ScholarFlow – Project Overview (Detailed Features)

Project Overview

ScholarFlow is a SaaS platform designed for **researchers, students, professors, and academic teams** to **upload, manage, annotate, and collaborate on academic papers**. It leverages **AI-powered summarization, semantic search, and citation tooling** to enhance research workflows. Users can create **workspaces**, organize papers into **collections**, track activities, and collaborate in real-time while maintaining version history of annotations and AI summaries.

Core Features

Paper Management
Upload, import, OCR, parse, tag, and bulk import papers.

Citation & References
Format citations, visualize citation graph, suggest missing citations, and export citations.

Semantic Search & Discovery
Full-text search, embeddings for semantic search, filters, similar papers, and trends.

Collaboration & Annotation
Annotations, version history, collections per workspace, activity logs, and real-time updates.

AI Writing & Research Assistance
Abstract generator, literature review outlining, and self-plagiarism check.

Workspaces & Teams
Multiple workspaces, invite members, role-based access, and workspace activity feed.

Payments & Plans
Freemium, Pro, Institutional plans, payment integrations, and webhook handling.

Backend & Data Management
User, Paper, Annotation, Collection, Workspace, Subscription & Payment, AI embeddings.

API / Integration Highlights
Paper, Annotation, Collection, Workspace, and Search endpoints.

Detailed Features

1. Paper Management
- Upload PDF, DOCX, LaTeX documents.
- Import papers via DOI, arXiv, OpenAlex, Semantic Scholar.
- OCR for scanned documents using Tesseract (Advanced-Optional if time-limited).
- Text parsing to extract title, authors, and abstract.
- AI-based smart tagging (topics, methods) and short summaries.
- Bulk import with progress tracking.

2. Citation & References
- Citation formatting: **APA, MLA, IEEE**.
- **Citation graph visualization** (Advanced-Optional).
- Missing citation suggestions using LLM similarity (Advanced-Optional).
- **Export citations** per collection as bibliography.

3. Semantic Search & Discovery
- Full-text search with **keyword or semantic toggle**.
- **Embeddings for semantic search** using pgvector (Advanced-Optional: vector index for large-scale).
- Filters: workspace, year, tags.
- **Similar papers recommendations**.
- Trends via keyword extraction (Advanced-Optional).

4. Collaboration & Annotation
- **Annotations**: highlights, threaded comments, notes.
- **Version history** of annotations with revert capability.
- **Collections per workspace** with access control: Owner, Editor, Viewer.
- **Activity logs** per collection/workspace.
- **Real-time updates** via WebSocket/SSE (Advanced-Optional for minimal MVP).

5. AI Writing & Research Assistance
- **Abstract generator** (Advanced-Optional).
- **Literature review outlining** (Advanced-Optional).
- **Self-plagiarism check** using similarity across user's papers (Advanced-Optional).

6. Workspaces & Teams
- Multiple workspaces per user.
- Invite members via email.
- Role-based access: Owner, Editor, Viewer.
- Workspace activity feed.

7. Payments & Plans
- **Freemium**: Up to 100 papers, basic AI features, no shared collections.
- **Pro**: Unlimited uploads, AI tools, collaboration (Advanced-Optional for course MVP).
- **Institutional**: Workspace-wide access, SSO (Advanced-Optional).
- Payment integrations: Stripe (global), SSLCommerz (Bangladesh) (Advanced-Optional for MVP).
- Webhook handling for payment updates (Advanced-Optional).

8. Backend & Data Management (Prisma Schema Highlights)
- **User**: profile, role, workspace memberships.
- **Paper**: metadata, file URL, extracted text, tags, summaries.
- **Annotation**: highlights, comments, versions, timestamps.
- **Collection**: workspace-linked, paper links, members.
- **Workspace**: owner, members, settings.
- **Subscription & Payment** (Advanced-Optional).
- **AI embeddings / vector storage** for semantic search (Advanced-Optional).

