Learnify by EduHub — Final System Design & Roadmap (Next.js 15, JS, Postgres)

AI-powered course metasearch for students & developers. Aggregates MOOCs (Coursera, edX, Udemy, SWAYAM/NPTEL, etc.), normalizes metadata with OpenAI, and recommends the best 3–5 results per query.

---

0) Confirmed Stack (Your Choices)

- **Framework:** Next.js 15 (App Router, Server Actions)
- **Language: JavaScript** (no TypeScript)
- **Auth:** Clerk
- **DB:** PostgreSQL (**with** `pgvector` for embeddings)
- **ORM:** Prisma
- **Cache/Queues:** Redis
- **AI:** OpenAI (embeddings + LLM for normalization/reranking/summaries)
- **Payments:** Razorpay (UPI, RuPay, cards) — Stripe later (phase 4)
- **Deploy:** Vercel (web). Background worker on Railway/Fly.io/Render

> We drop Meilisearch to stay minimal. Retrieval is **Postgres FTS (tsvector/tsquery) + pgvector** for semantic rerank. This keeps infra lean and AI-friendly.

---

1) High-Level Architecture

```
[Browser]
  ↕ Clerk auth widgets
  ↕ Next.js routes & Server Actions
[Next.js 15 (Vercel)]
  • API routes + Server Actions (BFF)
  • Search flow: cache → Postgres FTS → pgvector similarity → rules rerank
  • If data staleness detected → enqueue provider crawl job
↘️
[Worker (Node on Railway/Fly.io)]
  • Provider connectors (Coursera/Udemy/edX/SWAYAM/NPTEL)
  • Page/API fetch → normalize with OpenAI → compute embeddings
  • Upsert Postgres
[PostgreSQL]
  • Core relational data + `pgvector` embeddings + FTS indexes
[Redis]
  • Cache (search results), rate limits, BullMQ queues
[Razorpay]
  • Checkout + webhooks → update subscription state
```

```
[OpenAI]
  • Embeddings + LLM normalization/rerank/summaries
```

---

2) Monorepo Folder Structure (JavaScript)

```
learnify-eduhub/
├─ apps/
│  ├─ web/                          # Next.js 15 (App Router)
│  │  ├─ app/
│  │  │  ├─ (marketing)/
│  │  │  │  ├─ page.jsx              # Landing
│  │  │  │  ├─ pricing/page.jsx
│  │  │  │  └─ docs/page.jsx
│  │  │  ├─ (app)/                   # Authenticated area
│  │  │  │  ├─ search/page.jsx
│  │  │  │  ├─ compare/[id]/page.jsx
│  │  │  │  ├─ saved/page.jsx
│  │  │  │  └─ settings/page.jsx
│  │  │  ├─ api/
│  │  │  │  ├─ search/route.js       # POST search
│  │  │  │  ├─ courses/route.js      # admin ingest controls
│  │  │  │  ├─ webhook/razorpay/route.js
│  │  │  │  └─ clerk/                 # (optional) custom hooks
│  │  ├─ components/
│  │  │  ├─ SearchBar.jsx
│  │  │  ├─ Filters.jsx
│  │  │  ├─ CourseCard.jsx
│  │  │  ├─ CompareTable.jsx
│  │  │  └─ Header.jsx/Footer.jsx
│  │  ├─ lib/
│  │  │  ├─ auth.js                  # Clerk helpers
│  │  │  ├─ prisma.js                # Prisma client (singleton)
│  │  │  ├─ cache.js                 # Redis helpers
│  │  │  ├─ scoring.js               # Ranking formula
│  │  │  ├─ ai/
│  │  │  │  ├─ normalize.js          # LLM metadata normalizer
│  │  │  │  ├─ embeddings.js         # compute/store vectors
│  │  │  │  └─ prompts.js
│  │  ├─ hooks/ styles/ types/
│  │  └─ tests/
│  │
│  └─ worker/                        # Background ingestion/indexing
│     ├─ src/
```

```
│     │  ├─ jobs/
│     │  │  ├─ crawlCoursera.js
│     │  │  ├─ crawlUdemy.js
│     │  │  ├─ crawlEdx.js
│     │  │  ├─ crawlSwayam.js
│     │  │  ├─ normalizeUpsert.js
│     │  │  └─ refreshEmbeddings.js
│     │  ├─ queues.js
│     │  ├─ env.js
│     │  └─ index.js                # BullMQ worker bootstrap
│     └─ package.json
│
├─ prisma/
│  ├─ schema.prisma
│  └─ migrations/
│
├─ scripts/                          # seed, utilities
├─ docker-compose.yml                # Postgres(pgvector)+Redis
├─ .env.example
├─ .github/workflows/
└─ README.md
```

