ScholarSphere - AI-Powered Research Paper Collaboration Hub

Updated for Postgres + pgvector, Next.js (App Router) frontend, and a separate Node.js + Express backend with Prisma. This document outlines the architecture, features, and implementation details for ScholarSphere, a SaaS platform designed to help researchers manage academic papers, collaborate with teams, and leverage AI for enhanced workflows.

1) Product Overview

- Name: ScholarSphere
- Type: SaaS platform for research paper organization, collaboration, and AI assistance
- Audience: Researchers, students, professors, academic teams, institutions
- Goal: Help users upload, manage, annotate, and collaborate on academic papers; power workflows with AI (summaries, semantic search, citation tooling, literature review assistance); provide team workspaces and paid plans.

2) System Architecture (Separated Frontend and Backend)

High level

- Frontend Web App (Next.js, Vercel):
  - Next.js App Router (React, TypeScript, Tailwind, ShadCN).
  - Auth handled by Auth.js (NextAuth) with JWT sessions.
  - Calls the backend API with Bearer JWT; uses RTK Query for data fetching.
- Backend API (Node.js, Express, Prisma, Postgres):
  - A dedicated service (Railway/Render/Fly.io). Single source of truth for business logic.
  - Validates Auth.js JWT via shared NEXTAUTH_SECRET (HS256) or JWKS if using asymmetric signing.
  - Exposes REST endpoints for papers, annotations, search, collections, workspaces, billing.
  - Handles payment webhooks (Stripe/SSLCommerz) and writes Payment/Subscription records.
- Worker/Jobs Service (optional but recommended):
  - Handles heavy tasks: file parsing, OCR, chunking, embeddings, AI pipelines.
  - Queue with Redis + BullMQ. Workers consume jobs and persist results via Prisma.
- Database & Storage:
  - Postgres 15+ with pgvector extension; accessed via Prisma.
  - Object storage (S3-compatible) for PDFs and assets; signed URLs from backend.
  - CDN in front of storage for fast delivery.
- Observability & Operations:
  - PostHog/Amplitude for product analytics; OpenTelemetry/logging for API observability.
  - Alerting on queue backlogs, failed webhooks, and AI provider errors.

Data flow highlights

- Upload: Frontend uploads file -> Backend pre-signs S3 URL -> Client uploads -> Backend enqueues “ingest” job -> Worker parses/OCRs -> chunks + embeddings -> records in DB -> UI shows progress.
- Auth: Frontend signs in via Auth.js -> stores JWT -> sends to Backend in Authorization header -> Backend enforces RBAC.
- Billing: Frontend starts checkout -> redirects to provider -> provider posts webhook to Backend -> Backend activates Subscription -> notifies Frontend via polling or SSE/Webhook to client.

3) Core Features

A. Paper Management

- Upload PDFs/DOCX/LaTeX, import via DOI/arXiv/OpenAlex/Semantic Scholar.
- OCR for scanned docs (Tesseract); parse text; extract title/authors/abstract via regex + NLP.
- Smart tagging (topics/methods) via OpenAI/HF; short AI summaries.
- Bulk import with progress tracking.

B. Citation & References

- Citation formatting (APA/MLA/IEEE).
- Citation graph visualization; missing-citation suggestions (LLM + similarity).
- Export citations; per-collection bibliography.

C. Semantic Search & Discovery

- Chunked text (500–1,000 tokens, 50–100 overlap), embeddings per chunk (1536 or 3072 dims).
- pgvector with ivfflat index and cosine distance; filters by workspace, year, tags.
- Similar papers recommender; trends via keyword extraction.

D. Collaboration

- Annotations: highlights, comments, notes; threading; version history and revert.
- Collections per workspace with membership; activity logs; sharing controls.
- Real-time comments/annotation updates via WebSocket/SSE.

E. AI Writing Assistance

- Abstract generator, literature review outliner.
- Self-plagiarism check via cosine similarity across user’s papers.

F. Payments & Plans

- Stripe (global), SSLCommerz (BD). Freemium + Pro + Institutional.
- Webhooks update Payments/Subscriptions; customer portal links for Stripe.

