goal Placement

An orthogonal taxonomy view built from `competencyCatalog` plus `competencyRefs`, for example process competencies `K1`–`K6`. Broad capability families are taxonomy entries, neither goals nor program units.

Composition view

DE: Composition View, Sichtkomposition — see View Projection and Goal Placement

An explicit, reviewed definition of the learner-facing upper tree for one resolved scope: structure nodes, labels, ordering, and references to canonical subtree roots. It contains neither newly authored atomic goals nor new `requires` edges and—where present—is the authoritative source for that scope's default tree.

Canonical subtree reference

DE: kanonische Teilbaumreferenz — see View Projection and Goal Placement

A pointer from a composition view to a reviewed cluster root of the canonical graph; the subtree contents stay in the canonical model. Within one compiled view tree, expanded references must be pairwise disjoint.

Projection role

DE: Projektionsrolle — see View Projection and Goal Placement

Authored per composition-view reference as `projectionRole` (defaulting to `target`):

- `target` — a selectable learning target in this scope; counts toward navigation, frontier, progress, and completion.
- `prerequisiteOnly` — a supporting goal whose global mastery may unlock or block targets but which is not itself selectable here.

Roles are never inferred implicitly from `requires`, `phase`, year, or stage. On overlap, specificity decides (direct `goalEntry` beats subtree role, deeper subtree beats broader subtree; only at equal specificity does `target` win).

Applicability

DE: Gültigkeit, Anwendbarkeit — see Node Types and Context & Views

Optional compiled goal metadata mapping filter dimensions (`jurisdiction`, `schoolForm`, `stage`, `durationModel`, `courseProfile`, …) to allowed values. An omitted dimension is unrestricted. Applicability is not a node type and never creates parent edges; it only filters visibility in scoped views.

Applicability override

DE: Gültigkeits-Override — see Context & Views

Audit metadata under `extendedData.applicabilityOverrides` recording which active applicability values were added via an explicit, justified exception rather than ordinary source alignment. At runtime, compiled `applicability` remains authoritative.

Entry scope

DE: Einstiegskontext — see General Goal System and Migration

The small set of contextual choices made before navigating content: school form, stage, jurisdiction, duration model, course profile, current year or phase. Entry scope acts as a query and navigation filter and never forces goal duplication.

Resolved scope

DE: aufgelöster Kontext — see View Projection and Goal Placement

A fully determined combination such as `DE-HE / Gymnasium / SekII / Mathematik / LK`. Default trees, single-occurrence validity, and coverage checks are always evaluated relative to one resolved scope.

Default tree

DE: Standardbaum — see View Projection and Goal Placement

The single tree a learner sees for a resolved scope. Every visible edge comes from exactly one explicit source (`contains`, `parentUnitId`, `goalPlacements`, or `competencyRefs`), never from title or `phase` heuristics. **Single-occurrence rule:** within one resolved scope each visible goal appears at most once and under at most one visible parent.

Content tree, program tree, competency view

DE: Inhaltsbaum, Programmbaum, Kompetenzsicht — see View Projection and Goal Placement

The three classic view families, each with one structural source of truth (`contains` for content, `programUnits` + `goalPlacements` for program, `competencyCatalog` + `competencyRefs` for competency axes), complemented by the scope-specific composition view as the preferred learner-facing default.

`phase`

DE: Phase — see Graph Definition

Compatibility metadata (`E`, `Q1`–`Q4`, …) used for badges, filtering, or migration tooling. `phase` is explicitly **not** canonical graph semantics and never implies parentage.

`ALL`

DE: `ALL`-Platzhalter — see Context & Views

A query-only sentinel meaning ”do not restrict this dimension” during scope resolution. `ALL` must never be serialized as an applicability or placement value.

5 Learner runtime

Levels of personalization

DE: Personalisierungsstufen — see Levels of Personalization

The four-step narrowing from curriculum to individual mastery: **1** base curriculum → **2** committed personal curriculum → **3a** current focus plus **3b** one active atomic goal → **4** global mastery. Catalog navigation such as school, university, or languages precedes Level 1 and is not learner scope.

Base curriculum

DE: Basiscurriculum — see Learning Workflow

The skill landscape chosen before learning starts; it provides the full goal universe for that learner.

Personal curriculum

DE: persönliches Curriculum, persönlicher Lehrplan — see Learning Workflow

The committed, longer-lived subset of the base curriculum in scope for one learner (jurisdiction or canonical view, applicable duration model, stage, subjects, and course profiles per subject). It resolves the learner-facing composition view and becomes the candidate space for focus, frontier, progress, and completion, while the backend may still consult the full graph for strict prerequisite checks.