---

3) Database Schema (Prisma + pgvector)

```
// prisma/schema.prisma

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

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

enum Role { USER ADMIN }

enum Provider { COURSERA EDX UDEMY SWAYAM NPTEL OTHER }

enum Level { BEGINNER INTERMEDIATE ADVANCED MIXED }

model User {
  id          String   @id @default(cuid())
  clerkId     String   @unique
```

```
  email       String   @unique
  name        String?
  role        Role     @default(USER)
  preferences Json?
  plan        String?  // free | pro
  planEndsAt  DateTime?
  saved       SavedCourse[]
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}

model Course {
  id            String   @id @default(cuid())
  provider      Provider
  sourceId      String?
  url           String   @unique
  title         String
  subtitle      String?
  description   String?
  language      String?
  topics        String[]
  level         Level?
  price         Float?
  currency      String?
  isFree        Boolean  @default(false)
  rating        Float?
  ratingCount   Int?
  credential    Boolean  @default(false)
  providerCred  Int?
  durationHrs   Float?
  effortPerWk   Float?
  publishedAt   DateTime?
  thumbnailUrl  String?
  vector        Bytes?   // embedding (pgvector)
  lastSeenAt    DateTime @default(now())
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt

  @@index([provider])
  @@index([level])
  @@index([isFree])
  @@index([title])
}

model SavedCourse {
  id        String   @id @default(cuid())
  userId    String
  courseId  String
```

```
  notes     String?
  createdAt DateTime @default(now())

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

  @@unique([userId, courseId])
}

model SearchEvent {
  id        String   @id @default(cuid())
  userId    String?
  query     String
  filters   Json?
  resultIds String[]
  clickedId String?
  createdAt DateTime @default(now())
}
```

Migration for `pgvector`

```
CREATE EXTENSION IF NOT EXISTS vector;
ALTER TABLE "Course" ADD COLUMN IF NOT EXISTS vector vector(1536);
CREATE INDEX IF NOT EXISTS course_vector_idx ON "Course" USING ivfflat (vector
vector_l2_ops);
```

FTS index (title/description)

```
ALTER TABLE "Course" ADD COLUMN IF NOT EXISTS tsv tsvector;
CREATE INDEX IF NOT EXISTS course_tsv_idx ON "Course" USING GIN(tsv);
-- trigger to keep tsv in sync
CREATE OR REPLACE FUNCTION course_tsv_update() RETURNS trigger AS $$
BEGIN
  NEW.tsv := to_tsvector('english', coalesce(NEW.title,'') || ' ' ||
coalesce(NEW.description,''));
  RETURN NEW;
END;$$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS course_tsv_tg ON "Course";
CREATE TRIGGER course_tsv_tg BEFORE INSERT OR UPDATE ON "Course"
FOR EACH ROW EXECUTE PROCEDURE course_tsv_update();
```

---

4) Search & Ranking Pipeline (Server Action)

1. **Parse Query** → lightweight NLP to extract intent (level, free/paid, language)
2. **Cache Check (Redis)** → `search:{hash(query+filters)}` TTL 30–60m
3. **Lexical Retrieve (Postgres FTS)** → `tsquery` over `title/description`
4. **Vector Similarity (pgvector)** → cosine/L2 between query embedding and `Course.vector`
5. **Rules-based Rerank** (in `lib/scoring.js`):
6. Relevance = α*(FTS score)* + *β*(cosine similarity)
7. Quality = rating + log(ratingCount) + recency decay
8. Credibility = providerCred + credential
9. Value = price fit (free boost) + duration fit
10. Personalization = matches vs user prefs/history
11. **Top K (5–10)** → return to client; write `SearchEvent`
12. **Freshness** → if `lastSeenAt` too old for providers in the result set → enqueue crawl