4) Detailed Tech Stack

- Frontend: Next.js (App Router), React, TypeScript, Tailwind, ShadCN, Framer Motion; Forms with RHF + Zod; State with RTK Query + light UI slice.
- Backend API: Node.js, Express.js, Prisma ORM; Zod for input validation; Helmet, CORS; RBAC middleware by role and workspace membership.
- Workers: Node.js with BullMQ (Redis) for ingest/OCR/embeddings/AI tasks.
- Database: Postgres + Prisma; pgvector for embeddings (stored as Unsupported("vector") in Prisma, queried via raw SQL).
- AI: OpenAI, HuggingFace, LangChain, Tesseract.
- Storage: S3-compatible (R2/S3/MinIO) with signed URLs.
- Payments: Stripe SDK; SSLCommerz (sslcommerz-lts).
- DevOps: Vercel (frontend); Railway/Render/Fly.io (backend + workers); Redis for queues/cache.

5) Authentication & Authorization

- Auth.js (NextAuth) on the frontend app; JWT session strategy.
- Backend validates JWT (bearer) with shared secret; extracts userId and role.
- Workspace-based access: WorkspaceMember role gates access to papers/collections inside a workspace.
- Collection-level sharing for granular collaboration.
- Admin users can manage users, payments, and system settings.

6) API Surface (Backend Express)

- Auth
  - POST /auth/session/validate (optional ping) – verifies JWT valid/claims
- Papers
  - POST /papers/upload-url – get pre-signed upload URL
  - POST /papers/import – import by DOI/API providers
  - GET /papers – list/filter; supports semantic=true for vector search
  - GET /papers/:id – detail with metadata
  - DELETE /papers/:id – soft delete
- Annotations
  - GET /papers/:id/annotations
  - POST /papers/:id/annotations
  - GET /papers/:id/annotations/versions
  - POST /papers/:id/annotations/versions/revert
- Collections
  - GET /collections
  - POST /collections
  - GET /collections/:id
  - POST /collections/:id/papers
  - POST /collections/:id/invite
- Search & AI
  - POST /search/semantic
  - GET /papers/:id/similar
  - POST /papers/:id/ai/summarize
  - POST /papers/:id/ai/citation-suggestions
- Graph
  - GET /graph/paper/:id
- Workspaces
  - POST /workspaces
  - POST /workspaces/:id/invite
  - PUT /workspaces/:id/members/:userId
  - GET /workspaces/:id/activities
- Billing
  - GET /subscriptions
  - POST /subscriptions/checkout
  - GET /payments
  - POST /payments/ssl/init
  - POST /webhooks/stripe
  - POST /webhooks/sslcommerz
- Admin
  - GET /admin/users
  - PUT /admin/users/:id
  - GET /admin/metrics

Security and cross-cutting

- CORS: allow frontend domain; block others by default.
- CSRF: not required for pure bearer APIs; ensure cookie usage is httpOnly/secure if used.
- Rate limiting: per-IP and per-user; stricter on AI endpoints.
- Input validation: Zod schemas on every route; sanitize filters for search.
- Audit logs and notifications for sensitive actions.

7) UI/UX – Page-by-Page (Frontend Next.js)

Global Chrome

- Sidebar: Dashboard, Papers, Collections, Upload, Graph, Teams, Billing, Admin
- Header: Global search, notifications, quick actions, profile menu
- Mobile FAB: Upload / New annotation

1. Landing (Marketing)

- Hero CTA, features, pricing, testimonials; analytics events.

2. Auth

- OAuth (Google/ORCID/GitHub) or email magic link; profile completion (name, institution, role).

3. Onboarding

- Upload/import first paper; create workspace and collection; invite teammates; guided tips.

4. Dashboard

- Recent uploads, recommended papers, active collections, notifications; quick actions.

5. Papers Library

- Search bar with keyword/semantic toggle; filters; list/grid; bulk actions (add to collection, export citations).

6. Paper Detail / Reader

- PDF viewer + extracted-text toggle; annotation panel (threaded); AI actions (summarize, cite, paraphrase); versioning and revert.

7. Collections

- List + detail (papers grid, activity, access control); drag-and-drop; invites; export.

8. Semantic Search / Discovery

- Ranked results with similarity score; snippet previews; similar-papers panel; trends.

9. Citation Graph

- Interactive graph (D3/Cytoscape); cluster/filter; node panel with open/cite actions.

10. Upload / Import

- Drag-drop; OCR toggle; import by DOI/URL or providers; background job status.

11. Annotation History

- Version timeline; side-by-side diff; restore.

12. Team / Workspace

- Workspace list and details; members and roles; invites; activity log.

13. Billing

- Plan status; upgrade/checkout; payment history; invoices; customer portal link.