This state is backend-owned and configured exclusively through the first-party WebGUI and Cockpit; the chat coach neither asks for nor mutates it.

Personalization

DE: Personalisierung — see Behavioral Integration of the MCP Coach

The learner's authored validity choices such as jurisdiction or course profile. Personalization is never inferred from `contains` or `requires`. Selectable options always come from the latest backend state as opaque IDs. Course profiles apply per subject so Mathematik LK and Physik GK can coexist.

Learning scope and focus

DE: Lernumfang und Fokus — see Behavioral Integration of the MCP Coach

Two levels the coach must keep apart:

- **Learning scope:** the committed Personal Curriculum for this learner.
- **Focus (Level 3a):** the currently selected subtree or learning corridor (`plannedGoals`) from which frontier and goal options are drawn. Once the Personal Curriculum is complete, focus is never empty: without a valid explicit selection, the highest first visible root is chosen deterministically. A year or phase may bound focus without becoming a goal.

Frontier and goal recommendations are offered only once scope and focus are resolved.

Active goal

DE: aktives Ziel — see Learning Workflow

Exactly one backend-confirmed atomic goal (**Level 3b**) currently worked on in UI and coach. Normally drawn from the frontier; for a non-frontier selection, the system diagnoses missing prerequisites rather than silently bypassing them.

Frontier

DE: Frontier, nächste erreichbare Lernziele — see Graph Definition

The set of not-yet-mastered **atomic** goals whose effective prerequisites are all satisfied — the didactic zone of proximal development, deterministically computed from the graph rather than guessed by the AI.

Satisfaction

DE: Erfüllung — see Graph Definition

$\mathrm{Sat}(g)$ holds iff the atomic basis of g is non-empty and fully mastered. An atomic goal is satisfied iff it is mastered; a cluster iff all its atomic descendants are mastered.

Optimistic mode

DE: optimistischer Modus — see Context & Views

Filter to the selected view scope first (including explicit `prerequisiteOnly` foundations), then evaluate prerequisites. Learners can start directly in their chosen scope without being globally blocked by gaps from earlier years. This is the default mode.

Strict mode (pessimistic mode)

DE: strikter Modus, pessimistischer Modus — see Context & Views

Candidates still come from the filtered scope, but prerequisites are enforced globally across the entire graph. Used diagnostically to surface missing foundations outside the active filter (`strict Mode`).

Missing prerequisites

DE: fehlende Voraussetzungen — see Context & Views

The unsatisfied effective prerequisites of a goal, split into gaps inside and outside the active filter — the diagnostic answer to ”why is this goal still locked?”.

Mastery

DE: Beherrschung; Lernziel geschafft oder noch nicht geschafft — see Learning Workflow and Implicit Mastery

The single stored outcome for an atomic goal: **mastered or not mastered** (encodable as `0` and `1`, without intermediate partial grades or extra qualification tiers). Mastery belongs to stable goal IDs and applies globally across composition views. Cluster or subject progress is the weighted share of unique mastered atomic goals.

The binding 4 November 2026 concept adds **effective mastery** $M_{\text{eff}} = M \cup I$, where I is derived recognition of ordinary competencies through independent application in valid local assessment contracts. Exam and memory goals never enter I. Runtime adaptation to this extended contract remains pending.

Curriculum timeline and pacing plan

DE: Curriculum-Zeitachse, Curriculum-Zeitplan, Stoffverteilungsplan — see Curriculum Timeline, Buffer Planning, and Target-vs-Actual Pacing

A versioned execution-plan overlay assigning resolved curriculum clusters to concrete local date ranges and teaching capacity for a class or learner. It owns calendar, buffer, plan blocks, and revisions without ever writing dates into canonical goals, program units, placements, or composition views.

Planning unit

DE: Planungseinheit — see Curriculum Timeline, Buffer Planning, and Target-vs-Actual Pacing

A plan-local unit used to distribute expected curriculum progress across available time capacity (transparent baseline: one unit per schedulable atomic target). It is neither a grade nor canonical `weight`; baseline mastery before plan start does not count as newly learned planning units.

Plan milestone

DE: Planmeilenstein, konkretes Ziel bis Datum — see Curriculum Timeline, Buffer Planning, and Target-vs-Actual Pacing

A dated commitment such as ”complete the selected Mathematics Abitur tasks by 15 March”. It separates subject-matter kind from deadline rigidity and scheduled workload from mastery-derived completion, driving dated goal lists, forecasts, and replanning without creating mastery on its own.