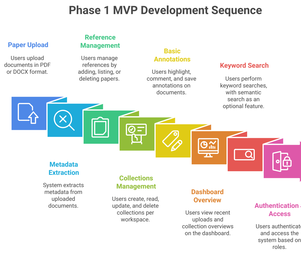
9. API / Integration Highlights
- **Paper endpoints**: upload, import, list/filter, detail, delete.
- **Annotation endpoints**: create, list, version history, revert.
- **Collection endpoints**: CRUD, add/remove papers, invite members.
- **Workspace endpoints**: create, invite, manage roles, activity log.
- **Search endpoints**: semantic search, similar papers, AI summaries (Advanced-Optional).

Phase 1 Prioritized MVP

Phase 1 core MVP should focus on:

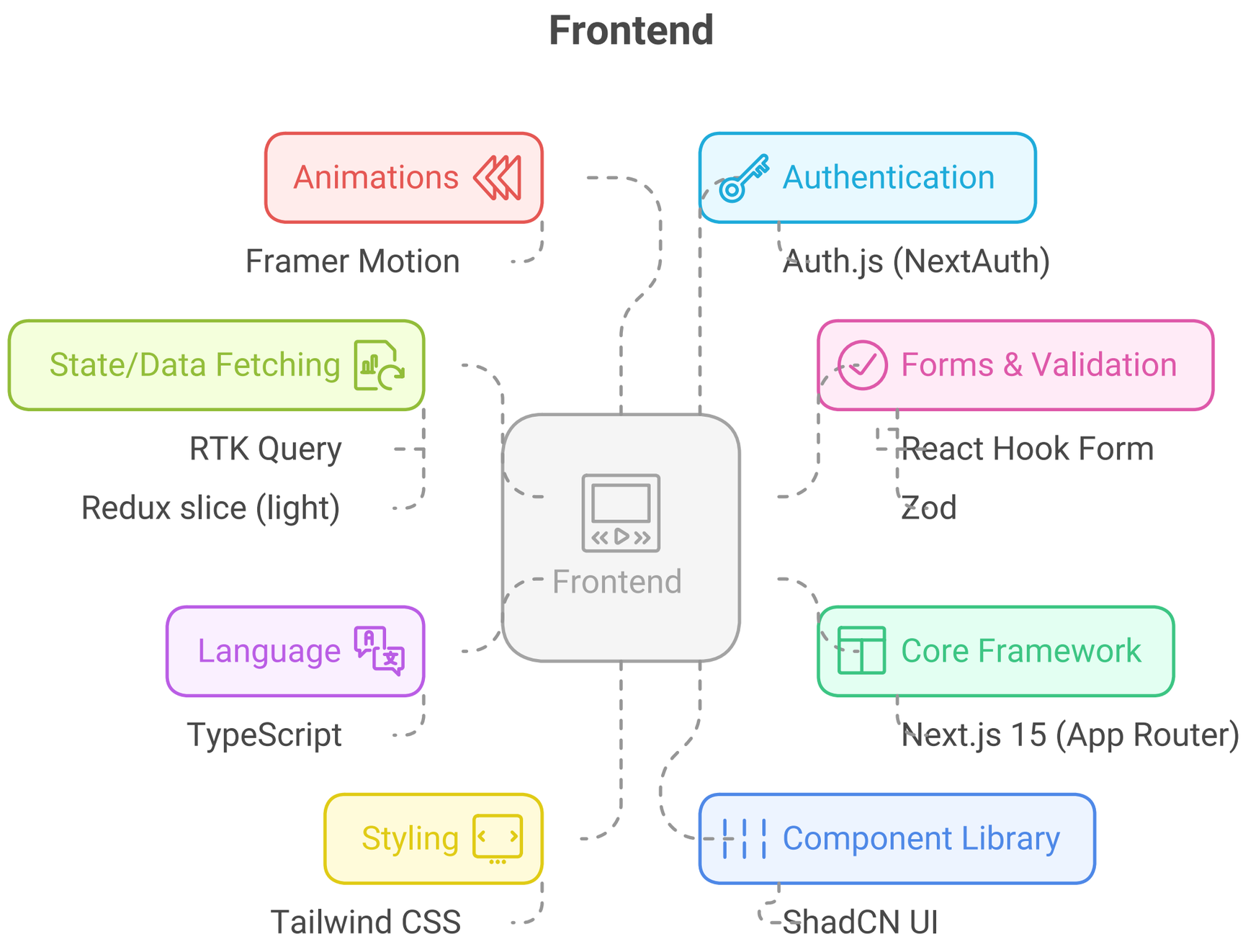
1. Paper upload (PDF/DOCX) + metadata extraction.
2. Simple reference management (add, list, delete papers).
3. Collections per workspace (CRUD + membership).
4. Basic annotations (highlight, comment, save).
5. Dashboard with recent uploads and collection overview.
6. Keyword search (semantic search can be marked Advanced-Optional).
7. Basic authentication & role-based access (JWT + workspace roles).