14. Admin

- User management; payments analytics (MRR, churn); system logs; API keys & quotas.

15. Settings / Profile

- Profile, security, integrations (ORCID, Scholar), BYO OpenAI key, data export/delete.

16. Help / Docs

- FAQs, tutorials, support tickets; in-app tooltips.

8) Non-Functional Requirements

- Performance: paginate lists; stream large PDFs; index pgvector with ivfflat (cosine).
- Reliability: idempotent payment webhooks; store raw webhook payloads; retries with backoff.
- Background jobs: all heavy tasks (OCR/embeddings) offloaded; job results written back atomically.
- Security: signed URLs for file access; strict RBAC; input validation; secrets management (runtime env).
- Internationalization: i18n-ready; date-fns for formatting.
- Accessibility: keyboard-friendly annotations; ARIA for viewer controls.

9) Monetization

- Free: up to 100 papers, basic AI features, no shared collections.
- Pro ($10/mo): unlimited uploads, all AI tools, collaboration, priority queue.
- Institutional: workspace/institution-wide seats and SSO (future).

10) Prisma Schema (without ERD generator config)

The canonical source for the ERD. This schema uses UUID string IDs and includes createdAt, updatedAt, isDeleted on every model.

```
// Prisma schema for RefAIra (Postgres + pgvector)
// - All ids are String UUIDs
// - Every model includes createdAt, updatedAt, isDeleted

generator client {
  provider        = "prisma-client-js"
```

```
  provider        = "prisma-client-js"
  previewFeatures = ["postgresqlExtensions"]
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

enum Role {
  RESEARCHER
  PRO_RESEARCHER
  TEAM_LEAD
  ADMIN
}

enum AnnotationType {
  HIGHLIGHT
  COMMENT
  NOTE
}

enum PaymentProvider {
  STRIPE
  SSLCOMMERZ
}

enum PaymentStatus {
  PENDING
  SUCCEEDED
  FAILED
  REFUNDED
}

enum SubscriptionStatus {
  ACTIVE
  EXPIRED
  CANCELED
  PAST_DUE
}

enum PlanTier {
  FREE
  PRO
  INSTITUTIONAL
}

model User {
  id    String  @id @default(uuid())
  email String  @unique
  name  String?
  image String?
  role  Role    @default(RESEARCHER)

  // Auth.js relations
  accounts Account[]
  sessions Session[]

  // App relations
  memberships     WorkspaceMember[]
  workspacesOwned Workspace[]       @relation("WorkspaceOwner")
  uploadedPapers  Paper[]           @relation("PaperUploader")
  annotations     Annotation[]
  collections     Collection[]      @relation("CollectionOwner")
  subscriptions   Subscription[]
  payments        Payment[]
```

```
  searchHistory   SearchHistory[]
  notifications   Notification[]
  usageEvents     UsageEvent[]
  activities      ActivityLog[]

  createdAt         DateTime            @default(now())
  updatedAt         DateTime            @updatedAt
  isDeleted         Boolean             @default(false)
  AnnotationVersion AnnotationVersion[]
  CollectionPaper   CollectionPaper[]
  CollectionMember  CollectionMember[]
}

model Workspace {
  id          String            @id @default(uuid())
  name        String
  ownerId     String
  owner       User              @relation("WorkspaceOwner", fields: [ownerId], references: [id])
  members     WorkspaceMember[]
  collections Collection[]
  papers      Paper[]
  activities  ActivityLog[]

  createdAt    DateTime       @default(now())
  updatedAt    DateTime       @updatedAt
  isDeleted    Boolean        @default(false)
  Subscription Subscription[]
  UsageEvent   UsageEvent[]

  @@index([ownerId])
}

model WorkspaceMember {
  id          String   @id @default(uuid())
  workspaceId String
  userId      String
  role        Role     @default(RESEARCHER)
  joinedAt    DateTime @default(now())

  workspace Workspace @relation(fields: [workspaceId], references: [id])
  user      User      @relation(fields: [userId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([workspaceId, userId])
  @@index([userId])
}

model Paper {
  id          String  @id @default(uuid())
  workspaceId String
  uploaderId  String
  title       String
  abstract    String?
  metadata    Json    @db.JsonB
  source      String? // 'upload' | 'arxiv' | 'openalex' | 'doi' | 'semantic-scholar'
  doi         String? @unique

  uploader        User              @relation("PaperUploader", fields: [uploaderId], references: [id])
  workspace       Workspace         @relation(fields: [workspaceId], references: [id])
  file            PaperFile?
  chunks          PaperChunk[]
  citationsFrom   Citation[]        @relation("CitationsFrom")
```

```
  citationsTo     Citation[]        @relation("CitationsTo")
  annotations     Annotation[]
  collectionJoins CollectionPaper[]
  aiSummaries     AISummary[]

  createdAt  DateTime     @default(now())
  updatedAt  DateTime     @updatedAt
  isDeleted  Boolean      @default(false)
  UsageEvent UsageEvent[]

  @@index([workspaceId, createdAt])
  @@index([uploaderId])
}

model PaperFile {
