EduSphere API Contracts — GraphQL Federation Schema

> **Reference document for Claude Code and all developers.** Every subgraph, entity, query, mutation, subscription, input type, and directive is defined here. Do NOT deviate from these definitions without updating this document first. This document is the **single source of truth** for the EduSphere supergraph API surface.

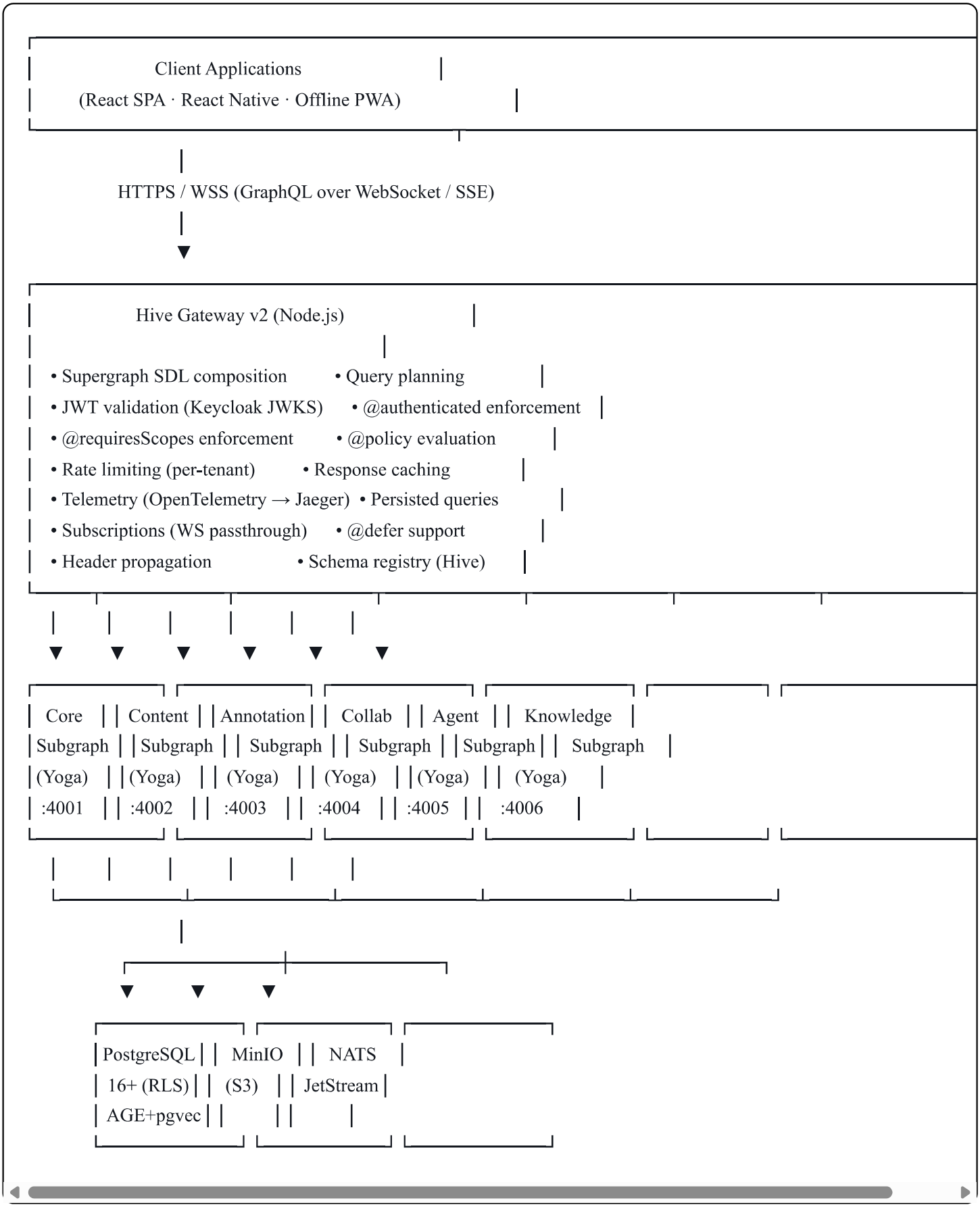
Table of Contents

---

1. Federation Architecture Overview

EduSphere uses **GraphQL Federation v2.7+** (Apollo Federation v2 spec compatible) with **Hive Gateway v2** (MIT-licensed) as the supergraph gateway. Each subgraph is a standalone NestJS service backed by **GraphQL Yoga** with `YogaFederationDriver` from `@graphql-yoga/nestjs`.

> **Why Hive Gateway over Apollo Router?** Apollo Router uses the Elastic License v2 (ELv2), which restricts offering it as a managed service. Hive Gateway v2 is fully MIT-licensed, supports the same Federation v2.7 spec, and is nearly 2x faster than competitors among JS-based gateways. It runs on all runtimes (Node, Deno, Bun, Cloudflare Workers). See EduSphere Architecture Guide §5 for details.



Key Decisions:

- **Schema-first design**: SDL files are the source of truth; resolvers implement the contract

- **MIT-licensed gateway**: Hive Gateway v2 (MIT) over Apollo Router (ELv2) — full Federation v2 spec compatibility
- **GraphQL Yoga subgraphs**: Each NestJS service uses `@graphql-yoga/nestjs` with `YogaFederationDriver` for federation
- **Transport-based multi-tenancy**: JWT carries `tenant_id`; gateway propagates `x-tenant-id` header to subgraphs; RLS enforces at database level
- **Relay Cursor Connections**: all list fields use the Relay connection spec for forward/backward cursor pagination
- **Demand-oriented schema**: types are shaped for client needs, not database tables
- **Entity ownership**: each entity is owned by exactly one subgraph; other subgraphs extend with `@key` stubs
- **Soft deletes hidden**: `deleted_at` is never exposed; filtered out in resolvers
- **Three-layer AI architecture**: Vercel AI SDK (LLM abstraction) → LangGraph.js (agent orchestration) → LlamaIndex.TS (RAG/knowledge)
- **HybridRAG pattern**: pgvector semantic search + Apache AGE graph traversal fused before LLM generation
- **Hive Schema Registry**: Breaking change detection and schema evolution tracking via GraphQL Hive

---

2. Subgraph Decomposition Map

Subgraph	Port	Owned Entities	Extended Entities	Domain
Core	4001	`Tenant`, `User`	—	Identity, tenancy, auth
Content	4002	`Course`, `Module`, `MediaAsset`, `Transcript`, `TranscriptSegment`	`User` (createdCourses), `Tenant`	Courses, media pipeline, transcription
Annotation	4003	`Annotation`	`User`, `MediaAsset`	Markings, sketches, spatial comments
Collaboration	4004	`CollabDocument`, `CollabSession`	`User`, `Annotation`	CRDT persistence, real-time presence
Agent	4005	`AgentDefinition`, `AgentExecution`	`User`, `Annotation`, `Course`	AI agents, templates, executions

Subgraph	Port	Owned Entities	Extended Entities	Domain
Knowledge	4006	`Concept`, `Person`, `Term`, `Source`, `TopicCluster`, `KnowledgeRelation`, `SemanticSearchResult`	`MediaAsset`, `TranscriptSegment`, `Annotation`	Knowledge graph, embeddings, search

Entity Ownership Rules

Entity	Owning Subgraph	@key fields
Tenant	Core	id
User	Core	id
Course	Content	id
Module	Content	id
MediaAsset	Content	id
Transcript	Content	id
TranscriptSegment	Content	id
Annotation	Annotation	id
CollabDocument	Collaboration	id
CollabSession	Collaboration	id
AgentDefinition	Agent	id
AgentExecution	Agent	id
Concept	Knowledge	id
Person	Knowledge	id
Term	Knowledge	id
Source	Knowledge	id
TopicCluster	Knowledge	id

---

3. Shared Types & Scalars

All subgraphs MUST import the following shared definitions. These are declared in a shared package `@edusphere/graphql-shared` and included via build-time codegen.

```graphql
```

```graphql
# ——— File: packages/graphql-shared/src/scalars.graphql ———

"""ISO 8601 date-time string with timezone (e.g. "2025-01-15T10:30:00Z")"""
scalar DateTime

"""UUID v4 string (e.g. "550e8400-e29b-41d4-a716-446655440000")"""
scalar UUID

"""Arbitrary JSON object"""
scalar JSON

"""Arbitrary JSON object (alias for interoperability)"""
scalar JSONObject

"""Base64-encoded binary data"""
scalar Base64

"""
Opaque cursor string for Relay-style pagination.
Clients MUST NOT parse or construct cursors — treat as opaque tokens.
"""
scalar Cursor

"""URL string (validated as a valid URL)"""
scalar URL

"""Email address string (validated)"""
scalar EmailAddress

"""Non-negative integer (≥ 0)"""
scalar NonNegativeInt

"""Positive integer (> 0)"""
scalar PositiveInt

"""Float between 0.0 and 1.0 inclusive"""
scalar UnitFloat
```

Shared Enums

graphql

```
# ——— File: packages/graphql-shared/src/enums.graphql ———

enum UserRole {
  SUPER_ADMIN
  ORG_ADMIN
  INSTRUCTOR
  STUDENT
  RESEARCHER
}

enum MediaType {
  VIDEO
  AUDIO
  PDF
  IMAGE
  DOCUMENT
}

enum TranscriptionStatus {
  PENDING
  PROCESSING
  COMPLETED
  FAILED
}

enum AnnotationType {
  TEXT
  SKETCH
  LINK
  BOOKMARK
  SPATIAL_COMMENT
}

enum AnnotationLayer {
  PERSONAL
  SHARED
  INSTRUCTOR
  AI_GENERATED
}

enum AgentTemplate {
  CHAVRUTA
  SUMMARIZER
  QUIZ_MASTER
  RESEARCH_SCOUT
  EXPLAINER
```

```
  CUSTOM
}

enum AgentTrigger {
  MANUAL
  ON_ANNOTATION
  ON_NEW_CONTENT
  SCHEDULED
}

enum AgentOutputFormat {
  CHAT
  OVERLAY
  SIDE_PANEL
  NOTIFICATION
}

enum ExecutionStatus {
  QUEUED
  RUNNING
  COMPLETED
  FAILED
  CANCELLED
}

enum TenantPlan {
  FREE
  STARTER
  PROFESSIONAL
  ENTERPRISE
}

"""Sort direction for ordered queries"""
enum SortDirection {
  ASC
  DESC
}
```

---

4. Pagination & Connection Spec

All list fields MUST use the **Relay Cursor Connections Specification**. No raw lists for user-facing collections.

Connection Types (Generic Pattern)

```graphql
```

```graphql
# ——— File: packages/graphql-shared/src/pagination.graphql ———

"""
Relay-compliant page info for cursor-based pagination.
Used by all Connection types across every subgraph.
"""
type PageInfo {
  """When paginating forwards, are there more items?"""
  hasNextPage: Boolean!
  """When paginating backwards, are there more items?"""
  hasPreviousPage: Boolean!
  """Cursor of the first edge in the current page"""
  startCursor: Cursor
  """Cursor of the last edge in the current page"""
  endCursor: Cursor
}

"""
Standard pagination arguments accepted by all connection fields.
At least one of (first, last) MUST be provided.
first + after = forward pagination.
last + before = backward pagination.
Providing both first AND last is STRONGLY DISCOURAGED.
"""
input ConnectionArgs {
  """Returns the first N edges after the cursor"""
  first: PositiveInt
  """Returns edges after this cursor"""
  after: Cursor
  """Returns the last N edges before the cursor"""
  last: PositiveInt
  """Returns edges before this cursor"""
  before: Cursor
}
```

Naming Convention for Connections

Every entity `Foo` generates:

- `FooConnection` — contains `edges` and `pageInfo`
- `FooEdge` — contains `node` and `cursor`

```graphql
```

```graphql
# Example: CourseConnection
type CourseConnection {
  edges: [CourseEdge!]!
  pageInfo: PageInfo!
  """Total count of items matching filters (computed only if requested)"""
  totalCount: NonNegativeInt
}

type CourseEdge {
  node: Course!
  cursor: Cursor!
}
```

Pagination Limits

Context	Default `first`	Max `first`/`last`
Standard lists	20	100
Transcript segments	50	500
Search results	10	50
Agent executions	20	100
Knowledge graph relations	20	200

If `first`/`last` exceeds the maximum, the server clamps to max and returns a `PAGINATION_CLAMPED` extension warning.

---

5. Authentication & Authorization Directives

Federation v2 Auth Directives

```graphql
```

```
# ——— Applied at the gateway level (Hive Gateway config) ———

"""
Requires a valid JWT. Unauthenticated requests receive 401.
Applied to all fields by default; opt-out with @public.
"""
directive @authenticated on FIELD_DEFINITION | OBJECT | INTERFACE | SCALAR | ENUM

"""
Requires specific OAuth2 scopes from the JWT.
Scopes use AND within inner array, OR across outer array.
"""
directive @requiresScopes(
  scopes: [[String!]!]!
) on FIELD_DEFINITION | OBJECT | INTERFACE | SCALAR | ENUM

"""
Custom authorization policy evaluated by the router.
Policies are defined in router.yaml and evaluated against JWT claims.
"""
directive @policy(
  policies: [[String!]!]!
) on FIELD_DEFINITION | OBJECT | INTERFACE | SCALAR | ENUM
```

Custom Application Directives

```
graphql
```

```graphql
# ——— File: packages/graphql-shared/src/directives.graphql ———

"""
Marks a field or type as publicly accessible without authentication.
Overrides the default @authenticated requirement.
"""
directive @public on FIELD_DEFINITION | OBJECT

"""
Requires the caller to have one of the specified roles.
Evaluated in subgraph resolvers using context.user.role.
"""
directive @requiresRole(
  roles: [UserRole!]!
) on FIELD_DEFINITION | OBJECT

"""
Marks a field as only accessible by the resource owner.
The resolver checks context.user.id against the entity's userId/creatorId.
"""
directive @ownerOnly on FIELD_DEFINITION

"""
Rate-limits a field to N calls per window per user.
Evaluated at the router level via the rate-limiter plugin.
"""
directive @rateLimit(
  max: Int!
  window: String!  # e.g., "1m", "1h", "1d"
) on FIELD_DEFINITION

"""
Marks a field as deprecated with a reason and optional removal date.
Clients using deprecated fields will see warnings in tooling.
"""
directive @deprecated(
  reason: String!
  removalDate: String  # ISO 8601 date
) on FIELD_DEFINITION | ENUM_VALUE | ARGUMENT_DEFINITION | INPUT_FIELD_DEFINITION
```

Role-to-Scope Mapping

Role	JWT Scopes	Capabilities
SUPER_ADMIN	admin:*	Full system access, cross-tenant

Role	JWT Scopes	Capabilities
ORG_ADMIN	org:manage, org:users, org:billing	Manage tenant settings, users, billing
INSTRUCTOR	course:write, annotation:write, agent:write, media:upload	Create/edit courses, annotations, agents
STUDENT	course:read, annotation:write:own, agent:execute	View courses, create own annotations, run agents
RESEARCHER	course:read, annotation:read, knowledge:read, knowledge:write	Read content, query knowledge graph, add concepts

JWT Claims Structure (Propagated by Router)

typescript

```typescript
// Extracted from Keycloak JWT and propagated as headers
interface JWTClaims {
  sub: string;         // Keycloak user ID
  tenant_id: string;   // UUID of the tenant
  user_id: string;     // EduSphere internal user UUID
  email: string;
  roles: UserRole[];
  scopes: string[];
  iat: number;
  exp: number;
}
```