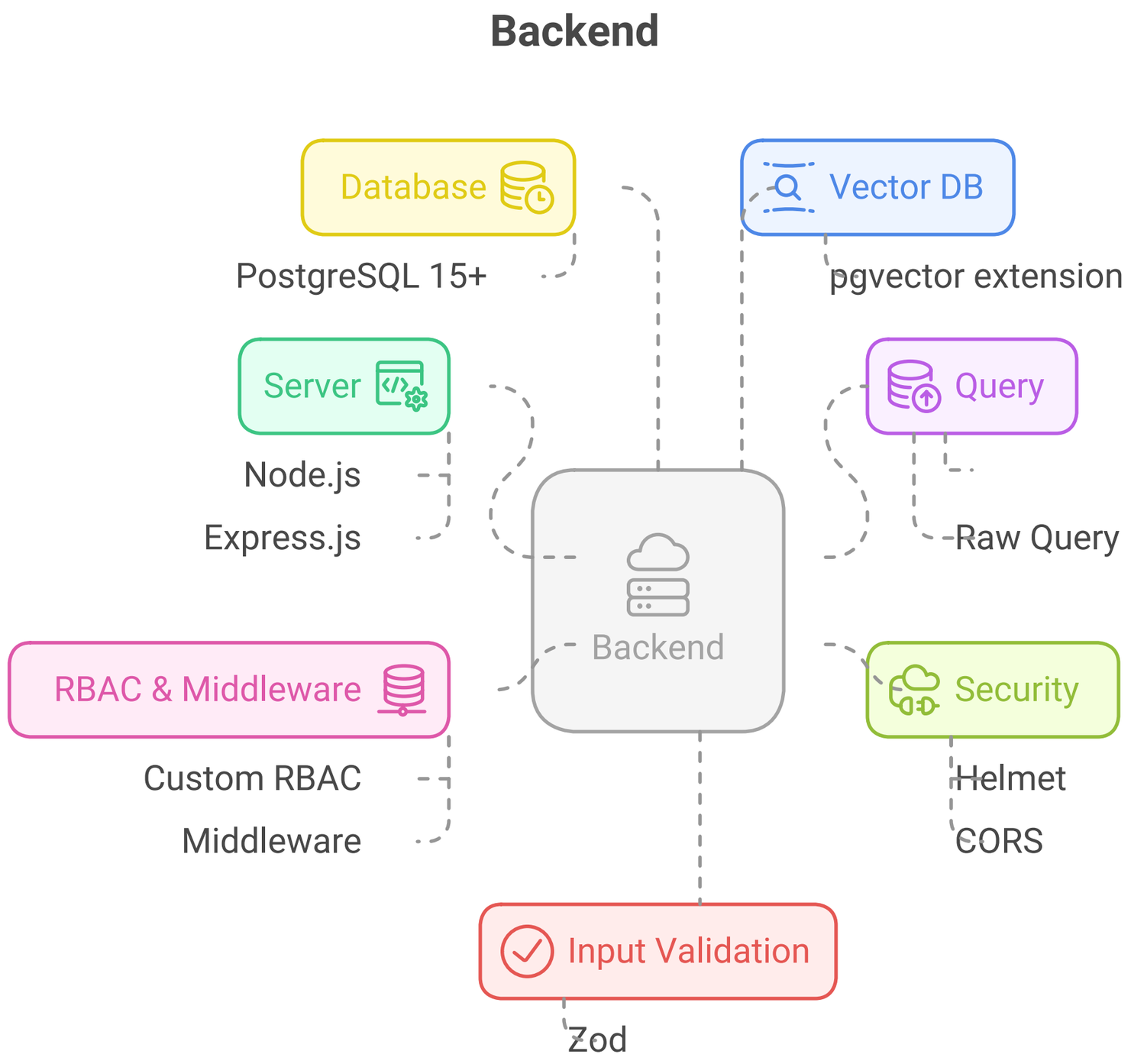
Phase 1 MVP Development Sequence



Advanced/Optional features for post-course or Phase 2:
- AI summarization & abstract generator.
- Citation graph & similarity suggestions.
- Full semantic search with pgvector.
- Real-time updates via WebSocket/SSE.
- Payment, subscription tiers, and institutional SSO.
- Literature review assistance & self-plagiarism detection.