Course plan status snapshot

DE: Kurs-Planlage, Plan-und-Planlage-Seite — see Curriculum Timeline, Buffer Planning, and Target-vs-Actual Pacing

A digest-bound as-of read model of one plan revision showing textual course plan, planned block progress, confirmed teaching coverage, and separately labeled mastery distribution with input-data quality. It introduces no new source of truth alongside plan and mastery data.

Learning velocity

DE: Lerngeschwindigkeit — see Learning Velocity

The observed rate of atomic goals becoming mastered over time. The Cockpit chart counts recorded completions per calendar week; a bound pacing plan compares newly achieved planning units per effective teaching week against its target curve. Neither form is a grade or learner ranking.

Autopilot

DE: Autopilot — see Frontier Goal Selection Strategies & Autopilot

An opt-in mode that automatically activates the first entry of the strategy-ordered next-steps list once the active goal reaches mastered state.

Selection strategy

DE: Auswahlpriorisierung — see Frontier Goal Selection Strategies & Autopilot

How frontier goals are ordered: `Random` for variety across topic areas, `Sequential` for linear progression along the curriculum structure. Stored per learner alongside autopilot and `strict Mode`.

6 Coach, session, and identity

SkillPilot ID

DE: SkillPilot-ID — see Quickstart

The learner's permanent, pseudonymous key to their learning profile — requiring no registration with name or email address. The permanent SkillPilot ID stays strictly backend-only and never appears as a tool argument, tool result, or chat content.

Learning session

DE: Lernsession — see OAuth and Learner Session Architecture

An opaque, randomly generated key with an absolute 24-hour lifetime addressing the exact learner context confirmed by **Start learning** (*Lernen starten*). Every start creates a fresh session; the adapter passes it unchanged on every learner-specific tool call.

OAuth app binding

DE: OAuth-Verbindung — see OpenAI MCP Client Binding

The revocable OAuth authorization authenticating an approved provider integration against the backend. It authenticates the **integration**, not the learner: OAuth alone neither selects a learner profile nor creates a learning session.

SkillPilot Lerncoach

DE: SkillPilot Lerncoach — see ChatClient/Backend Communication Contract

The AI-facing dialogue layer turning backend state into a learning conversation. The coach leads the dialogue; the backend decides authoritatively on learning state, active filters, allowed transitions, and reachable next steps.

Cockpit

DE: Cockpit — see Quickstart

The SkillPilot web surface displaying learning map, progress, learning-plan status, and SRS flashcard practice. Cockpit and coach operate on the same backend learning state; the unaided recall test (*Verified Recall*) is graded through the coach flow rather than the Cockpit drill.

Safe state projection

DE: sichere Zustandsprojektion — see ChatClient/Backend Communication Contract

The rule that only an explicitly allowlisted, data-minimized projection of learner state crosses the provider boundary. Internal identities, secrets, and private file paths are stripped before anything reaches model, chat, or widget.

Provider-neutral boundary

DE: providerneutrale Grenze — see ChatClient/Backend Communication Contract

The strict separation between the shared SkillPilot domain core and provider-specific adapters. Domain rules, exam authorization, and state projection live in the shared core so adding or replacing an AI provider never forks learning logic.

7 Quality, review, and governance

Curriculum Champion

DE: Curriculum-Champion — see Champion Guide

A subject specialist taking responsibility for the practical trial of a curriculum or a clearly scoped topic area: working through goals as a learner, reporting stumbling blocks directly on the goal or via GitHub, and supporting subject-matter refinement.

Four-level champion model

DE: Vier-Stufen-Champion-Modell — see Four-Level Champion Model

The QA cascade separating machine verification from human trials: `1a` the automated CI pipeline (active today), `1b` static AI review agent and `1c` dynamic AI agent, and `2` independent human Champion trials.

QA scope

DE: QA-Scope — see Curriculum Quality

An explicitly configured check profile covering one learner-facing curriculum section, defined by curriculum, label (such as `Sekundarstufe I`), motivation anchors, selected atomic goals, terminal autonomy areas, and cluster selector. A QA scope counts registered check profiles — neither paths in the graph nor exercise counts. The earlier label ”Routen” has been retired.

CQR rule

DE: CQR-Regel — see Curriculum Quality

A *Curriculum-Quality-Review* rule with a stable ID evaluated by the quality dashboard, for example `CQR-003` for bidirectional jurisdiction coverage or `CQR-101` for closed routes. CQR rules judge curriculum content quality.