Header Propagation (Router → Subgraphs)

yaml

```
# Headers the router propagates to all subgraph requests
headers:
  all:
    request:
      - propagate:
          named: "x-tenant-id"
      - propagate:
          named: "x-user-id"
      - propagate:
          named: "x-user-role"
      - propagate:
          named: "x-user-scopes"
      - propagate:
          named: "authorization"
      - propagate:
          named: "x-request-id"
      - propagate:
          named: "x-correlation-id"
```

---

6. Error Handling Contract

Standard Error Format

All subgraphs MUST return errors following this structure. The `extensions` field provides machine-readable metadata for client error handling.

```
graphql
```

```
# Conceptual — errors are part of the GraphQL response, not the schema
# Included here as the contract all resolvers MUST follow

"""
Standard error extension fields returned in every GraphQL error.
"""
type ErrorExtension {
  """Machine-readable error code (UPPER_SNAKE_CASE)"""
  code: String!
  """HTTP-equivalent status code"""
  status: Int!
  """ISO 8601 timestamp of the error"""
  timestamp: DateTime!
  """Correlation ID for tracing"""
  requestId: String!
  """Field path that caused the error"""
  path: [String!]
  """Additional context for debugging (only in development)"""
  debug: JSON
}
```

Error Codes Catalog

Code	HTTP Status	Description	Retryable
`UNAUTHENTICATED`	401	Missing or invalid JWT	No
`FORBIDDEN`	403	Valid JWT but insufficient permissions	No
`NOT_FOUND`	404	Requested resource does not exist	No
`CONFLICT`	409	Resource version conflict (optimistic locking)	Yes
`VALIDATION_ERROR`	400	Input validation failure	No
`RATE_LIMITED`	429	Too many requests	Yes (after `retryAfter`)
`TENANT_QUOTA_EXCEEDED`	403	Tenant has exceeded plan limits	No
`MEDIA_PROCESSING_ERROR`	500	Media transcoding/transcription failure	Yes
`AGENT_EXECUTION_ERROR`	500	Agent runtime failure	Yes
`GRAPH_QUERY_ERROR`	500	Apache AGE query failure	Yes

Code	HTTP Status	Description	Retryable
`EMBEDDING_ERROR`	500	Vector embedding generation failure	Yes
`INTERNAL_ERROR`	500	Unhandled server error	Yes
`PAGINATION_CLAMPED`	200	Requested page size was clamped (warning)	N/A
`OPTIMISTIC_LOCK_FAILED`	409	Version mismatch on update	Yes

Error Response Example

```json
{
  "data": null,
  "errors": [
    {
      "message": "Course not found",
      "locations": [{ "line": 2, "column": 3 }],
      "path": ["course"],
      "extensions": {
        "code": "NOT_FOUND",
        "status": 404,
        "timestamp": "2025-06-15T10:30:00Z",
        "requestId": "req_abc123def456"
      }
    }
  ]
}
```

Partial Success Pattern

For mutations that operate on multiple items, subgraphs MUST return a `MutationResult` union indicating per-item success/failure:

```graphql
```

```
type MutationSuccess {
  message: String
}

type MutationError {
  message: String!
  code: String!
  field: String
}

union MutationResult = MutationSuccess | MutationError
```

---

7. Subgraph: Core (Tenants & Users)

Owns: `Tenant`, `User` **Port:** 4001 **Database tables:** `tenants`, `users`

Schema Definition

> **Note:** The `@link(url: "https://specs.apollo.dev/federation/v2.7" ...)` directive imports the Federation v2 specification, which is an open standard. Hive Gateway v2 fully supports this spec — the `specs.apollo.dev` URL is the canonical spec location, not a dependency on Apollo products.

graphql

```
# ——— File: apps/subgraph-core/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: [
      "@key",
      "@shareable",
      "@inaccessible",
      "@authenticated",
      "@requiresScopes",
      "@policy",
      "@tag",
      "@override"
    ])

# ═══════════════════════════════════════════════════════════════
#  Tenant
# ═══════════════════════════════════════════════════════════════

"""
Top-level organizational unit. All data is tenant-isolated.
Each tenant represents an educational institution, organization, or team.
"""
type Tenant @key(fields: "id") {
  id: UUID!
  """Human-readable name of the organization"""
  name: String!
  """URL-friendly unique slug (e.g., "bar-ilan-university")"""
  slug: String!
  """Current subscription plan"""
  plan: TenantPlan!
  """Whether the tenant is active (deactivated tenants cannot access the system)"""
  isActive: Boolean!
  """Maximum number of users allowed under current plan"""
  maxUsers: NonNegativeInt!
  """Tenant-specific settings and branding"""
  settings: TenantSettings
  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Resolved by other subgraphs ———
  # users: UserConnection!        → resolved by Core itself
  # courses: CourseConnection!    → resolved by Content subgraph
}

type TenantSettings {
```

```
  defaultLanguage: String
  allowedLlmProviders: [String!]
  maxStorageGb: NonNegativeInt
  customOntologyFields: [OntologyFieldDef!]
  brandingColors: BrandingColors
}

type OntologyFieldDef {
  name: String!
  type: String!
}

type BrandingColors {
  primary: String!
  secondary: String!
}

# ═══════════════════════════════════════════════════════════════
#  User
# ═══════════════════════════════════════════════════════════════

"""
An authenticated user within a tenant.
Users are synced from Keycloak on first login.
"""
type User @key(fields: "id") {
  id: UUID!
  """Email address (unique within tenant)"""
  email: EmailAddress!
  """Display name shown in UI and collaboration features"""
  displayName: String!
  """URL to avatar image"""
  avatarUrl: URL
  """Role within the tenant"""
  role: UserRole!
  """Whether the user can currently access the system"""
  isActive: Boolean!
  """User preferences for UI and behavior"""
  preferences: UserPreferences
  createdAt: DateTime!
  updatedAt: DateTime!

  # ─── Tenant relationship ───
  tenant: Tenant!

  # ─── Resolved by other subgraphs ───
  # annotations: AnnotationConnection!      → Annotation subgraph
```

```
  # agentDefinitions: AgentDefinitionConnection! → Agent subgraph
  # agentExecutions: AgentExecutionConnection!   → Agent subgraph
  # createdCourses: CourseConnection!        → Content subgraph
}

type UserPreferences {
  language: String
  theme: ThemePreference
  defaultAnnotationLayer: AnnotationLayer
  notificationsEnabled: Boolean
  agentDefaults: AgentPreferenceDefaults
}

enum ThemePreference {
  LIGHT
  DARK
  SYSTEM
}

type AgentPreferenceDefaults {
  difficulty: String
  personality: String
}

# ──────────────────────────────────────────────────────────────
#  Queries
# ──────────────────────────────────────────────────────────────

type Query {
  """
  Returns the currently authenticated user.
  This is the primary entry point for client initialization.
  """
  me: User! @authenticated

  """Fetch a user by ID within the current tenant"""
  user(id: UUID!): User @authenticated

  """
  List all users within the current tenant.
  Filterable by role and activity status.
  """
  users(
    first: PositiveInt = 20
    after: Cursor
    last: PositiveInt
    before: Cursor
```

```
    filter: UserFilterInput
    orderBy: UserOrderByInput
  ): UserConnection! @authenticated

  """
  Fetch the current tenant's details.
  """
  currentTenant: Tenant! @authenticated

  """
  Fetch a tenant by slug. Used for SSO login flow to resolve tenant.
  """
  tenantBySlug(slug: String!): Tenant @public
}

# ---------------------------------------------------------------------------
#  Mutations
# ---------------------------------------------------------------------------

type Mutation {
  """
  Update the current user's profile and preferences.
  Users can only update their own profile.
  """
  updateMyProfile(input: UpdateProfileInput!): User! @authenticated

  """
  Update a user's role. Requires ORG_ADMIN or SUPER_ADMIN.
  """
  updateUserRole(
    userId: UUID!
    role: UserRole!
  ): User! @authenticated @requiresScopes(scopes: [["org:users"]])

  """
  Deactivate a user (soft). Requires ORG_ADMIN.
  Deactivated users cannot log in.
  """
  deactivateUser(
    userId: UUID!
  ): User! @authenticated @requiresScopes(scopes: [["org:users"]])

  """
  Reactivate a previously deactivated user.
  """
  reactivateUser(
    userId: UUID!
```

```
  ): User! @authenticated @requiresScopes(scopes: [["org:users"]])

  """
  Update tenant settings. Requires ORG_ADMIN.
  """
  updateTenantSettings(
    input: UpdateTenantSettingsInput!
  ): Tenant! @authenticated @requiresScopes(scopes: [["org:manage"]])
}

# ──────────────────────────────────────────────────────────────
#  Input Types
# ──────────────────────────────────────────────────────────────

input UpdateProfileInput {
  displayName: String
  avatarUrl: URL
  preferences: UserPreferencesInput
}

input UserPreferencesInput {
  language: String
  theme: ThemePreference
  defaultAnnotationLayer: AnnotationLayer
  notificationsEnabled: Boolean
  agentDefaults: AgentPreferenceDefaultsInput
}

input AgentPreferenceDefaultsInput {
  difficulty: String
  personality: String
}

input UpdateTenantSettingsInput {
  defaultLanguage: String
  allowedLlmProviders: [String!]
  maxStorageGb: NonNegativeInt
  brandingColors: BrandingColorsInput
  customOntologyFields: [OntologyFieldDefInput!]
}

input BrandingColorsInput {
  primary: String!
  secondary: String!
}

input OntologyFieldDefInput {
```

```
  name: String!
  type: String!
}

input UserFilterInput {
  """Filter by role"""
  role: UserRole
  """Filter by active status"""
  isActive: Boolean
  """Full-text search on displayName and email"""
  search: String
}

input UserOrderByInput {
  field: UserSortField!
  direction: SortDirection!
}

enum UserSortField {
  DISPLAY_NAME
  EMAIL
  CREATED_AT
  ROLE
}

# ──────────────────────────────────────────────────────────────
#  Connection Types
# ──────────────────────────────────────────────────────────────

type UserConnection {
  edges: [UserEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type UserEdge {
  node: User!
  cursor: Cursor!
}
```

---

8. Subgraph: Content (Courses, Media, Transcripts)

Owns: `Course`, `Module`, `MediaAsset`, `Transcript`, `TranscriptSegment` **Port:** 4002 **Database tables:** `courses`, `modules`, `media_assets`, `transcripts`, `transcript_segments`