Tech Stack


Frontend
Animations
Framer Motion
Authentication
Auth.js (NextAuth)
State/Data Fetching
RTK Query
Redux slice (light)
Forms & Validation
React Hook Form
Zod
Frontend
Language
TypeScript
Core Framework
Next.js 15 (App Router)
Styling
Tailwind CSS
Component Library
ShadCN UI


Backend
Database
PostgreSQL 15+
Vector DB
pgvector extension
Server
Node.js
Express.js
Query
Raw Query
Backend
RBAC & Middleware
Custom RBAC
Middleware
Security
Helmet
CORS
Input Validation
Zod

Database Relationships Explained

This document outlines the relationships between the different tables in the database, focusing on the entities User, Paper, and their related components. It details the cardinality and constraints of each relationship, including cascading deletes and unique indexes, to ensure data integrity and efficient querying.

User Relationships

The User table is central to the application, and several other tables relate to it.

- **Accounts and Sessions (Auth):** A user can have multiple Account and Session records, which are used for authentication and session management. This is a one-to-many relationship. Deleting a user will cascade the deletion to all associated Account and Session records. This is enforced by the relation configuration on the child side (Account and Session).

> *   `User` 1:N `Account`

> *   `User` 1:N `Session`

- **Uploaded Papers:** A user can upload multiple papers. This is a one-to-many relationship between User and Paper, linked via the Paper.uploaderId field.

> *   `User` 1:N `Paper` (via `Paper.uploaderId`)

- **Annotations:** A user can create multiple annotations. This is a one-to-many relationship between User and Annotation, linked via the Annotation.userId field.

> *   `User` 1:N `Annotation` (via `Annotation.userId`)

- **Collections:** A user can own multiple collections. This is a one-to-many relationship between User and Collection, linked via the Collection.ownerId field.

> *   `User` 1:N `Collection` (via `Collection.ownerId`)

- **Search History:** A user can have multiple search history entries. This is a one-to-many relationship between User and SearchHistory, linked via the SearchHistory.userId field.

> *   `User` 1:N `SearchHistory` (via `SearchHistory.userId`)

- **Payments:** A user can make multiple payments. This is a one-to-many relationship between User and Payment, linked via the Payment.userId field.

> *   `User` 1:N `Payment` (via `Payment.userId`)

Paper Relationships

The Paper table represents a research paper and has several relationships with other tables.

- **Uploader:** Each paper is uploaded by one user. This is a many-to-one relationship between Paper and User, linked via the Paper.uploaderId field. This is the back-relation of the uploadedPapers relationship from the User table. The uploaderId field is indexed for efficient querying.

> *   `Paper` N:1 `User` (via `Paper.uploaderId`)

- **File:** Each paper has one associated file. This is a one-to-one relationship between Paper and PaperFile, enforced by a unique constraint on the paperId field in the PaperFile table.

> *   `Paper` 1:1 `PaperFile` (via `PaperFile.paperId` with unique constraint)

- **Chunks:** Each paper is divided into multiple chunks. This is a one-to-many relationship between Paper and PaperChunk.

> *   `Paper` 1:N `PaperChunk`

- **Annotations:** Each paper can have multiple annotations. This is a one-to-many relationship between Paper and Annotation.

> *   `Paper` 1:N `Annotation`

- **Collections:** A paper can belong to multiple collections. This is a many-to-many relationship between Paper and Collection, realized through the CollectionPaper join table.

> *   `Paper` N:N `Collection` (via `CollectionPaper`)

- **AI Summaries:** Each paper can have multiple AI-generated summaries. This is a one-to-many relationship between Paper and AISummary. The uniqueness of a summary is enforced by a composite unique constraint on (paperId, model, promptHash).