GVR rule

DE: GVR-Regel — see Graph Validation Rules

A *Graph-Validation-Rollout* rule with a stable ID enforced by the CI validator, for example `GVR-001` (no goal may require its own `contains` ancestor). GVR rules enforce structural graph integrity.

Maturity level

DE: Reifegrad — see Curriculum Quality and Quality and Human-Trial Concept

A cumulative, conservative rating (`M0`–`M7`) bound to an explicitly named scope and content revision. A single QA scope reaches at most `M3`; a curriculum can reach `M0` through `M7`. `M5` marks core automated readiness (sources captured, jurisdiction views proven bidirectionally, mandatory QA scopes route- and exam-complete). `M6` adds the reviewed memory layer; `M7` is the fully completed machine QA level across all five per-goal gates (`D` descriptions, `P` understanding profiles, `A` atomicity, `M` memory, `V` visualizations).

Human trials by Curriculum Champions are reported as a separate attribute (not `M8`); `M7` alone does not certify human approval or classroom effectiveness.

Semantic atomicity

DE: semantische Atomarität — see Semantic Atomicity Review

The subject-matter review question of whether a learning goal truly expresses a single assessable competence rather than bundling multiple skills. Structural atomicity (no `contains` children) is necessary but not sufficient.

Generated artifact

DE: generiertes Artefakt — see Documentation Guidelines

A report, status snapshot, or documentation output produced by a script and carrying an explicit notice naming its generator, regeneration command, and source of truth. Substantive decisions belong in the canonical source files, never as manual edits to the generated artifact.

8 Distinctions that are easy to get wrong

These are not the same	Why it matters
`contains` vs `requires`	Containment bundles content hierarchically; `requires` sequences prerequisites. Only `requires` drives the frontier.
`contains` vs `goalPlacement`	Content composition vs program membership. A placement never re-parents the content tree.
`requires` vs effective prerequisites	`requires` records direct edges; effective prerequisites also include prerequisites inherited from ancestors during migration.
Atomic goal vs cluster goal	Mastery, frontier, and coverage counts are defined on atomic goals; clusters only aggregate.
Structural vs semantic atomicity	A leaf node without `contains` children can still bundle multiple competences.
`applicability` vs `projectionRole`	`applicability` decides *whether* a goal is visible in a scope; `projectionRole` decides whether it is a `target` or `prerequisiteOnly`.
`applicability` vs node type	`applicability` is contextual view metadata, not a fourth learning mode.
Entry scope vs personal curriculum	Entry scope narrows a query; the personal curriculum is the learner's durably committed scope.

These are not the same	Why it matters
Learning scope vs focus	Scope defines what is relevant overall; focus selects the active working corridor.
Frontier vs recommendation	The frontier is computed mathematically from graph and mastery; selection strategy and autopilot only order within it.
Optimistic vs strict mode	Same candidate goals, different prerequisite reach — strict mode is a diagnostic, not a stricter curriculum.
SRS practice vs verified recall	Practice schedules review intervals; only the unaided coach recall check closes a memorization goal.
SkillPilot ID vs learning session	Permanent backend-only pseudonymous identity vs fresh 24-hour key sent to the provider integration.
OAuth binding vs learning session	OAuth authenticates the app/integration; the learning session selects the learner context.
`phase` vs program unit	`phase` is compatibility metadata; only an explicit program unit forms a structural node.
Learning unit vs personalization level	A recursively composed educational whole with derived completion vs one of the four scope/state layers.
Learning unit vs goal difficulty	A unit of acquisition and completion vs the demand level of one goal (`level`).
Unit completion vs goal availability	Fulfilled child and own obligations vs prerequisites for starting work; a technical whole-unit gate requires a separate policy.
QA scope vs learning route	A QA scope is a registered check profile, not a single path walked by a learner.
CQR vs GVR rules	Content quality review vs structural graph validation.
Curriculum vs skill graph	Normative external document vs versioned operative graph model derived from it.
Skill graph vs skill landscape	Formal model and semantics vs one delimited, published subject instance with `landscapeId`, Champion, and maturity level.

Maintenance

- `cd app && npm run check:terminology` rejects retired synonyms; add a rule there whenever a term is consolidated here.
- Add a term here once it is used across documents and its meaning is not immediately obvious from the code.
- Keep each entry to a concise definition plus delimitation; detailed rules belong in the linked normative document.
- If a summary here and a linked normative document ever disagree, the normative document wins and this entry must be updated.