Schema Definition

graphql

```
# ——— File: apps/subgraph-content/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: [
      "@key", "@shareable", "@external", "@provides",
      "@requires", "@authenticated", "@requiresScopes", "@tag"
    ])

# ——————————————————————————————————————————————
#  Entity Stubs (owned by other subgraphs)
# ——————————————————————————————————————————————

type User @key(fields: "id", resolvable: false) {
  id: UUID!
}

type Tenant @key(fields: "id", resolvable: false) {
  id: UUID!
}

# ——————————————————————————————————————————————
#  Extend User with content fields
# ——————————————————————————————————————————————

extend type User {
  """Courses created by this user (instructors/admins only)"""
  createdCourses(
    first: PositiveInt = 20
    after: Cursor
    filter: CourseFilterInput
  ): CourseConnection!
}

# ——————————————————————————————————————————————
#  Course
# ——————————————————————————————————————————————

"""
A course is the primary organizational unit for content.
Courses contain modules, which contain media assets.
Courses support versioning and can be forked.
"""
type Course @key(fields: "id") {
  id: UUID!
  title: String!
```

```
  description: String
  """Structured syllabus as a JSON tree"""
  syllabusJson: JSON
  """Monotonically increasing version number"""
  version: NonNegativeInt!
  """Whether the course is visible to students"""
  isPublished: Boolean!
  """Tags for categorization and search"""
  tags: [String!]!
  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Relationships ———
  """The user who created this course"""
  creator: User!
  """The tenant this course belongs to"""
  tenant: Tenant!
  """If this course was forked, the original course"""
  forkedFrom: Course
  """Ordered list of modules in this course"""
  modules(
    first: PositiveInt = 50
    after: Cursor
    orderBy: ModuleOrderByInput
  ): ModuleConnection!
}

# ═══════════════════════════════════════════════════════════════════
#  Module
# ═══════════════════════════════════════════════════════════════════

"""
A subdivision within a course. Modules contain media assets
and are ordered by orderIndex.
"""
type Module @key(fields: "id") {
  id: UUID!
  title: String!
  description: String
  """Position within the course (0-indexed)"""
  orderIndex: NonNegativeInt!
  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Relationships ———
  course: Course!
  """Media assets attached to this module"""
```

```
  mediaAssets(
    first: PositiveInt = 20
    after: Cursor
    filter: MediaAssetFilterInput
  ): MediaAssetConnection!
}

# ──────────────────────────────────────────────────────────────
#  MediaAsset
# ──────────────────────────────────────────────────────────────

"""
A media file (video, audio, PDF, image, document) uploaded to the system.
Videos are transcoded to HLS; audio/video files are transcribed.
"""
type MediaAsset @key(fields: "id") {
  id: UUID!
  type: MediaType!
  title: String!
  description: String
  """Original filename as uploaded"""
  originalFilename: String
  """MIME type of the original file"""
  mimeType: String
  """File size in bytes"""
  fileSizeBytes: NonNegativeInt
  """Duration in seconds (video/audio only)"""
  durationSeconds: Float
  """Current transcription status"""
  transcriptionStatus: TranscriptionStatus
  """Presigned URL to the HLS manifest for video playback"""
  hlsManifestUrl: URL
  """Presigned URL to the thumbnail image"""
  thumbnailUrl: URL
  """Presigned URL for direct download"""
  downloadUrl: URL
  """Additional technical metadata"""
  metadata: MediaMetadata
  createdAt: DateTime!
  updatedAt: DateTime!

  # ─── Relationships ───
  """The user who uploaded this asset"""
  uploader: User
  """The module this asset belongs to"""
  module: Module
  """Transcription of this asset (if available)"""
```

```
  transcript: Transcript
  """Time-aligned transcript segments"""
  segments(
    first: PositiveInt = 50
    after: Cursor
    filter: SegmentFilterInput
  ): TranscriptSegmentConnection!
}

type MediaMetadata {
  width: NonNegativeInt
  height: NonNegativeInt
  codec: String
  bitrate: NonNegativeInt
  language: String
}

# ═══════════════════════════════════════════════════════════════
#  Transcript
# ═══════════════════════════════════════════════════════════════

"""
Full transcription of a media asset.
Generated by Whisper or other speech-to-text models.
"""
type Transcript @key(fields: "id") {
  id: UUID!
  """The complete transcription text"""
  fullText: String!
  """ISO 639-1 language code (e.g., "he", "en")"""
  language: String!
  """Model used for transcription (e.g., "whisper-large-v3")"""
  modelUsed: String
  """Average confidence score (0-1)"""
  confidence: UnitFloat
  """Total word count"""
  wordCount: NonNegativeInt
  createdAt: DateTime!

  # ─── Relationships ───
  asset: MediaAsset!
  """Time-aligned segments of this transcript"""
  segments(
    first: PositiveInt = 50
    after: Cursor
  ): TranscriptSegmentConnection!
}
```

```
# ═══════════════════════════════════════════════════════════════
#  TranscriptSegment
# ═══════════════════════════════════════════════════════════════

"""
A time-aligned segment of a transcript.
Used for video synchronization, search, and knowledge graph anchoring.
"""
type TranscriptSegment @key(fields: "id") {
  id: UUID!
  """Start time in seconds from the beginning of the media"""
  startTime: Float!
  """End time in seconds from the beginning of the media"""
  endTime: Float!
  """Transcribed text for this segment"""
  text: String!
  """Speaker name (if speaker diarization was performed)"""
  speaker: String
  """Confidence score for this segment (0-1)"""
  confidence: UnitFloat
  createdAt: DateTime!

  # ─── Relationships ───
  transcript: Transcript!
  asset: MediaAsset!
}

# ═══════════════════════════════════════════════════════════════
#  Queries
# ═══════════════════════════════════════════════════════════════

type Query {
  """Fetch a single course by ID"""
  course(id: UUID!): Course @authenticated

  """
  List courses within the current tenant.
  Students see only published courses; instructors see all.
  """
  courses(
    first: PositiveInt = 20
    after: Cursor
    last: PositiveInt
    before: Cursor
    filter: CourseFilterInput
    orderBy: CourseOrderByInput
```

```
  ): CourseConnection! @authenticated

  """Fetch a single module by ID"""
  module(id: UUID!): Module @authenticated

  """Fetch a single media asset by ID"""
  mediaAsset(id: UUID!): MediaAsset @authenticated

  """
  List media assets within the current tenant.
  Supports filtering by type and transcription status.
  """
  mediaAssets(
    first: PositiveInt = 20
    after: Cursor
    filter: MediaAssetFilterInput
    orderBy: MediaAssetOrderByInput
  ): MediaAssetConnection! @authenticated

  """Fetch transcript segments for a time range within a media asset"""
  segmentsForTimeRange(
    assetId: UUID!
    startTime: Float!
    endTime: Float!
  ): [TranscriptSegment!]! @authenticated

  """Full-text search across all transcripts in the current tenant"""
  searchTranscripts(
    query: String!
    first: PositiveInt = 10
    after: Cursor
    assetIds: [UUID!]
  ): TranscriptSearchResultConnection! @authenticated
}

# ═══════════════════════════════════════════════════════════════
#  Mutations
# ═══════════════════════════════════════════════════════════════

type Mutation {
  # ─── Courses ───

  """Create a new course. Requires INSTRUCTOR or ORG_ADMIN role."""
  createCourse(input: CreateCourseInput!): Course!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Update a course. Only the creator or ORG_ADMIN can update."""
```

```
  updateCourse(id: UUID!, input: UpdateCourseInput!): Course!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Soft-delete a course. Only the creator or ORG_ADMIN can delete."""
  deleteCourse(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Publish or unpublish a course."""
  toggleCoursePublished(id: UUID!, isPublished: Boolean!): Course!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Fork a course (creates a copy linked to the original)."""
  forkCourse(courseId: UUID!): Course!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  # ——— Modules ———

  """Add a module to a course."""
  createModule(input: CreateModuleInput!): Module!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Update a module."""
  updateModule(id: UUID!, input: UpdateModuleInput!): Module!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Delete a module (cascades to media assets)."""
  deleteModule(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Reorder modules within a course."""
  reorderModules(courseId: UUID!, moduleOrder: [UUID!]!): [Module!]!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  # ——— Media Assets ———

  """
  Initiate a media upload. Returns a presigned S3 upload URL.
  The client uploads directly to S3, then calls completeMediaUpload.
  """
  initiateMediaUpload(input: InitiateUploadInput!): UploadTicket!
    @authenticated @requiresScopes(scopes: [["media:upload"]])
    @rateLimit(max: 20, window: "1h")

  """
  Mark an upload as complete. Triggers transcoding/transcription pipelines.
  """
  completeMediaUpload(uploadId: UUID!): MediaAsset!
```

```
    @authenticated @requiresScopes(scopes: [["media:upload"]])

  """Update media asset metadata."""
  updateMediaAsset(id: UUID!, input: UpdateMediaAssetInput!): MediaAsset!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Soft-delete a media asset."""
  deleteMediaAsset(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["course:write"]])

  """Re-trigger transcription for a media asset."""
  retriggerTranscription(assetId: UUID!): MediaAsset!
    @authenticated @requiresScopes(scopes: [["course:write"]])
}

# ─────────────────────────────────────────────────────────────
#  Input Types
# ─────────────────────────────────────────────────────────────

input CreateCourseInput {
  title: String!
  description: String
  syllabusJson: JSON
  tags: [String!]
  isPublished: Boolean = false
}

input UpdateCourseInput {
  title: String
  description: String
  syllabusJson: JSON
  tags: [String!]
  """Used for optimistic locking — must match current version"""
  expectedVersion: NonNegativeInt
}

input CreateModuleInput {
  courseId: UUID!
  title: String!
  description: String
  orderIndex: NonNegativeInt
}

input UpdateModuleInput {
  title: String
  description: String
  orderIndex: NonNegativeInt
```

```
}

input InitiateUploadInput {
  moduleId: UUID!
  type: MediaType!
  title: String!
  description: String
  originalFilename: String!
  mimeType: String!
  fileSizeBytes: NonNegativeInt!
}

input UpdateMediaAssetInput {
  title: String
  description: String
}

input CourseFilterInput {
  """Filter by published status"""
  isPublished: Boolean
  """Filter by creator ID"""
  creatorId: UUID
  """Full-text search on title and description"""
  search: String
  """Filter by tags (ANY match)"""
  tags: [String!]
}

input CourseOrderByInput {
  field: CourseSortField!
  direction: SortDirection!
}

enum CourseSortField {
  TITLE
  CREATED_AT
  UPDATED_AT
  VERSION
}

input MediaAssetFilterInput {
  """Filter by media type"""
  type: MediaType
  """Filter by transcription status"""
  transcriptionStatus: TranscriptionStatus
  """Filter by module"""
  moduleId: UUID
```

```
  """Full-text search on title"""
  search: String
}

input MediaAssetOrderByInput {
  field: MediaAssetSortField!
  direction: SortDirection!
}

enum MediaAssetSortField {
  TITLE
  CREATED_AT
  FILE_SIZE
  DURATION
  TYPE
}

input ModuleOrderByInput {
  field: ModuleSortField!
  direction: SortDirection!
}

enum ModuleSortField {
  ORDER_INDEX
  TITLE
  CREATED_AT
}

input SegmentFilterInput {
  """Filter by speaker name"""
  speaker: String
  """Full-text search within segment text"""
  search: String
  """Minimum confidence threshold"""
  minConfidence: UnitFloat
}

# ---------------------------------------------------------------------------
#  Connection Types
# ---------------------------------------------------------------------------

type CourseConnection {
  edges: [CourseEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}
```

```
type CourseEdge {
  node: Course!
  cursor: Cursor!
}

type ModuleConnection {
  edges: [ModuleEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type ModuleEdge {
  node: Module!
  cursor: Cursor!
}

type MediaAssetConnection {
  edges: [MediaAssetEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type MediaAssetEdge {
  node: MediaAsset!
  cursor: Cursor!
}

type TranscriptSegmentConnection {
  edges: [TranscriptSegmentEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type TranscriptSegmentEdge {
  node: TranscriptSegment!
  cursor: Cursor!
}

# ——— Upload Ticket ———

"""
Returned by initiateMediaUpload. Contains a presigned S3 URL
for the client to upload the file directly.
"""
type UploadTicket {
  """Unique ID for this upload session"""
  uploadId: UUID!
```

```
  """Presigned PUT URL for direct upload to S3"""
  presignedUrl: URL!
  """Expiration time of the presigned URL"""
  expiresAt: DateTime!
  """Required headers to include in the PUT request"""
  requiredHeaders: JSON
}

# ——— Transcript Search ———

type TranscriptSearchResult {
  segment: TranscriptSegment!
  asset: MediaAsset!
  """Relevance score"""
  score: Float!
  """Highlighted text snippet with <mark> tags"""
  highlightedText: String!
}

type TranscriptSearchResultConnection {
  edges: [TranscriptSearchResultEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type TranscriptSearchResultEdge {
  node: TranscriptSearchResult!
  cursor: Cursor!
}
```

---

9. Subgraph: Annotation (Layers, Sketches, Comments)

Owns: `Annotation` **Port:** 4003 **Database tables:** `annotations`

Schema Definition

graphql

```graphql
# ——— File: apps/subgraph-annotation/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: ["@key", "@external", "@requires", "@authenticated", "@requiresScopes"])

# ═══════════════════════════════════════════════════════════════
#  Entity Stubs
# ═══════════════════════════════════════════════════════════════

type User @key(fields: "id", resolvable: false) {
  id: UUID!
}

type MediaAsset @key(fields: "id", resolvable: false) {
  id: UUID!
}

# ═══════════════════════════════════════════════════════════════
#  Extend entities with annotation fields
# ═══════════════════════════════════════════════════════════════

extend type User {
  """All annotations created by this user"""
  annotations(
    first: PositiveInt = 20
    after: Cursor
    filter: AnnotationFilterInput
  ): AnnotationConnection!
}

extend type MediaAsset {
  """All annotations on this media asset"""
  annotations(
    first: PositiveInt = 20
    after: Cursor
    filter: AnnotationFilterInput
  ): AnnotationConnection!

  """Annotations at a specific timestamp (±tolerance seconds)"""
  annotationsAtTimestamp(
    timestamp: Float!
    toleranceSeconds: Float = 2.0
    layers: [AnnotationLayer!]
  ): [Annotation!]!
}
```

```
# ═══════════════════════════════════════════════════════════════
#  Annotation
# ═══════════════════════════════════════════════════════════════

"""
An annotation is any user or AI marking on content:
text notes, sketches, bookmarks, spatial comments, or links.
Annotations are layered (personal, shared, instructor, AI-generated)
and can be temporally/spatially anchored to media.
"""
type Annotation @key(fields: "id") {
  id: UUID!
  type: AnnotationType!
  """Visibility layer"""
  layer: AnnotationLayer!

  # ─── Content ───
  """Text content or markdown (for text, link, bookmark types)"""
  content: String
  """Konva.js sketch data (for sketch type)"""
  sketchData: JSON

  # ─── Temporal anchoring ───
  """Start time in seconds (null for non-temporal annotations)"""
  timestampStart: Float
  """End time in seconds (null for point annotations)"""
  timestampEnd: Float

  # ─── Spatial anchoring (video frame) ───
  """Normalized X position (0-1) on the video frame"""
  spatialX: Float
  """Normalized Y position (0-1) on the video frame"""
  spatialY: Float

  # ─── PDF anchoring ───
  """Page number (1-indexed) for PDF annotations"""
  pageNumber: NonNegativeInt
  """Character offset start for text range selection"""
  textRangeStart: NonNegativeInt
  """Character offset end for text range selection"""
  textRangeEnd: NonNegativeInt

  # ─── Metadata ───
  metadata: AnnotationMetadata
  createdAt: DateTime!
  updatedAt: DateTime!
```

```
  # ——— Relationships ———
  """The user who created this annotation"""
  author: User!
  """The media asset this annotation is attached to"""
  asset: MediaAsset!
  """Parent annotation (for reply chains)"""
  parent: Annotation
  """Replies to this annotation"""
  replies(
    first: PositiveInt = 20
    after: Cursor
  ): AnnotationConnection!
  """Agent that created this annotation (if AI-generated)"""
  agentId: UUID
}

type AnnotationMetadata {
  color: String
  pinned: Boolean
  resolvedAt: DateTime
  linkedConceptIds: [UUID!]
  aiConfidence: UnitFloat
}

# ═══════════════════════════════════════════════════════════════
#  Queries
# ═══════════════════════════════════════════════════════════════

type Query {
  """Fetch a single annotation by ID"""
  annotation(id: UUID!): Annotation @authenticated

  """
  List annotations with filtering.
  Students see their own personal layer + shared/instructor/AI layers.
  """
  annotations(
    first: PositiveInt = 20
    after: Cursor
    last: PositiveInt
    before: Cursor
    filter: AnnotationFilterInput
    orderBy: AnnotationOrderByInput
  ): AnnotationConnection! @authenticated

  """Get annotation thread (all replies to a root annotation)"""
```

```
  annotationThread(rootId: UUID!): [Annotation!]! @authenticated
}

# ---------------------------------------------------------------------------
#  Mutations
# ---------------------------------------------------------------------------

type Mutation {
  """Create a new annotation on a media asset."""
  createAnnotation(input: CreateAnnotationInput!): Annotation!
    @authenticated

  """
  Update an annotation. Users can only update their own annotations.
  Instructors/admins can update shared layer annotations.
  """
  updateAnnotation(id: UUID!, input: UpdateAnnotationInput!): Annotation!
    @authenticated

  """
  Delete an annotation and all its replies (cascade).
  Users can only delete their own annotations.
  """
  deleteAnnotation(id: UUID!): Boolean!
    @authenticated

  """Pin or unpin an annotation (instructors/admins)."""
  toggleAnnotationPin(id: UUID!, pinned: Boolean!): Annotation!
    @authenticated @requiresScopes(scopes: [["annotation:write"]])

  """Mark an annotation as resolved."""
  resolveAnnotation(id: UUID!): Annotation!
    @authenticated

  """Batch-move annotations between layers (instructor/admin only)."""
  moveAnnotationsToLayer(
    annotationIds: [UUID!]!
    targetLayer: AnnotationLayer!
  ): [Annotation!]!
    @authenticated @requiresScopes(scopes: [["annotation:write"]])
}

# ---------------------------------------------------------------------------
#  Input Types
# ---------------------------------------------------------------------------

input CreateAnnotationInput {
```

```
  assetId: UUID!
  type: AnnotationType!
  layer: AnnotationLayer = PERSONAL
  content: String
  sketchData: JSON
  timestampStart: Float
  timestampEnd: Float
  spatialX: Float
  spatialY: Float
  pageNumber: NonNegativeInt
  textRangeStart: NonNegativeInt
  textRangeEnd: NonNegativeInt
  parentId: UUID
  metadata: AnnotationMetadataInput
}

input UpdateAnnotationInput {
  content: String
  sketchData: JSON
  timestampStart: Float
  timestampEnd: Float
  spatialX: Float
  spatialY: Float
  metadata: AnnotationMetadataInput
}

input AnnotationMetadataInput {
  color: String
  linkedConceptIds: [UUID!]
}

input AnnotationFilterInput {
  """Filter by media asset"""
  assetId: UUID
  """Filter by annotation type"""
  type: AnnotationType
  """Filter by layer(s)"""
  layers: [AnnotationLayer!]
  """Filter by author"""
  userId: UUID
  """Only show pinned annotations"""
  pinnedOnly: Boolean
  """Only show unresolved annotations"""
  unresolvedOnly: Boolean
  """Only show root annotations (no replies)"""
  rootOnly: Boolean
  """Full-text search in content"""
```

```
  search: String
  """Filter by time range (start)"""
  timestampFrom: Float
  """Filter by time range (end)"""
  timestampTo: Float
  """Filter by page number (PDF)"""
  pageNumber: NonNegativeInt
}

input AnnotationOrderByInput {
  field: AnnotationSortField!
  direction: SortDirection!
}

enum AnnotationSortField {
  CREATED_AT
  TIMESTAMP_START
  PAGE_NUMBER
  LAYER
}

# ═══════════════════════════════════════════════════════════════
#  Connection Types
# ═══════════════════════════════════════════════════════════════

type AnnotationConnection {
  edges: [AnnotationEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type AnnotationEdge {
  node: Annotation!
  cursor: Cursor!
}
```