> *   `Paper` 1:N `AISummary`

PaperFile Relationships

The PaperFile table stores the actual file data for a paper.

- **Paper:** Each PaperFile is associated with one Paper. This is a many-to-one relationship between PaperFile and Paper, linked via the PaperFile.paperId field. The paperId field has a unique constraint, enforcing a strict one-to-one pairing with the Paper table.

> *   `PaperFile` N:1 `Paper` (via `PaperFile.paperId` with unique constraint)

PaperChunk Relationships

The PaperChunk table stores individual chunks of a paper's content.

- **Paper:** Each PaperChunk belongs to one Paper. This is a many-to-one relationship between PaperChunk and Paper. A composite unique constraint @@unique([paperId, idx]) ensures that the chunks for a paper are ordered and deduplicated.

> *   `PaperChunk` N:1 `Paper`

Annotation Relationships

The Annotation table stores annotations made on papers.

- **Paper:** Each Annotation is associated with one Paper. This is a many-to-one relationship between Annotation and Paper.

> *   `Annotation` N:1 `Paper`

- **User:** Each Annotation is created by one User. This is a many-to-one relationship between Annotation and User.

> *   `Annotation` N:1 `User`

Collection Relationships

The Collection table stores collections of papers.

- **Owner:** Each Collection is owned by one User. This is a many-to-one relationship between Collection and User, linked via the Collection.ownerId field.

> *   `Collection` N:1 `User` (via `Collection.ownerId`)

- **Papers:** Each Collection can contain multiple Paper records. This is a many-to-many relationship between Collection and Paper, realized through the CollectionPaper join table.

> *   `Collection` N:N `Paper` (via `CollectionPaper`)

CollectionPaper Relationships (Join Table)

The CollectionPaper table is a join table that represents the many-to-many relationship between Collection and Paper.

- **Collection:** Each CollectionPaper entry is associated with one Collection. This is a many-to-one relationship between CollectionPaper and Collection.

> *   `CollectionPaper` N:1 `Collection`

- **Paper:** Each CollectionPaper entry is associated with one Paper. This is a many-to-one relationship between CollectionPaper and Paper.

> *   `CollectionPaper` N:1 `Paper`

A composite unique constraint @@unique([collectionId, paperId]) prevents duplicate associations between a collection and a paper.

SearchHistory Relationships

The SearchHistory table stores the search history of users.

- **User:** Each SearchHistory entry is associated with one User. This is a many-to-one relationship between SearchHistory and User. The table is indexed by (userId, createdAt) for quick lookups of recent history.

> *   `SearchHistory` N:1 `User`

AISummary Relationships

The AISummary table stores AI-generated summaries of papers.

- **Paper:** Each AISummary is associated with one Paper. This is a many-to-one relationship between AISummary and Paper. A composite unique constraint @@unique([paperId, model, promptHash]) prevents storing duplicate summaries for the same paper, model, and prompt variant.