  id              String  @id @default(uuid())
  paperId         String  @unique
  storageProvider String // s3 | gcs | local
  objectKey       String
  contentType     String?
  sizeBytes       Int?
  pageCount       Int?
  checksum        String?

  paper Paper @relation(fields: [paperId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)
}

model PaperChunk {
  id         String                 @id @default(uuid())
  paperId    String
  idx        Int
  page       Int?
  content    String
  embedding  Unsupported("vector")? // pgvector column (e.g., vector(1536)); ERD does not depend on actual DB
  tokenCount Int?

  paper Paper @relation(fields: [paperId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([paperId, idx])
  @@index([paperId])
}

model Citation {
  id            String  @id @default(uuid())
  sourcePaperId String
  targetPaperId String
  context       String?
  location      String?

  sourcePaper Paper @relation("CitationsFrom", fields: [sourcePaperId], references: [id])
  targetPaper Paper @relation("CitationsTo", fields: [targetPaperId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([sourcePaperId, targetPaperId, context])
  @@index([targetPaperId])
```

```
  @@index([targetPaperId])
}

model Annotation {
  id       String         @id @default(uuid())
  paperId  String
  userId   String
  type     AnnotationType @default(HIGHLIGHT)
  anchor   Json           @db.JsonB // position anchors
  text     String
  version  Int            @default(1)
  parentId String?

  parent   Annotation?  @relation("AnnotationThread", fields: [parentId], references: [id])
  children Annotation[] @relation("AnnotationThread")

  paper    Paper               @relation(fields: [paperId], references: [id])
  user     User                @relation(fields: [userId], references: [id])
  versions AnnotationVersion[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([paperId])
  @@index([userId])
}

model AnnotationVersion {
  id           String   @id @default(uuid())
  annotationId String
  version      Int
  text         String
  changedById  String
  timestamp    DateTime @default(now())

  annotation Annotation @relation(fields: [annotationId], references: [id])
  changedBy  User       @relation(fields: [changedById], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([annotationId, version])
}

model Collection {
  id          String  @id @default(uuid())
  workspaceId String
  ownerId     String
  name        String
  description String?
  isPublic    Boolean @default(false)

  owner     User      @relation("CollectionOwner", fields: [ownerId], references: [id])
  workspace Workspace @relation(fields: [workspaceId], references: [id])

  papers  CollectionPaper[]
  members CollectionMember[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([workspaceId])
  @@index([ownerId])
```

```
}

model CollectionPaper {
  id           String   @id @default(uuid())
  collectionId String
  paperId      String
  addedById    String
  addedAt      DateTime @default(now())

  collection Collection @relation(fields: [collectionId], references: [id])
  paper      Paper      @relation(fields: [paperId], references: [id])
  addedBy    User       @relation(fields: [addedById], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([collectionId, paperId])
  @@index([paperId])
}

model CollectionMember {
  id           String   @id @default(uuid())
  collectionId String
  userId       String
  role         Role     @default(RESEARCHER)
  invitedAt    DateTime @default(now())

  collection Collection @relation(fields: [collectionId], references: [id])
  user       User       @relation(fields: [userId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([collectionId, userId])
}

model SearchHistory {
  id      String @id @default(uuid())
  userId  String
  query   String
  filters Json?  @db.JsonB
  results Json?  @db.JsonB

  user User @relation(fields: [userId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([userId, createdAt])
}

model AISummary {
  id         String @id @default(uuid())
  paperId    String
  model      String
  summary    String
  promptHash String

  paper Paper @relation(fields: [paperId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)
```

```
  isDeleted Boolean  @default(false)

  @@unique([paperId, model, promptHash])
}

model Plan {
  id            String  @id @default(uuid())
  code          String  @unique
  name          String
  priceCents    Int
  currency      String
  interval      String // month, year, etc.
  stripePriceId String?
  features      Json?   @db.JsonB
  active        Boolean @default(true)

  subscriptions Subscription[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)
}

model Subscription {
  id                     String             @id @default(uuid())
  userId                 String
  workspaceId            String?
  planId                 String