---

10. Subgraph: Collaboration (CRDT, Real-time)

Owns: `CollabDocument`, `CollabSession` **Port:** 4004 **Database tables:** `collab_documents`, `crdt_updates`, `collab_sessions`

Schema Definition

graphql

```graphql
# ——— File: apps/subgraph-collaboration/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: ["@key", "@external", "@authenticated", "@requiresScopes"])

# ═══════════════════════════════════════════════════════════════
#  Entity Stubs
# ═══════════════════════════════════════════════════════════════

type User @key(fields: "id", resolvable: false) {
  id: UUID!
}

# ═══════════════════════════════════════════════════════════════
#  CollabDocument
# ═══════════════════════════════════════════════════════════════

"""
A collaborative document backed by Yjs CRDT.
Used for shared annotation layers, course notes, and real-time editing.
"""
type CollabDocument @key(fields: "id") {
  id: UUID!
  """Hocuspocus document name (unique within tenant)"""
  documentName: String!
  """Human-readable title"""
  title: String
  """Type of entity this document is associated with"""
  entityType: String
  """ID of the associated entity"""
  entityId: UUID
  """Number of pending (uncompacted) CRDT updates"""
  pendingUpdates: NonNegativeInt!
  """Whether a compacted state vector snapshot exists"""
  hasSnapshot: Boolean!
  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Relationships ———
  """Currently active collaboration sessions"""
  activeSessions: [CollabSession!]!
  """Number of active collaborators"""
  activeCollaboratorCount: NonNegativeInt!
}
```

```
# ═══════════════════════════════════════════════════════════════
#  CollabSession
# ═══════════════════════════════════════════════════════════════

"""
Represents an active user session within a collaborative document.
Ephemeral — cleaned up when the user disconnects.
"""
type CollabSession @key(fields: "id") {
  id: UUID!
  """The user participating in this session"""
  user: User!
  """The document being collaborated on"""
  document: CollabDocument!
  """Current cursor position of this collaborator"""
  cursorPosition: CursorPosition
  """When the user connected"""
  connectedAt: DateTime!
  """Last heartbeat from the user"""
  lastSeenAt: DateTime!
}

type CursorPosition {
  x: Float
  y: Float
  """Video timestamp if collaborating on video annotations"""
  timestamp: Float
  """PDF page number if collaborating on PDF annotations"""
  pageNumber: NonNegativeInt
  """Display name for the cursor label"""
  userName: String
  """Color assigned to this collaborator"""
  color: String
}

# ═══════════════════════════════════════════════════════════════
#  Queries
# ═══════════════════════════════════════════════════════════════

type Query {
  """Fetch a collaborative document by ID"""
  collabDocument(id: UUID!): CollabDocument @authenticated

  """Fetch a collaborative document by its document name"""
  collabDocumentByName(documentName: String!): CollabDocument @authenticated

  """List collaborative documents for an entity"""
```

```
  collabDocumentsForEntity(
    entityType: String!
    entityId: UUID!
  ): [CollabDocument!]! @authenticated

  """Get the Hocuspocus WebSocket URL for a document"""
  collabConnectionInfo(
    documentId: UUID!
  ): CollabConnectionInfo! @authenticated
}

type CollabConnectionInfo {
  """WebSocket URL for Hocuspocus connection"""
  wsUrl: URL!
  """Authentication token for the WebSocket connection"""
  authToken: String!
  """Document name to use in the Hocuspocus protocol"""
  documentName: String!
}

# ═══════════════════════════════════════════════════════════════
#  Mutations
# ═══════════════════════════════════════════════════════════════

type Mutation {
  """Create a new collaborative document"""
  createCollabDocument(input: CreateCollabDocumentInput!): CollabDocument!
    @authenticated

  """Force-compact CRDT updates into a snapshot (admin)"""
  compactCollabDocument(documentId: UUID!): CollabDocument!
    @authenticated @requiresScopes(scopes: [["org:manage"]])
}

input CreateCollabDocumentInput {
  documentName: String!
  title: String
  entityType: String
  entityId: UUID
}
```

---

11. Subgraph: Agent (Definitions, Executions, Templates)

Owns: `AgentDefinition`, `AgentExecution` **Port:** 4005 **Database tables:** `agent_definitions`, `agent_executions`

Schema Definition

graphql

```
# ——— File: apps/subgraph-agent/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: ["@key", "@external", "@authenticated", "@requiresScopes"])

# ═══════════════════════════════════════════════════════════════
#  Entity Stubs
# ═══════════════════════════════════════════════════════════════

type User @key(fields: "id", resolvable: false) {
  id: UUID!
}

type Course @key(fields: "id", resolvable: false) {
  id: UUID!
}

# ═══════════════════════════════════════════════════════════════
#  Extend User with agent fields
# ═══════════════════════════════════════════════════════════════

extend type User {
  """Agent definitions created by this user"""
  agentDefinitions(
    first: PositiveInt = 20
    after: Cursor
    filter: AgentDefinitionFilterInput
  ): AgentDefinitionConnection!

  """Agent execution history for this user"""
  agentExecutions(
    first: PositiveInt = 20
    after: Cursor
    filter: AgentExecutionFilterInput
  ): AgentExecutionConnection!
}

# ═══════════════════════════════════════════════════════════════
#  AgentDefinition
# ═══════════════════════════════════════════════════════════════

"""
A user-created or built-in AI agent definition.
Agents process content using LLMs based on configurable templates
(Chavruta debate partner, summarizer, quiz master, etc.).
```

```
"""
type AgentDefinition @key(fields: "id") {
  id: UUID!
  name: String!
  description: String
  """Base template type"""
  templateType: AgentTemplate!
  """Whether this is a system-provided built-in agent"""
  isBuiltIn: Boolean!
  """Whether this agent is visible to all users in the tenant"""
  isPublic: Boolean!

  # ——— Configuration ———
  config: AgentConfig!

  # ——— Usage Stats ———
  """Total number of times this agent has been executed"""
  executionCount: NonNegativeInt!
  """Average execution time in milliseconds"""
  avgExecutionMs: Float
  """Last time this agent was executed"""
  lastExecutedAt: DateTime

  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Relationships ———
  """User who created this agent definition"""
  creator: User!
  """Recent executions of this agent"""
  recentExecutions(
    first: PositiveInt = 10
    after: Cursor
  ): AgentExecutionConnection!
}

type AgentConfig {
  """Scope of data the agent can access"""
  dataScope: AgentDataScope!
  """Custom Cypher query for CUSTOM_QUERY scope"""
  customQuery: String
  """What triggers the agent"""
  triggerType: AgentTrigger!
  """How the agent's output is displayed"""
  outputFormat: AgentOutputFormat!
  """Agent personality description"""
  personality: String
```

```
  """Difficulty level for educational agents"""
  difficulty: AgentDifficulty
  """Override the default LLM model"""
  modelOverride: String
  """Maximum tokens for the response"""
  maxTokens: NonNegativeInt
  """LLM temperature (0.0-2.0)"""
  temperature: Float
  """Custom system prompt"""
  systemPrompt: String
  """List of MCP tools this agent can use"""
  toolsEnabled: [String!]
}

enum AgentDataScope {
  MY_NOTES
  COURSE
  ORGANIZATION
  CUSTOM_QUERY
}

enum AgentDifficulty {
  BEGINNER
  INTERMEDIATE
  ADVANCED
}

# ═══════════════════════════════════════════════════════════════════
#  AgentExecution
# ═══════════════════════════════════════════════════════════════════

"""
A record of a single agent execution.
Contains input, output, performance metrics, and debugging info.
"""
type AgentExecution @key(fields: "id") {
  id: UUID!
  """Current status of the execution"""
  status: ExecutionStatus!

  # ─── Input / Output ───
  """Input provided to the agent"""
  input: JSON!
  """Agent's output (null while running)"""
  output: JSON
  """Chain-of-thought reasoning (for debugging)"""
  reasoning: String
```

```
  """Error message if execution failed"""
  errorMessage: String

  # ——— Performance ———
  """Number of tokens consumed"""
  tokensUsed: NonNegativeInt
  """Execution time in milliseconds"""
  executionMs: NonNegativeInt
  """LLM model used for this execution"""
  modelUsed: String

  createdAt: DateTime!
  updatedAt: DateTime!

  # ——— Relationships ———
  """The agent definition that was executed"""
  agent: AgentDefinition!
  """The user who triggered the execution"""
  user: User!
}

# ═══════════════════════════════════════════════════════════════════
#  Queries
# ═══════════════════════════════════════════════════════════════════

type Query {
  """Fetch an agent definition by ID"""
  agentDefinition(id: UUID!): AgentDefinition @authenticated

  """List available agent definitions (includes public + own)"""
  agentDefinitions(
    first: PositiveInt = 20
    after: Cursor
    filter: AgentDefinitionFilterInput
    orderBy: AgentDefinitionOrderByInput
  ): AgentDefinitionConnection! @authenticated

  """List built-in agent templates"""
  agentTemplates: [AgentTemplateInfo!]! @authenticated

  """Fetch a specific agent execution"""
  agentExecution(id: UUID!): AgentExecution @authenticated

  """List agent executions with filtering"""
  agentExecutions(
    first: PositiveInt = 20
    after: Cursor
```

```
    filter: AgentExecutionFilterInput
  ): AgentExecutionConnection! @authenticated
}

"""Information about a built-in agent template"""
type AgentTemplateInfo {
  templateType: AgentTemplate!
  name: String!
  description: String!
  """Default configuration for this template"""
  defaultConfig: AgentConfig!
  """Icon identifier for UI"""
  icon: String!
}

# ============================================================
#  Mutations
# ============================================================

type Mutation {
  """Create a new agent definition"""
  createAgentDefinition(input: CreateAgentDefinitionInput!): AgentDefinition!
    @authenticated @requiresScopes(scopes: [["agent:write"]])

  """Update an agent definition (creator only)"""
  updateAgentDefinition(id: UUID!, input: UpdateAgentDefinitionInput!): AgentDefinition!
    @authenticated @requiresScopes(scopes: [["agent:write"]])

  """Delete an agent definition (creator only, cascades executions)"""
  deleteAgentDefinition(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["agent:write"]])

  """
  Execute an agent. Returns immediately with QUEUED status.
  Use the agentExecutionUpdated subscription to track progress.
  """
  executeAgent(input: ExecuteAgentInput!): AgentExecution!
    @authenticated @requiresScopes(scopes: [["agent:execute"]])
    @rateLimit(max: 30, window: "1h")

  """Cancel a running agent execution"""
  cancelAgentExecution(executionId: UUID!): AgentExecution!
    @authenticated
}

# ============================================================
#  Input Types
```

```
# ---------------------------------------------------------------------------

input CreateAgentDefinitionInput {
  name: String!
  description: String
  templateType: AgentTemplate!
  isPublic: Boolean = false
  config: AgentConfigInput!
}

input UpdateAgentDefinitionInput {
  name: String
  description: String
  isPublic: Boolean
  config: AgentConfigInput
}

input AgentConfigInput {
  dataScope: AgentDataScope!
  customQuery: String
  triggerType: AgentTrigger!
  outputFormat: AgentOutputFormat!
  personality: String
  difficulty: AgentDifficulty
  modelOverride: String
  maxTokens: NonNegativeInt
  temperature: Float
  systemPrompt: String
  toolsEnabled: [String!]
}

input ExecuteAgentInput {
  agentId: UUID!
  """Context-specific input (e.g., annotation ID, course ID, free-text prompt)"""
  input: JSON!
}

input AgentDefinitionFilterInput {
  templateType: AgentTemplate
  isBuiltIn: Boolean
  isPublic: Boolean
  creatorId: UUID
  search: String
}

input AgentDefinitionOrderByInput {
  field: AgentDefinitionSortField!
```

```
  direction: SortDirection!
}

enum AgentDefinitionSortField {
  NAME
  CREATED_AT
  EXECUTION_COUNT
  TEMPLATE_TYPE
}

input AgentExecutionFilterInput {
  agentId: UUID
  status: ExecutionStatus
  userId: UUID
}

# ═══════════════════════════════════════════════════════════════
#  Connection Types
# ═══════════════════════════════════════════════════════════════

type AgentDefinitionConnection {
  edges: [AgentDefinitionEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type AgentDefinitionEdge {
  node: AgentDefinition!
  cursor: Cursor!
}

type AgentExecutionConnection {
  edges: [AgentExecutionEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type AgentExecutionEdge {
  node: AgentExecution!
  cursor: Cursor!
}
```

---

12. Subgraph: Knowledge (Graph, Embeddings, Semantic Search)

Owns: `Concept`, `Person`, `Term`, `Source`, `TopicCluster`, `KnowledgeRelation`, `SemanticSearchResult` **Port:** 4006