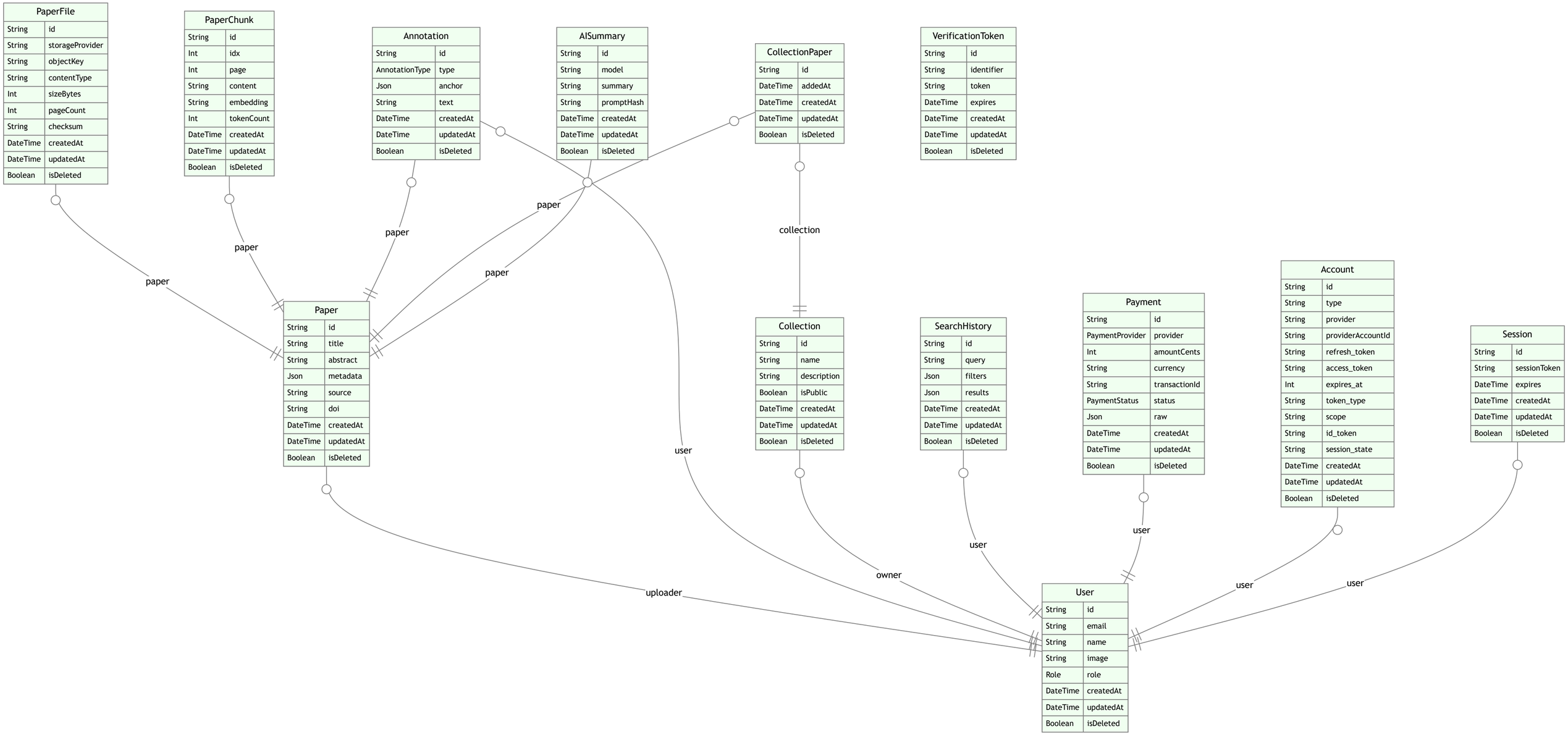
> *   `AISummary` N:1 `Paper`

Payment Relationships

The Payment table stores payment information for users.

- **User:** Each Payment is associated with one User. This is a many-to-one relationship between Payment and User. The transactionId field has a unique constraint to ensure idempotency per provider transaction. The table is indexed by (userId, createdAt) for reporting and history purposes.

> *   `Payment` N:1 `User`


PaperFile
String id
String storageProvider
String objectKey
String contentType
Int sizeBytes
Int pageCount
String checksum
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
PaperChunk
String id
Int idx
Int page
String content
String embedding
Int tokenCount
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Annotation
String id
AnnotationType type
Json anchor
String text
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
AISummary
String id
String model
String summary
String promptHash
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
CollectionPaper
String id
DateTime addedAt
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
VerificationToken
String id
String identifier
String token
DateTime expires
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Paper
String id
String title
String abstract
Json metadata
String source
String doi
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Collection
String id
String name
String description
Boolean isPublic
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
SearchHistory
String id
String query
Json filters
Json results
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Payment
String id
PaymentProvider provider
Int amountCents
String currency
String transactionId
PaymentStatus status
Json raw
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Account
String id
String type
String provider
String providerAccountId
String refresh_token
String access_token
Int expires_at
String token_type
String scope
String id_token
String session_state
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
Session
String id
String sessionToken
DateTime expires
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
User
String id
String email
String name
String image
Role role
DateTime createdAt
DateTime updatedAt
Boolean isDeleted
paper
paper
paper
paper
paper
collection
user
owner
user
user
user
user
uploader