  status                 SubscriptionStatus
  provider               PaymentProvider
  providerCustomerId     String?
  providerSubscriptionId String?
  cancelAtPeriodEnd      Boolean            @default(false)
  startedAt              DateTime           @default(now())
  expiresAt              DateTime

  user      User       @relation(fields: [userId], references: [id])
  workspace Workspace? @relation(fields: [workspaceId], references: [id])
  plan      Plan       @relation(fields: [planId], references: [id])
  payments  Payment[]

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([userId])
  @@index([workspaceId])
  @@index([planId])
}

model Payment {
  id             String          @id @default(uuid())
  userId         String
  subscriptionId String?
  provider       PaymentProvider
  amountCents    Int
  currency       String
  transactionId  String          @unique
  status         PaymentStatus
  raw            Json?           @db.JsonB

  user         User          @relation(fields: [userId], references: [id])
  subscription Subscription? @relation(fields: [subscriptionId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
```

```
  isDeleted Boolean  @default(false)

  @@index([userId, createdAt])
}

model WebhookEvent {
  id          String          @id @default(uuid())
  provider    PaymentProvider
  eventId     String
  type        String
  payload     Json            @db.JsonB
  receivedAt  DateTime        @default(now())
  processedAt DateTime?
  status      String
  error       String?

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([provider, eventId])
}

model Notification {
  id      String    @id @default(uuid())
  userId  String
  type    String
  payload Json      @db.JsonB
  readAt  DateTime?

  user User @relation(fields: [userId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([userId, createdAt])
}

model UsageEvent {
  id          String  @id @default(uuid())
  userId      String
  workspaceId String?
  kind        String // "upload", "ai_summarize", "semantic_search", etc.
  units       Int     @default(1)
  paperId     String?

  user      User       @relation(fields: [userId], references: [id])
  workspace Workspace? @relation(fields: [workspaceId], references: [id])
  paper     Paper?     @relation(fields: [paperId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([userId, createdAt])
}

model ActivityLog {
  id          String  @id @default(uuid())
  userId      String?
  workspaceId String?
  entity      String
  entityId    String
  action      String
```

```
  details     Json?   @db.JsonB

  user      User?     @relation(fields: [userId], references: [id])
  workspace Workspace? @relation(fields: [workspaceId], references: [id])

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@index([workspaceId, createdAt])
}

// -------- Auth.js (NextAuth) models --------

model Account {
  id                String  @id @default(uuid())
  userId            String
  type              String
  provider          String
  providerAccountId String
  refresh_token     String? @db.Text
  access_token      String? @db.Text
  expires_at        Int?
  token_type        String?
  scope             String?
  id_token          String? @db.Text
  session_state     String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([provider, providerAccountId])
  @@index([userId])
}

model Session {
  id           String   @id @default(uuid())
  sessionToken String   @unique
  userId       String
  expires      DateTime

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)
}

model VerificationToken {
  id         String   @id @default(uuid())
  identifier String
  token      String   @unique
  expires    DateTime

  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  isDeleted Boolean  @default(false)

  @@unique([identifier, token])
}
```

11) Deployment

- Frontend: Vercel project with environment-bound NEXTAUTH_SECRET, NEXT_PUBLIC_API_BASE_URL.
- Backend: Railway/Render/Fly.io container; set DATABASE_URL, JWT/Auth secrets, Stripe/SSL keys, S3 creds.
- Workers: Same platform as backend, separate process using the same codebase (monorepo) or separate repo.
- Migrations: Prisma migrate on backend deploy; maintain seed scripts for local/dev environments.

12) Next Steps

- Implement Auth.js JWT strategy; add backend middleware to validate tokens and attach user context.
- Build upload flow with S3 pre-signed URLs and background ingest job.
- Implement vector search queries using pgvector + raw SQL; add ivfflat index.
- Wire Stripe/SSLCommerz checkout and webhooks; create Subscription/Payment write paths.
- Harden RBAC at workspace and collection boundaries; add activity logging for sensitive flows.

If you want, I can generate a base Express project structure (controllers, routers, middleware, Prisma client, BullMQ setup) and a Next.js starter wired to this API.