Database: Apache AGE graph + pgvector embeddings

Schema Definition

```graphql
```

```
# ——— File: apps/subgraph-knowledge/src/schema.graphql ———

extend schema
  @link(url: "https://specs.apollo.dev/federation/v2.7",
    import: ["@key", "@external", "@requires", "@authenticated", "@requiresScopes"])

# ═══════════════════════════════════════════════════════════════
#  Entity Stubs
# ═══════════════════════════════════════════════════════════════

type MediaAsset @key(fields: "id", resolvable: false) {
  id: UUID!
}

type TranscriptSegment @key(fields: "id", resolvable: false) {
  id: UUID!
}

type Annotation @key(fields: "id", resolvable: false) {
  id: UUID!
}

# ═══════════════════════════════════════════════════════════════
#  Extend MediaAsset with knowledge fields
# ═══════════════════════════════════════════════════════════════

extend type MediaAsset {
  """Concepts mentioned in this media asset's transcript"""
  mentionedConcepts(
    first: PositiveInt = 20
    after: Cursor
  ): ConceptConnection!

  """
  Semantic context for a specific timestamp.
  Powers the "Contextual Copilot" sidebar.
  """
  semanticContextAt(timestamp: Float!): SemanticContext!
}

extend type TranscriptSegment {
  """Concepts mentioned in this specific segment"""
  mentionedConcepts: [ConceptMention!]!
}

extend type Annotation {
```

```
  """Concepts linked to this annotation"""
  linkedConcepts: [Concept!]!
}

# ═══════════════════════════════════════════════════════════════
#  Concept (Knowledge Graph Vertex)
# ═══════════════════════════════════════════════════════════════

"""
A semantic concept in the knowledge graph.
Concepts are extracted from content (AI or manual) and form
the backbone of the knowledge graph.
"""
type Concept @key(fields: "id") {
  id: UUID!
  """Primary label (Hebrew/English)"""
  label: String!
  """Detailed description of the concept"""
  description: String
  """Alternative names, abbreviations, synonyms"""
  aliases: [String!]!
  """Academic or knowledge domain (e.g., "Physics", "Talmud")"""
  domain: String
  """Confidence: 1.0 = manually created, <1.0 = AI-extracted"""
  confidence: UnitFloat!
  """How this concept was created"""
  sourceType: ConceptSourceType!
  createdAt: DateTime!

  # ─── Knowledge Graph Relationships ───
  """Concepts related to this one (with strength scores)"""
  relatedConcepts(
    first: PositiveInt = 20
    after: Cursor
    minStrength: UnitFloat
  ): KnowledgeRelationConnection!

  """Concepts that contradict this one"""
  contradictions: [Contradiction!]!

  """Prerequisites for understanding this concept"""
  prerequisites: [PrerequisiteLink!]!

  """Concepts that require this concept as a prerequisite"""
  dependents: [PrerequisiteLink!]!

  """Full prerequisite learning path (up to 5 levels deep)"""
```

```
  prerequisiteChain: [Concept!]!

  """Media segments where this concept is mentioned"""
  mentions(
    first: PositiveInt = 20
    after: Cursor
  ): ConceptMentionConnection!

  """Topic cluster this concept belongs to"""
  topicCluster: TopicCluster
}

enum ConceptSourceType {
  MANUAL
  AI_EXTRACTED
  IMPORTED
}

# ═══════════════════════════════════════════════════════════════════
#  Knowledge Graph Edge Types
# ═══════════════════════════════════════════════════════════════════

"""A relationship between two concepts"""
type KnowledgeRelation {
  """The related concept"""
  concept: Concept!
  """Semantic similarity or relationship strength (0-1)"""
  strength: UnitFloat!
  """How the relationship was established"""
  method: RelationMethod!
  """Human-readable explanation"""
  evidence: String
}

enum RelationMethod {
  MANUAL
  EMBEDDING_SIMILARITY
  CO_OCCURRENCE
  CITATION
}

"""A contradiction between two concepts"""
type Contradiction {
  """The contradicting concept"""
  concept: Concept!
  """Explanation of why they contradict"""
  evidence: String!
```

```
  """Severity of the contradiction"""
  severity: ContradictionSeverity!
  """How the contradiction was detected"""
  detectedBy: ContradictionDetector!
  """Source media asset A"""
  sourceAssetA: MediaAsset
  """Source media asset B"""
  sourceAssetB: MediaAsset
}

enum ContradictionSeverity {
  MINOR
  MODERATE
  MAJOR
}

enum ContradictionDetector {
  MANUAL
  CHAVRUTA_AGENT
  AI_INFERENCE
}

"""A prerequisite relationship between concepts"""
type PrerequisiteLink {
  """The prerequisite (or dependent) concept"""
  concept: Concept!
  """How strongly this is recommended"""
  strength: PrerequisiteStrength!
}

enum PrerequisiteStrength {
  REQUIRED
  RECOMMENDED
  OPTIONAL
}

"""A mention of a concept in a specific transcript segment"""
type ConceptMention {
  concept: Concept!
  segment: TranscriptSegment!
  asset: MediaAsset!
  startTime: Float!
  endTime: Float!
  confidence: UnitFloat!
}

"""Auto-generated grouping of related concepts"""
```

```
type TopicCluster {
  id: UUID!
  label: String!
  conceptCount: NonNegativeInt!
  coherenceScore: UnitFloat!
  """Concepts in this cluster"""
  concepts(
    first: PositiveInt = 20
    after: Cursor
  ): ConceptConnection!
}

# ---------------------------------------------------------------------------
#  Person (Knowledge Graph Vertex)
# ---------------------------------------------------------------------------

"""
A person referenced in content — author, speaker, historical figure.
Stored as a vertex in the Apache AGE knowledge graph.
"""
type Person @key(fields: "id") {
  id: UUID!
  """Full name of the person"""
  name: String!
  """Role in context (e.g., "Speaker", "Author", "Historical")"""
  role: String
  """External reference URL (e.g., Wikipedia)"""
  externalUrl: URL
  """Content this person authored"""
  authoredSources(
    first: PositiveInt = 10
    after: Cursor
  ): SourceConnection!
  """Concepts this person is associated with"""
  relatedConcepts: [Concept!]!
  createdAt: DateTime!
}

# ---------------------------------------------------------------------------
#  Term (Knowledge Graph Vertex)
# ---------------------------------------------------------------------------

"""
A domain-specific term — narrower and more precise than a Concept.
Terms are leaf nodes in the knowledge graph, linked to parent Concepts.
"""
type Term @key(fields: "id") {
```

```
  id: UUID!
  """Display label of the term"""
  label: String!
  """Knowledge domain (e.g., "Physics", "Talmud")"""
  domain: String!
  """Formal definition of the term"""
  definition: String
  """Parent concept this term belongs to"""
  parentConcept: Concept
  createdAt: DateTime!
}

# ---------------------------------------------------------------------------
#  Source (Knowledge Graph Vertex)
# ---------------------------------------------------------------------------

"""
An external reference — book, paper, URL, lecture, or internal asset.
Sources are vertices in the knowledge graph and can be linked to
Concepts via CITES or REFERS_TO edges.
"""
type Source @key(fields: "id") {
  id: UUID!
  """Title of the source material"""
  title: String!
  """Type of source"""
  type: SourceType!
  """External URL (for web resources)"""
  externalUrl: URL
  """Internal media asset (if type is INTERNAL_ASSET)"""
  internalAsset: MediaAsset
  """ISBN for books"""
  isbn: String
  """DOI for papers"""
  doi: String
  """Person(s) who authored this source"""
  authors: [Person!]!
  """Concepts cited or referenced in this source"""
  citedConcepts(
    first: PositiveInt = 20
    after: Cursor
  ): ConceptConnection!
  createdAt: DateTime!
}

enum SourceType {
  BOOK
```

```
  PAPER
  URL
  LECTURE
  INTERNAL_ASSET
}

# ---------------------------------------------------------------------------
#  Semantic Context (Contextual Copilot)
# ---------------------------------------------------------------------------

"""
Rich semantic context for a point in time within a media asset.
Combines knowledge graph traversal with vector similarity.
Powers the "Contextual Copilot" sidebar in the video player.
"""
type SemanticContext {
  """Concepts being discussed at this timestamp"""
  activeConcepts: [ConceptWithRelations!]!
  """Suggested related content from other assets"""
  relatedContent: [RelatedContentSuggestion!]!
  """Detected contradictions at this point"""
  activeContradictions: [Contradiction!]!
  """Suggested prerequisite concepts the viewer may need"""
  suggestedPrerequisites: [Concept!]!
}

type ConceptWithRelations {
  concept: Concept!
  """How this concept connects to other active concepts"""
  connections: [ConceptConnection!]!
}

type ConceptConnection {
  relatedConcept: Concept!
  relationType: String!
  strength: UnitFloat!
}

type RelatedContentSuggestion {
  """The related media asset"""
  asset: MediaAsset!
  """The segment within the asset that is most relevant"""
  segment: TranscriptSegment
  """Why this content is suggested"""
  reason: String!
  """Relevance score"""
  relevance: UnitFloat!
```

```
}

# ═══════════════════════════════════════════════════════════════════════
#  Semantic Search
# ═══════════════════════════════════════════════════════════════════════

"""
Result of a semantic (vector) search across content.
Combines pgvector nearest-neighbor search with knowledge graph context.
"""
type SemanticSearchResult {
  """The transcript segment that matched"""
  segment: TranscriptSegment!
  """The media asset containing the match"""
  asset: MediaAsset!
  """The text that was matched"""
  matchedText: String!
  """Cosine similarity score"""
  similarity: UnitFloat!
  """Concepts mentioned in the matched segment"""
  relatedConcepts: [Concept!]!
}

# ═══════════════════════════════════════════════════════════════════════
#  Queries
# ═══════════════════════════════════════════════════════════════════════

type Query {
  """Fetch a concept by ID"""
  concept(id: UUID!): Concept @authenticated

  """Search concepts by label, alias, or description"""
  concepts(
    first: PositiveInt = 20
    after: Cursor
    filter: ConceptFilterInput
    orderBy: ConceptOrderByInput
  ): ConceptConnection! @authenticated

  """
  Semantic search across all content using natural language.
  Uses pgvector HNSW index for approximate nearest neighbor search.
  """
  semanticSearch(
    """Natural language query (will be embedded by the server)"""
    query: String!
    first: PositiveInt = 10
```

```
  after: Cursor
  """Minimum similarity threshold (0-1)"""
  minSimilarity: UnitFloat = 0.7
  """Limit search to specific asset IDs"""
  assetIds: [UUID!]
): SemanticSearchResultConnection! @authenticated

"""
HybridRAG search: combines vector search + knowledge graph traversal.
Best for complex queries that need both semantic and structural context.
"""
hybridSearch(
  query: String!
  first: PositiveInt = 10
  """How many graph hops to follow from vector results"""
  graphDepth: PositiveInt = 2
): HybridSearchResultConnection! @authenticated

"""Find concepts related to a given concept (graph traversal)"""
relatedConcepts(
  conceptId: UUID!
  """Max graph traversal depth"""
  maxDepth: PositiveInt = 3
  first: PositiveInt = 50
): KnowledgeRelationConnection! @authenticated

"""Find contradictions for a concept"""
contradictions(
  conceptId: UUID!
): [Contradiction!]! @authenticated

"""Build a learning path (prerequisite chain) for a concept"""
learningPath(
  conceptId: UUID!
  maxDepth: PositiveInt = 5
): [Concept!]! @authenticated

"""List topic clusters"""
topicClusters(
  first: PositiveInt = 20
  after: Cursor
): TopicClusterConnection! @authenticated

# ——— Person queries ———

"""Fetch a person by ID"""
person(id: UUID!): Person @authenticated
```

```
  """Search people referenced in the knowledge graph"""
  people(
    first: PositiveInt = 20
    after: Cursor
    search: String
  ): PersonConnection! @authenticated

  # ——— Term queries ———

  """Fetch a term by ID"""
  term(id: UUID!): Term @authenticated

  """Search domain-specific terms"""
  terms(
    first: PositiveInt = 20
    after: Cursor
    domain: String
    search: String
  ): TermConnection! @authenticated

  # ——— Source queries ———

  """Fetch a source by ID"""
  source(id: UUID!): Source @authenticated

  """Search sources (books, papers, URLs, etc.)"""
  sources(
    first: PositiveInt = 20
    after: Cursor
    type: SourceType
    search: String
  ): SourceConnection! @authenticated
}

# ═══════════════════════════════════════════════════════════════
#  Mutations
# ═══════════════════════════════════════════════════════════════

type Mutation {
  """Manually create a concept in the knowledge graph"""
  createConcept(input: CreateConceptInput!): Concept!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Update a concept's label, description, aliases, or domain"""
  updateConcept(id: UUID!, input: UpdateConceptInput!): Concept!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])
```

```
"""Delete a concept and all its edges from the knowledge graph"""
deleteConcept(id: UUID!): Boolean!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Create a relationship between two concepts"""
createRelation(input: CreateRelationInput!): Boolean!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Remove a relationship between two concepts"""
deleteRelation(
  fromConceptId: UUID!
  toConceptId: UUID!
  relationType: String!
): Boolean!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Mark a contradiction between two concepts"""
createContradiction(input: CreateContradictionInput!): Contradiction!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""
Trigger re-indexing of embeddings for a media asset.
Used after transcript updates or corrections.
"""
reindexAssetEmbeddings(assetId: UUID!): Boolean!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Review and approve/reject an AI-inferred relationship"""
reviewInferredRelation(
  fromConceptId: UUID!
  toConceptId: UUID!
  approved: Boolean!
): Boolean!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

# ——— Person mutations ———

"""Create a person reference in the knowledge graph"""
createPerson(input: CreatePersonInput!): Person!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Update a person's details"""
updatePerson(id: UUID!, input: UpdatePersonInput!): Person!
  @authenticated @requiresScopes(scopes: [["knowledge:write"]])

"""Delete a person from the knowledge graph"""
```

```
  deletePerson(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  # ——— Term mutations ———

  """Create a domain-specific term"""
  createTerm(input: CreateTermInput!): Term!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Update a term's details"""
  updateTerm(id: UUID!, input: UpdateTermInput!): Term!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Delete a term from the knowledge graph"""
  deleteTerm(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  # ——— Source mutations ———

  """Create a source reference (book, paper, URL, etc.)"""
  createSource(input: CreateSourceInput!): Source!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Update a source's details"""
  updateSource(id: UUID!, input: UpdateSourceInput!): Source!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Delete a source from the knowledge graph"""
  deleteSource(id: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Link a person as author of a source (AUTHORED_BY edge)"""
  linkAuthorToSource(personId: UUID!, sourceId: UUID!): Boolean!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])

  """Link a source to a concept (CITES edge)"""
  linkSourceToConcept(
    sourceId: UUID!
    conceptId: UUID!
    page: Int
    timestamp: Float
    context: String
  ): Boolean!
    @authenticated @requiresScopes(scopes: [["knowledge:write"]])
}

# ——————————————————————————————————————————————————————————
```

```
# Input Types
# ---------------------------------------------------------------------------

input CreateConceptInput {
  label: String!
  description: String
  aliases: [String!]
  domain: String
}

input UpdateConceptInput {
  label: String
  description: String
  aliases: [String!]
  domain: String
}

input CreateRelationInput {
  fromConceptId: UUID!
  toConceptId: UUID!
  relationType: RelationType!
  strength: UnitFloat
  evidence: String
}

"""
All relationship/edge types in the Apache AGE knowledge graph.
Maps directly to edge labels defined in the graph ontology.
"""
enum RelationType {
  """Semantic similarity or co-occurrence relationship"""
  RELATED_TO
  """Concept A is a prerequisite for understanding Concept B"""
  PREREQUISITE_OF
  """Concept B was derived from Concept A"""
  DERIVED_FROM
  """Source cites a concept (with optional page/timestamp)"""
  CITES
  """Term/Concept belongs to a TopicCluster"""
  BELONGS_TO
  """A transcript segment mentions a concept"""
  MENTIONS
  """A source or concept refers to another entity"""
  REFERS_TO
  """Two concepts contradict each other"""
  CONTRADICTS
  """A source was authored by a person"""
```

```
  AUTHORED_BY
  """AI-inferred relationship (pending review)"""
  INFERRED_RELATED
}

input CreateContradictionInput {
  conceptIdA: UUID!
  conceptIdB: UUID!
  evidence: String!
  severity: ContradictionSeverity!
  sourceAssetIdA: UUID
  sourceAssetIdB: UUID
}

input ConceptFilterInput {
  """Full-text search on label, aliases, and description"""
  search: String
  """Filter by domain"""
  domain: String
  """Filter by source type"""
  sourceType: ConceptSourceType
  """Minimum confidence threshold"""
  minConfidence: UnitFloat
}

input ConceptOrderByInput {
  field: ConceptSortField!
  direction: SortDirection!
}

enum ConceptSortField {
  LABEL
  CONFIDENCE
  CREATED_AT
  DOMAIN
}

# ──────────────────────────────────────────────────────────────
#  Connection Types
# ──────────────────────────────────────────────────────────────

type ConceptConnection {
  edges: [ConceptEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}
```

```
type ConceptEdge {
  node: Concept!
  cursor: Cursor!
}

type KnowledgeRelationConnection {
  edges: [KnowledgeRelationEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type KnowledgeRelationEdge {
  node: KnowledgeRelation!
  cursor: Cursor!
}

type ConceptMentionConnection {
  edges: [ConceptMentionEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type ConceptMentionEdge {
  node: ConceptMention!
  cursor: Cursor!
}

type SemanticSearchResultConnection {
  edges: [SemanticSearchResultEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type SemanticSearchResultEdge {
  node: SemanticSearchResult!
  cursor: Cursor!
}

"""Hybrid search result combining vector similarity with graph context"""
type HybridSearchResult {
  """Vector search result"""
  segment: TranscriptSegment!
  asset: MediaAsset!
  matchedText: String!
  similarity: UnitFloat!
  """Additional context from knowledge graph traversal"""
  graphContext: [ConceptWithRelations!]!
```

```
  """Combined relevance score (vector + graph)"""
  combinedScore: Float!
}

type HybridSearchResultConnection {
  edges: [HybridSearchResultEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type HybridSearchResultEdge {
  node: HybridSearchResult!
  cursor: Cursor!
}

type TopicClusterConnection {
  edges: [TopicClusterEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type TopicClusterEdge {
  node: TopicCluster!
  cursor: Cursor!
}

# ---------------------------------------------------------------------------
#  Person / Term / Source Input Types
# ---------------------------------------------------------------------------

input CreatePersonInput {
  name: String!
  role: String
  externalUrl: URL
}

input UpdatePersonInput {
  name: String
  role: String
  externalUrl: URL
}

input CreateTermInput {
  label: String!
  domain: String!
  definition: String
  """Link to a parent concept"""
```

```
  parentConceptId: UUID
}

input UpdateTermInput {
  label: String
  domain: String
  definition: String
  parentConceptId: UUID
}

input CreateSourceInput {
  title: String!
  type: SourceType!
  externalUrl: URL
  """Link to an existing internal media asset"""
  internalAssetId: UUID
  isbn: String
  doi: String
}

input UpdateSourceInput {
  title: String
  type: SourceType
  externalUrl: URL
  internalAssetId: UUID
  isbn: String
  doi: String
}

# ---------------------------------------------------------------------------
#  Person / Term / Source Connection Types
# ---------------------------------------------------------------------------

type PersonConnection {
  edges: [PersonEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type PersonEdge {
  node: Person!
  cursor: Cursor!
}

type TermConnection {
  edges: [TermEdge!]!
  pageInfo: PageInfo!
```

```
  totalCount: NonNegativeInt
}

type TermEdge {
  node: Term!
  cursor: Cursor!
}

type SourceConnection {
  edges: [SourceEdge!]!
  pageInfo: PageInfo!
  totalCount: NonNegativeInt
}

type SourceEdge {
  node: Source!
  cursor: Cursor!
}
```

---

13. Subscriptions Contract

Subscriptions are handled via **Hive Gateway's WebSocket passthrough** mode. The gateway forwards WebSocket connections (using the `graphql-ws` protocol) directly to the subgraph that defines the subscription. For HTTP-based clients (e.g., React Native), the gateway also supports **Server-Sent Events (SSE)** as a fallback transport.

Each subgraph that defines subscriptions backs them with **NATS JetStream** events.

Subscription Definitions

graphql

```graphql
# ——— Defined in their respective subgraph schemas ———

# ═══════════════════════════════════════════════════════════════
#  Content Subgraph Subscriptions
# ═══════════════════════════════════════════════════════════════

# In: apps/subgraph-content/src/schema.graphql
type Subscription {
  """
  Fired when the transcription status of a media asset changes.
  Useful for showing progress in the upload flow.
  """
  transcriptionStatusChanged(assetId: UUID!): MediaAsset! @authenticated

  """
  Fired when a new transcript segment is available during live transcription.
  """
  transcriptSegmentAdded(assetId: UUID!): TranscriptSegment! @authenticated
}

# ═══════════════════════════════════════════════════════════════
#  Annotation Subgraph Subscriptions
# ═══════════════════════════════════════════════════════════════

# In: apps/subgraph-annotation/src/schema.graphql
type Subscription {
  """
  Fired when an annotation is created, updated, or deleted on an asset.
  Used for real-time annotation collaboration.
  """
  annotationChanged(
    assetId: UUID!
    layers: [AnnotationLayer!]
  ): AnnotationChangeEvent! @authenticated
}

type AnnotationChangeEvent {
  """Type of change"""
  changeType: ChangeType!
  """The annotation that changed (null if deleted)"""
  annotation: Annotation
  """ID of the deleted annotation (only for DELETE)"""
  deletedAnnotationId: UUID
}

enum ChangeType {
```

```
  CREATED
  UPDATED
  DELETED
}

# ═══════════════════════════════════════════════════════════════
#  Collaboration Subgraph Subscriptions
# ═══════════════════════════════════════════════════════════════

# In: apps/subgraph-collaboration/src/schema.graphql
type Subscription {
  """
  Fired when collaborators join/leave or update cursor position.
  Used for showing live presence indicators.
  """
  collaboratorPresenceChanged(
    documentId: UUID!
  ): PresenceEvent! @authenticated
}

type PresenceEvent {
  eventType: PresenceEventType!
  session: CollabSession
  disconnectedUserId: UUID
}

enum PresenceEventType {
  JOINED
  LEFT
  CURSOR_MOVED
}

# ═══════════════════════════════════════════════════════════════
#  Agent Subgraph Subscriptions
# ═══════════════════════════════════════════════════════════════

# In: apps/subgraph-agent/src/schema.graphql
type Subscription {
  """
  Fired when an agent execution's status changes.
  Stream updates from QUEUED → RUNNING → COMPLETED/FAILED.
  Also streams partial output during RUNNING state.
  """
  agentExecutionUpdated(
    executionId: UUID!
  ): AgentExecution! @authenticated
```

```
  """
  Stream the agent's response tokens as they're generated.
  Enables showing the agent's response character by character.
  """
  agentResponseStream(
    executionId: UUID!
  ): AgentStreamChunk! @authenticated
}

"""A chunk of the agent's streaming response"""
type AgentStreamChunk {
  """The text chunk"""
  text: String!
  """Whether this is the final chunk"""
  done: Boolean!
  """Running token count"""
  tokensSoFar: NonNegativeInt!
}

# ═══════════════════════════════════════════════════════════════
#  Knowledge Subgraph Subscriptions
# ═══════════════════════════════════════════════════════════════

# In: apps/subgraph-knowledge/src/schema.graphql
type Subscription {
  """
  Fired when new concepts are extracted from content
  (e.g., after transcription completes and NLP pipeline runs).
  """
  conceptsExtracted(
    assetId: UUID!
  ): [Concept!]! @authenticated
}
```

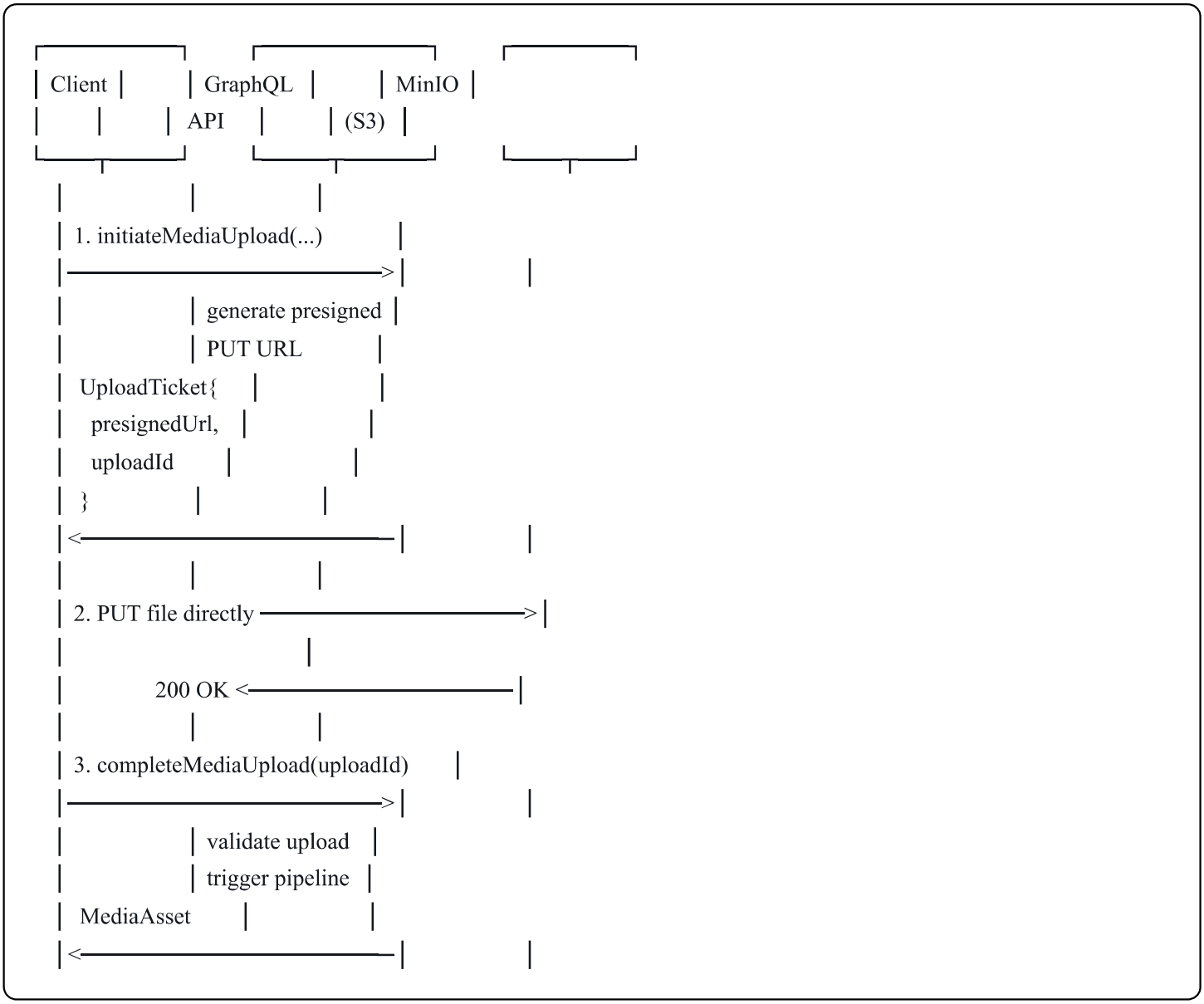
NATS Subject Mapping

Subscription	NATS Subject Pattern	Publisher
`transcriptionStatusChanged`	`edusphere.{tenant_id}.media.{asset_id}.transcription.status`	Media Worker
`transcriptSegmentAdded`	`edusphere.{tenant_id}.media.{asset_id}.segment.added`	Transcription Worker
`annotationChanged`	`edusphere.{tenant_id}.annotation.{asset_id}.changed`	Annotation Service
`collaboratorPresenceChanged`	`edusphere.{tenant_id}.collab.{document_id}.presence`	Hocuspocus Server

Subscription	NATS Subject Pattern	Publisher
agentExecutionUpdated	edusphere.{tenant_id}.agent.execution.{execution_id}.updated	Agent Runner
agentResponseStream	edusphere.{tenant_id}.agent.execution.{execution_id}.stream	Agent Runner
conceptsExtracted	edusphere.{tenant_id}.knowledge.{asset_id}.concepts.extracted	NLP Pipeline

14. File Upload Contract (Multipart)

File uploads follow the **two-phase upload pattern**:



Max file sizes per plan:

Plan	Max File Size	Max Total Storage
Free	100 MB	1 GB
Starter	500 MB	25 GB
Professional	2 GB	100 GB
Enterprise	10 GB	Unlimited

15. Gateway Configuration & Router Settings

Hive Gateway Configuration

typescript

```ts
// ——— File: apps/gateway/src/gateway.config.ts ———

import { defineConfig } from '@graphql-hive/gateway';

export const gatewayConfig = defineConfig({
  // Supergraph SDL source
  supergraph: './supergraph.graphql',

  // Server settings
  port: 4000,
  hostname: '0.0.0.0',
  graphqlEndpoint: '/graphql',
  landingPage: process.env.NODE_ENV !== 'production',

  // CORS
  cors: {
    origin: [
      'https://app.edusphere.dev',
      /\.edusphere\.dev$/,
      'http://localhost:3000',
    ],
    credentials: true,
    allowedHeaders: [
      'Content-Type',
      'Authorization',
      'x-request-id',
      'x-correlation-id',
    ],
  },

  // JWT Authentication (Keycloak)
  jwt: {
    jwks: {
      url: 'https://keycloak.edusphere.dev/realms/edusphere/protocol/openid-connect/certs',
    },
    tokenLookupLocations: [
      { type: 'header', name: 'authorization', prefix: 'Bearer ' },
    ],
    // Claims are extracted and propagated as headers to subgraphs
    forward: {
      payload: true,       // Forward full JWT payload
      extensions: true,
    },
  },

  // Subscription support (WebSocket passthrough)
```

```
subscriptions: {
  enabled: true,
  protocol: 'ws',        // graphql-ws protocol
  path: '/ws',
},

// Header propagation to subgraphs
propagateHeaders: {
  fromClientToSubgraphs({ request, subgraphName }) {
    return {
      authorization: request.headers.get('authorization') ?? '',
      'x-tenant-id': request.headers.get('x-tenant-id') ?? '',
      'x-user-id': request.headers.get('x-user-id') ?? '',
      'x-user-role': request.headers.get('x-user-role') ?? '',
      'x-request-id': request.headers.get('x-request-id') ?? crypto.randomUUID(),
      'x-correlation-id': request.headers.get('x-correlation-id') ?? '',
      'x-gateway-version': '1.0.0',
    };
  },
},

// OpenTelemetry integration
openTelemetry: {
  exporters: [
    {
      type: 'otlp',
      endpoint: 'http://jaeger:4317',
    },
  ],
},

// Prometheus metrics
prometheus: {
  enabled: true,
  endpoint: '/metrics',
},

// Rate limiting
rateLimiting: {
  global: {
    max: 10000,
    window: '1s',
  },
},

// Query depth limiting
security: {
```

```
  maxDepth: 15,
  maxAliases: 30,
  maxDirectives: 50,
  maxTokens: 10000,
 },

 // Persisted operations (APQ)
 persistedOperations: {
  enabled: true,
  allowArbitraryOperations: process.env.NODE_ENV !== 'production',
 },

 // Execution settings
 executionCancellation: true,
 maskedErrors: process.env.NODE_ENV === 'production',
 includeSubgraphErrors: process.env.NODE_ENV !== 'production',
});
```

Subgraph NestJS Bootstrap (GraphQL Yoga)

typescript

```
// ——— File: apps/subgraph-core/src/app.module.ts ———
// Pattern repeated for each subgraph with its own schema

import { Module } from '@nestjs/common';
import { GraphQLModule } from '@graphql-yoga/nestjs';
import { YogaFederationDriver, YogaFederationDriverConfig } from '@graphql-yoga/nestjs/federation';

@Module({
  imports: [
    GraphQLModule.forRoot<YogaFederationDriverConfig>({
      driver: YogaFederationDriver,
      typePaths: ['./src/**/*.graphql'],
      // GraphQL Yoga plugins
      plugins: [],
      // Context extraction from gateway headers
      context: ({ request }) => ({
        tenantId: request.headers.get('x-tenant-id'),
        userId: request.headers.get('x-user-id'),
        userRole: request.headers.get('x-user-role'),
        requestId: request.headers.get('x-request-id'),
      }),
    }),
    // ... domain modules
  ],
})
export class AppModule {}
```

Supergraph Composition

bash

```
# Compose supergraph SDL from subgraph schemas using Hive CLI
# Install: npm install -g @graphql-hive/cli

# Local composition (dev)
hive supergraph compose \
  --config supergraph-config.yaml \
  --output supergraph.graphql

# Publish schema to Hive registry (CI/CD)
hive schema:publish \
  --registry.endpoint https://app.graphql-hive.com \
  --registry.accessToken $HIVE_TOKEN \
  --service core \
  --url http://subgraph-core:4001/graphql \
  apps/subgraph-core/src/schema.graphql

# Schema check (CI — detects breaking changes)
hive schema:check \
  --registry.endpoint https://app.graphql-hive.com \
  --registry.accessToken $HIVE_TOKEN \
  --service core \
  apps/subgraph-core/src/schema.graphql
```

yaml

```yaml
# ——— File: supergraph-config.yaml ———
# Hive Gateway supergraph composition config
# Compatible with Apollo Federation v2.7 SDL spec

subgraphs:
  core:
    url: http://subgraph-core:4001/graphql
    schema:
      file: apps/subgraph-core/src/schema.graphql
  content:
    url: http://subgraph-content:4002/graphql
    schema:
      file: apps/subgraph-content/src/schema.graphql
  annotation:
    url: http://subgraph-annotation:4003/graphql
    schema:
      file: apps/subgraph-annotation/src/schema.graphql
  collaboration:
    url: http://subgraph-collaboration:4004/graphql
    schema:
      file: apps/subgraph-collaboration/src/schema.graphql
  agent:
    url: http://subgraph-agent:4005/graphql
    schema:
      file: apps/subgraph-agent/src/schema.graphql
  knowledge:
    url: http://subgraph-knowledge:4006/graphql
    schema:
      file: apps/subgraph-knowledge/src/schema.graphql
```

---

16. Schema Governance & Evolution Rules

Breaking Change Policy

Change Type	Allowed?	Process
Add field to type	✅ Always	PR review
Add optional argument	✅ Always	PR review
Add new type/enum value	✅ Always	PR review
Add required argument	⚠️ With default	PR + migration guide
Rename field	❌ Never	Add new field, deprecate old

Change Type	Allowed?	Process
Remove field	❌ Never (immediately)	Deprecate → 90-day sunset
Change field type	❌ Never	Add new field, deprecate old
Remove enum value	❌ Never (immediately)	Deprecate → 90-day sunset
Change nullability (nullable → non-null)	❌ Output fields	PR review for input fields
Change nullability (non-null → nullable)	✅ Output fields	PR review

Deprecation Lifecycle

1. PR submitted with @deprecated(reason: "Use xyz instead", removalDate: "2026-01-01")
2. Schema check detects usage in persisted queries → notify consuming teams
3. Field continues to work for 90 days
4. After removalDate: field is removed if usage drops to 0
5. If usage > 0 at removalDate: extend by 30 days, notify teams

Naming Conventions

Element	Convention	Example
Types	PascalCase	`MediaAsset`, `AgentExecution`
Fields	camelCase	`createdAt`, `syllabusJson`
Enums	UPPER_SNAKE_CASE	`QUIZ_MASTER`, `AI_GENERATED`
Enum types	PascalCase	`AnnotationLayer`, `MediaType`
Input types	PascalCase + `Input` suffix	`CreateCourseInput`
Filter inputs	PascalCase + `FilterInput` suffix	`CourseFilterInput`
Connection types	PascalCase + `Connection` suffix	`CourseConnection`
Edge types	PascalCase + `Edge` suffix	`CourseEdge`
Mutations	camelCase verb+noun	`createCourse`, `toggleCoursePublished`
Queries (single)	camelCase noun	`course`, `user`
Queries (list)	camelCase plural noun	`courses`, `users`
Subscriptions	camelCase event description	`annotationChanged`, `agentResponseStream`

17. Client Operation Examples

Initialize App (Single Request)

```graphql
query InitializeApp {
  me {
    id
    email
    displayName
    avatarUrl
    role
    preferences {
      language
      theme
      defaultAnnotationLayer
    }
    tenant {
      id
      name
      slug
      plan
      settings {
        defaultLanguage
        allowedLlmProviders
        brandingColors {
          primary
          secondary
        }
      }
    }
  }
}
```

Browse Course with Video Player Context

```graphql
```

```
query CoursePlayerView($courseId: UUID!, $assetId: UUID!, $timestamp: Float!) {
  course(id: $courseId) {
    id
    title
    modules(first: 50, orderBy: { field: ORDER_INDEX, direction: ASC }) {
      edges {
        node {
          id
          title
          orderIndex
          mediaAssets(first: 20) {
            edges {
              node {
                id
                title
                type
                durationSeconds
                thumbnailUrl
              }
            }
          }
        }
      }
    }
  }

  mediaAsset(id: $assetId) {
    id
    title
    hlsManifestUrl
    durationSeconds
    transcript {
      id
      fullText
      language
    }
    segments(first: 50) {
      edges {
        node {
          id
          startTime
          endTime
          text
          speaker
        }
      }
```

```
}
annotations(
  first: 30
  filter: { layers: [PERSONAL, SHARED, INSTRUCTOR, AI_GENERATED] }
) {
  edges {
    node {
      id
      type
      layer
      content
      timestampStart
      timestampEnd
      author {
        id
        displayName
        avatarUrl
      }
    }
  }
}
semanticContextAt(timestamp: $timestamp) {
  activeConcepts {
    concept {
      id
      label
      domain
    }
    connections {
      relatedConcept {
        id
        label
      }
      relationType
      strength
    }
  }
  activeContradictions {
    concept { id label }
    evidence
    severity
  }
  suggestedPrerequisites {
    id
    label
  }
}
```

```
  }
}
```

Execute Agent with Streaming

```graphql
mutation RunChavruta($agentId: UUID!, $annotationId: UUID!) {
  executeAgent(input: {
    agentId: $agentId
    input: { annotationId: $annotationId, mode: "debate" }
  }) {
    id
    status
  }
}

subscription StreamAgentResponse($executionId: UUID!) {
  agentResponseStream(executionId: $executionId) {
    text
    done
    tokensSoFar
  }
}
```

Semantic Search

```graphql
```

```
query SmartSearch($query: String!) {
  hybridSearch(query: $query, first: 10, graphDepth: 2) {
    edges {
      node {
        segment {
          id
          text
          startTime
          endTime
        }
        asset {
          id
          title
          type
          thumbnailUrl
        }
        matchedText
        similarity
        combinedScore
        graphContext {
          concept {
            id
            label
            domain
          }
          connections {
            relatedConcept { id label }
            relationType
            strength
          }
        }
      }
    }
    pageInfo {
      hasNextPage
      endCursor
    }
    totalCount
  }
}
```

---

18. TypeScript Codegen Configuration

GraphQL Code Generator Setup

typescript

```ts
// ——— File: packages/graphql-codegen/codegen.ts ———

import { CodegenConfig } from '@graphql-codegen/cli';

const config: CodegenConfig = {
  schema: [
    'apps/subgraph-core/src/schema.graphql',
    'apps/subgraph-content/src/schema.graphql',
    'apps/subgraph-annotation/src/schema.graphql',
    'apps/subgraph-collaboration/src/schema.graphql',
    'apps/subgraph-agent/src/schema.graphql',
    'apps/subgraph-knowledge/src/schema.graphql',
    'packages/graphql-shared/src/**/*.graphql',
  ],
  documents: [
    'apps/web/src/**/*.graphql',
    'apps/mobile/src/**/*.graphql',
  ],
  generates: {
    // ——— Shared types for all packages ———
    'packages/graphql-types/src/generated/types.ts': {
      plugins: [
        'typescript',
        'typescript-operations',
      ],
      config: {
        scalars: {
          UUID: 'string',
          DateTime: 'string',
          JSON: 'Record<string, unknown>',
          JSONObject: 'Record<string, unknown>',
          Base64: 'string',
          Cursor: 'string',
          URL: 'string',
          EmailAddress: 'string',
          NonNegativeInt: 'number',
          PositiveInt: 'number',
          UnitFloat: 'number',
        },
        enumsAsTypes: false,
        avoidOptionals: false,
        maybeValue: 'T | null',
        strictScalars: true,
        useTypeImports: true,
      },
    },
```

```
// ——— React hooks for web app (using graphql-request or urql) ———
'apps/web/src/generated/graphql.ts': {
  preset: 'client',
  plugins: [
    'typescript',
    'typescript-operations',
    'typescript-react-query',       // TanStack Query hooks
  ],
  config: {
    fetcher: {
      endpoint: 'process.env.VITE_GRAPHQL_ENDPOINT',
      fetchParams: {
        headers: {
          'content-type': 'application/json',
        },
      },
    },
    exposeFetcher: true,
    exposeQueryKeys: true,
    addSuspenseQuery: true,
  },
},

// ——— NestJS resolver types for each subgraph ———
'apps/subgraph-core/src/generated/resolvers.ts': {
  plugins: ['typescript', 'typescript-resolvers'],
  config: {
    federation: true,
    mappers: {
      Tenant: '../entities/tenant.entity#TenantEntity',
      User: '../entities/user.entity#UserEntity',
    },
    contextType: '../context#GraphQLContext',
  },
},
'apps/subgraph-content/src/generated/resolvers.ts': {
  plugins: ['typescript', 'typescript-resolvers'],
  config: {
    federation: true,
    mappers: {
      Course: '../entities/course.entity#CourseEntity',
      Module: '../entities/module.entity#ModuleEntity',
      MediaAsset: '../entities/media-asset.entity#MediaAssetEntity',
      Transcript: '../entities/transcript.entity#TranscriptEntity',
      TranscriptSegment: '../entities/segment.entity#SegmentEntity',
    },
```

```
      contextType: '../context#GraphQLContext',
    },
  },
  // ... (similar for annotation, collaboration, agent, knowledge)

  // ——— Schema introspection for tooling ———
  'packages/graphql-types/src/generated/schema.json': {
    plugins: ['introspection'],
  },

  // ——— Fragment matcher for client cache ———
  'apps/web/src/generated/fragment-matcher.ts': {
    plugins: ['fragment-matcher'],
  },
},
hooks: {
  afterAllFileWrite: ['prettier --write'],
},
};

export default config;
```

NPM Scripts

```bash
# Generate all types
pnpm --filter @edusphere/graphql-codegen exec graphql-codegen

# Watch mode during development
pnpm --filter @edusphere/graphql-codegen exec graphql-codegen --watch

# Check schema compatibility (via Hive CLI)
hive schema:check \
  --registry.endpoint https://app.graphql-hive.com \
  --registry.accessToken $HIVE_TOKEN \
  --service content \
  apps/subgraph-content/src/schema.graphql
```

Full Query/Mutation/Subscription Summary

Queries (44 total)

Subgraph	Query	Auth
Core	`me`	✅
Core	`user(id)`	✅
Core	`users(filter, orderBy, pagination)`	✅
Core	`currentTenant`	✅
Core	`tenantBySlug(slug)`	Public
Content	`course(id)`	✅
Content	`courses(filter, orderBy, pagination)`	✅
Content	`module(id)`	✅
Content	`mediaAsset(id)`	✅
Content	`mediaAssets(filter, orderBy, pagination)`	✅
Content	`segmentsForTimeRange(assetId, start, end)`	✅
Content	`searchTranscripts(query, pagination)`	✅
Annotation	`annotation(id)`	✅
Annotation	`annotations(filter, orderBy, pagination)`	✅
Annotation	`annotationThread(rootId)`	✅
Collaboration	`collabDocument(id)`	✅
Collaboration	`collabDocumentByName(name)`	✅
Collaboration	`collabDocumentsForEntity(type, id)`	✅
Collaboration	`collabConnectionInfo(docId)`	✅
Agent	`agentDefinition(id)`	✅
Agent	`agentDefinitions(filter, orderBy, pagination)`	✅
Agent	`agentTemplates`	✅

Subgraph	Query	Auth
Agent	`agentExecution(id)`	✅
Agent	`agentExecutions(filter, pagination)`	✅
Knowledge	`concept(id)`	✅
Knowledge	`concepts(filter, orderBy, pagination)`	✅
Knowledge	`semanticSearch(query, pagination)`	✅
Knowledge	`hybridSearch(query, graphDepth)`	✅
Knowledge	`relatedConcepts(conceptId, maxDepth)`	✅
Knowledge	`contradictions(conceptId)`	✅
Knowledge	`learningPath(conceptId, maxDepth)`	✅
Knowledge	`topicClusters(pagination)`	✅
Knowledge	`person(id)`	✅
Knowledge	`people(search, pagination)`	✅
Knowledge	`term(id)`	✅
Knowledge	`terms(domain, search, pagination)`	✅
Knowledge	`source(id)`	✅
Knowledge	`sources(type, search, pagination)`	✅

Mutations (44 total)

Subgraph	Mutation	Required Scope
Core	`updateMyProfile`	authenticated
Core	`updateUserRole`	`org:users`
Core	`deactivateUser`	`org:users`
Core	`reactivateUser`	`org:users`
Core	`updateTenantSettings`	`org:manage`
Content	`createCourse`	`course:write`

Subgraph	Mutation	Required Scope
Content	`updateCourse`	`course:write`
Content	`deleteCourse`	`course:write`
Content	`toggleCoursePublished`	`course:write`
Content	`forkCourse`	`course:write`
Content	`createModule`	`course:write`
Content	`updateModule`	`course:write`
Content	`deleteModule`	`course:write`
Content	`reorderModules`	`course:write`
Content	`initiateMediaUpload`	`media:upload`
Content	`completeMediaUpload`	`media:upload`
Content	`updateMediaAsset`	`course:write`
Content	`deleteMediaAsset`	`course:write`
Content	`retriggerTranscription`	`course:write`
Annotation	`createAnnotation`	authenticated
Annotation	`updateAnnotation`	authenticated (owner)
Annotation	`deleteAnnotation`	authenticated (owner)
Annotation	`toggleAnnotationPin`	`annotation:write`
Annotation	`resolveAnnotation`	authenticated
Annotation	`moveAnnotationsToLayer`	`annotation:write`
Collaboration	`createCollabDocument`	authenticated
Collaboration	`compactCollabDocument`	`org:manage`
Agent	`createAgentDefinition`	`agent:write`
Agent	`updateAgentDefinition`	`agent:write` (owner)
Agent	`deleteAgentDefinition`	`agent:write` (owner)

Subgraph	Mutation	Required Scope
Agent	executeAgent	agent:execute
Agent	cancelAgentExecution	authenticated
Knowledge	createConcept	knowledge:write
Knowledge	updateConcept	knowledge:write
Knowledge	deleteConcept	knowledge:write
Knowledge	createRelation	knowledge:write
Knowledge	deleteRelation	knowledge:write
Knowledge	createContradiction	knowledge:write
Knowledge	reindexAssetEmbeddings	knowledge:write
Knowledge	reviewInferredRelation	knowledge:write
Knowledge	createPerson	knowledge:write
Knowledge	updatePerson	knowledge:write
Knowledge	deletePerson	knowledge:write
Knowledge	createTerm	knowledge:write
Knowledge	updateTerm	knowledge:write
Knowledge	deleteTerm	knowledge:write
Knowledge	createSource	knowledge:write
Knowledge	updateSource	knowledge:write
Knowledge	deleteSource	knowledge:write
Knowledge	linkAuthorToSource	knowledge:write
Knowledge	linkSourceToConcept	knowledge:write

Subscriptions (7 total)

Subgraph	Subscription	Trigger
Content	transcriptionStatusChanged(assetId)	Media pipeline

Subgraph	Subscription	Trigger
Content	`transcriptSegmentAdded(assetId)`	Transcription worker
Annotation	`annotationChanged(assetId, layers)`	Annotation CRUD
Collaboration	`collaboratorPresenceChanged(documentId)`	Hocuspocus
Agent	`agentExecutionUpdated(executionId)`	Agent runner
Agent	`agentResponseStream(executionId)`	Agent LLM stream
Knowledge	`conceptsExtracted(assetId)`	NLP pipeline

19. AI/ML Architecture Integration

This section documents how the GraphQL API integrates with the platform's three-layer AI architecture, as specified in the EduSphere Architecture Guide.

Three-Layer AI Architecture

• Knowledge graph integration (Apache AGE Cypher queries)
• HybridRAG: parallel vector + graph retrieval → fused context

GraphQL integration: hybridSearch query, semanticSearch query, semanticContextAt on MediaAsset

Agent Template → LangGraph Workflow Mapping

Each `AgentTemplate` enum value maps to a pre-built LangGraph.js state machine:

Template	LangGraph Workflow	Description
`CHAVRUTA`	`chavruta-debate-graph`	Dialectical debate: present thesis → find contradictions → argue both sides → synthesize. Uses `CONTRADICTS` edges from knowledge graph
`SUMMARIZER`	`summarize-graph`	Progressive summarization: chunk → summarize → merge → condense. Operates on transcript segments
`QUIZ_MASTER`	`quiz-assess-graph`	Adaptive quizzing: assess level → generate questions → evaluate answers → adjust difficulty. Uses `PREREQUISITE_OF` edges
`RESEARCH_SCOUT`	`research-scout-graph`	Cross-reference finder: semantic search → graph traversal → contradiction detection → report. Powers deep research
`EXPLAINER`	`explain-graph`	Adaptive explanation: detect knowledge gaps → explain with analogies → verify understanding. Uses prerequisite chains
`CUSTOM`	User-defined via JSON config	Custom LangGraph state machine loaded from `AgentConfig.systemPrompt` with configurable nodes

MCP (Model Context Protocol) Integration

Agents use MCP for tool integrations. The `AgentConfig.toolsEnabled` field specifies which MCP tools an agent can call:

typescript

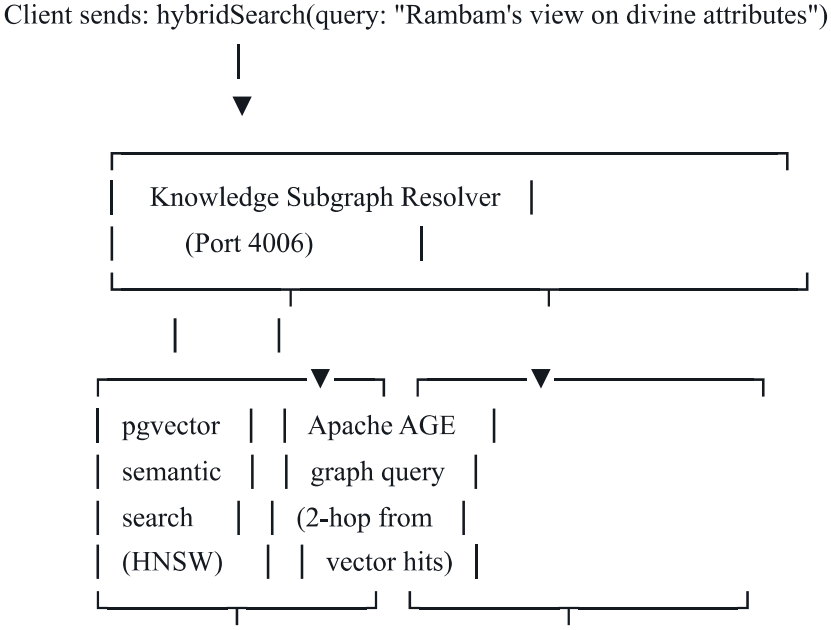
```typescript
// Available MCP tool categories
type MCPToolCategory =
  | 'knowledge_graph'    // Query/create concepts, relations, contradictions
  | 'semantic_search'    // Vector search across content
  | 'transcript_reader'  // Read transcript segments by time range
  | 'annotation_writer'  // Create annotations on behalf of user
  | 'web_search'         // External web search (sandboxed)
  | 'calculator'         // Mathematical computation
  | 'citation_formatter'; // Format academic citations
```

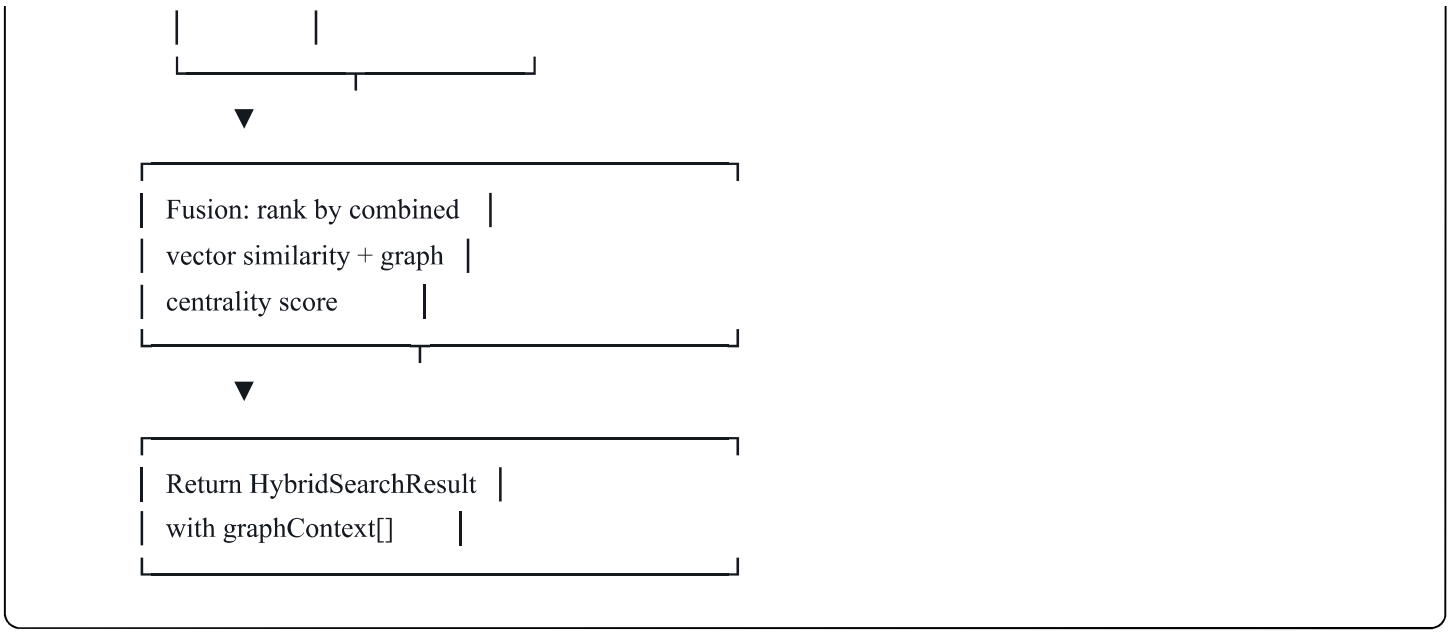
LLM Provider Routing

The Vercel AI SDK provider system is configured per-tenant:

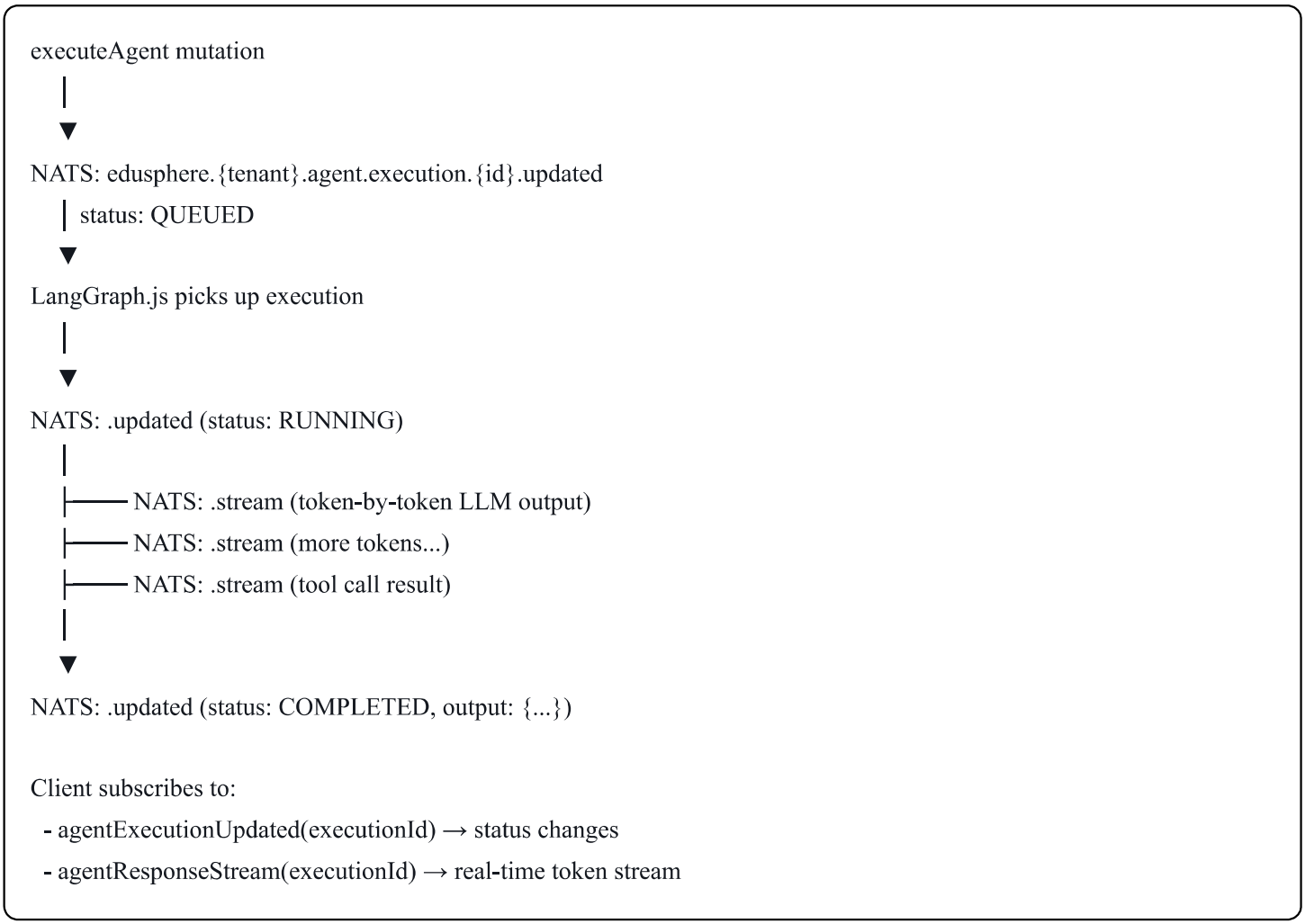
```typescript
// Tenant settings control LLM routing
// Maps to TenantSettings.allowedLlmProviders in the database schema

// Development: Ollama (local)
//   - Llama 3.1 8B (4-6 GB VRAM) — general tutoring
//   - Phi-4 14B (8-10 GB VRAM) — reasoning-heavy explanations

// Production: Cloud providers
//   - OpenAI (gpt-4o, gpt-4o-mini)
//   - Anthropic (claude-sonnet-4-20250514)

// Agent-level override via AgentConfig.modelOverride
// e.g., "phi4:14b" for a specific agent definition
```

HybridRAG Search Flow (GraphQL → Database)





Agent Execution Lifecycle (Subscription Flow)



Agent Sandboxing

For multi-tenant safety, agent executions are sandboxed:

- **gVisor** (Apache 2.0, by Google) provides user-space kernel isolation with only 10-20% performance overhead
- Each agent execution runs in an isolated gVisor container
- Resource limits: CPU, memory, network, and time are capped per-tenant plan

- MCP tool calls are proxy-mediated — agents cannot directly access databases or external services

---

20. Technology Stack Cross-Reference

This table maps every technology choice to its usage in the GraphQL API layer:

Technology	License	GraphQL API Usage
Hive Gateway v2	MIT	Supergraph gateway, query planning, auth enforcement, subscriptions
GraphQL Yoga	MIT	Subgraph HTTP server within each NestJS service
NestJS	MIT	Service framework for all 6 subgraphs
Drizzle ORM	Apache 2.0	Database access in resolvers, RLS via `withTenantContext()`
PostgreSQL 16+	PostgreSQL	Relational data store with RLS policies
Apache AGE	Apache 2.0	Knowledge graph queries (Cypher via `cypher()` function)
pgvector	PostgreSQL	Embedding storage and HNSW semantic search
Keycloak v26	Apache 2.0	JWT issuer, JWKS endpoint for gateway validation
NATS JetStream	Apache 2.0	Event transport for subscriptions, async agent execution
Yjs + Hocuspocus	MIT	CRDT collaboration (CollabDocument, WebSocket URLs)
Vercel AI SDK v6	Apache 2.0	LLM calls in Agent subgraph resolvers
LangGraph.js	MIT	Agent workflow orchestration (state machines)
LlamaIndex.TS	MIT	RAG pipeline, knowledge graph indexing
MinIO	AGPLv3	S3-compatible storage for media uploads (presigned URLs)
faster-whisper	MIT	Transcription pipeline (triggers via NATS events)
Video.js v8	Apache 2.0	Client-side: video player consuming HLS manifests
Konva.js v10	MIT	Client-side: sketch annotation canvas rendering
React + Vite	MIT	Web client consuming the supergraph
TanStack Query v5	MIT	Client-side: GraphQL data fetching and caching
Expo SDK 54	MIT	Mobile client with offline-first patterns
GraphQL Hive	MIT	Schema registry, breaking change detection
Traefik v3.6	MIT	Reverse proxy / ingress controller in production

Database Schema → GraphQL Type Mapping

Database Table	GraphQL Type	Subgraph
tenants	Tenant	Core
users	User	Core
courses	Course	Content
modules	Module	Content
media_assets	MediaAsset	Content
transcripts	Transcript	Content
transcript_segments	TranscriptSegment	Content
annotations	Annotation	Annotation
collab_documents	CollabDocument	Collaboration
crdt_updates	*(internal, not exposed)*	Collaboration
collab_sessions	CollabSession	Collaboration
agent_definitions	AgentDefinition	Agent
agent_executions	AgentExecution	Agent
content_embeddings	*(internal, powers semanticSearch)*	Knowledge
annotation_embeddings	*(internal, powers semanticSearch)*	Knowledge
concept_embeddings	*(internal, powers hybridSearch)*	Knowledge

Apache AGE Graph → GraphQL Type Mapping

AGE Vertex/Edge	GraphQL Type	Notes
Concept vertex	Concept	Primary knowledge graph entity
Person vertex	Person	Authors, speakers, historical figures
Term vertex	Term	Domain-specific terms
Source vertex	Source	External references (books, papers, URLs)

AGE Vertex/Edge	GraphQL Type	Notes
`TopicCluster` vertex	`TopicCluster`	Auto-generated concept groupings
`RELATED_TO` edge	`KnowledgeRelation`	Returned via `relatedConcepts` query
`CONTRADICTS` edge	`Contradiction`	Returned via `contradictions` query
`PREREQUISITE_OF` edge	`PrerequisiteLink`	Returned via `prerequisites` / `dependents`
`MENTIONS` edge	`ConceptMention`	Segment-to-concept temporal links
`CITES` edge	*(via `citedConcepts`)*	Source-to-concept citations
`AUTHORED_BY` edge	*(via `authors`)*	Person-to-source authorship
`INFERRED_RELATED` edge	`KnowledgeRelation`	AI-inferred, pending review
`REFERS_TO` edge	`KnowledgeRelation`	General reference between entities
`DERIVED_FROM` edge	`KnowledgeRelation`	Concept derivation chain
`BELONGS_TO` edge	*(via `topicCluster`)*	Concept-to-cluster membership