---

5) Payments (Razorpay Subscriptions)

Plan design - `free` → limited filters & daily search cap - `pro` → advanced filters, compare view, AI summaries

Flow 1. User clicks **Upgrade** → hit `/api/billing/create-subscription` → create Razorpay subscription + order 2. Frontend opens Razorpay Checkout with order id 3. On success/failure → Razorpay hits **webhook** → `/app/api/webhook/razorpay/route.js` 4. Verify signature → update `User.plan` + `planEndsAt` in Postgres

Webhook security - Verify `X-Razorpay-Signature` using webhook secret - Idempotency: store processed event ids

---

6) API Surface (Next.js Routes)

- `POST /api/search` → { query, filters } → ranked `Course[]`
- `GET /api/courses/[id]` → course details
- `POST /api/courses/ingest` (admin) → { provider } → enqueue crawl
- `POST /api/billing/create-subscription` → returns Razorpay order
- `POST /api/webhook/razorpay` → handle events (payment.authorized, subscription.charged, etc.)

Prefer **Server Actions** from pages (`search/page.jsx`) for internal calls; keep routes for webhooks/worker.

---

7) Caching, Rate Limits, Freshness

- Cache search results & course detail payloads (Redis)
- Token bucket per IP/user for `/api/search`
- Staleness policy: popular queries recrawled daily; long-tail weekly
- Exponential backoff & circuit breakers on connectors

---

8) Compliance & Scraping Policy

- Prefer official APIs/feeds; follow ToS & robots.txt
- Do not bypass paywalls/login; store only metadata; always deep-link
- Consider affiliate/partner programs when available

---

9) Docker (Local Dev)

```yaml
version: "3.9"
services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: learnify
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
    command: ["postgres", "-c", "shared_preload_libraries=vector"]

  redis:
    image: redis:7
    ports:
      - "6379:6379"
    volumes:
      - redisdata:/data

volumes:
  pgdata:
  redisdata:
```

---

10) Environment Variables (.env.example)

```
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/learnify
REDIS_URL=redis://localhost:6379
OPENAI_API_KEY=
CLERK_PUBLISHABLE_KEY=
CLERK_SECRET_KEY=
RAZORPAY_KEY_ID=
RAZORPAY_KEY_SECRET=
NEXT_PUBLIC_APP_URL=http://localhost:3000
NODE_ENV=development
```

---

11) Development Roadmap

Phase 0 — Project Setup (1 week) - Init repo (Next.js 15, JS). Add Tailwind, shadcn/ui. Configure Clerk. - Add Prisma + Postgres. Run migrations for schema + pgvector + FTS trigger. - Add Redis client + basic cache helper. Seed minimal course data.

Phase 1 — Core Search (2–3 weeks) - Build Search UI (SearchBar, Filters, CourseCard) - Implement `/api/search` + Server Action: FTS → vector rerank → rules - Track SearchEvent, click logging, basic analytics

Phase 2 — Ingestion & Normalization (1–2 weeks) - Implement 2 provider connectors (e.g., Udemy + SWAYAM) - LLM normalization (title, topics, duration estimate), embeddings - Admin page to trigger re-ingest; dedupe by canonical URL

Phase 3 — Save/Compare & Paywall (1–2 weeks) - Saved lists, Compare view - Razorpay subscription flow + webhook; gate premium features

Phase 4 — Quality & Scale (ongoing) - Provider credibility model, recency decay tuning - Personalization (boost by user prefs/history) - Observability: Sentry, logs, latency budgets - Add Stripe for international later

---

12) Security & Testing

- Clerk session validation in Server Actions & routes
- Input validation (lightweight zod optional even in JS)
- Row-level checks (user owns saved items)
- Webhook signature verification + idempotency
- Tests: unit (scoring), integration (search pipeline), E2E (Playwright)

---

13) UI Highlights

- Query chips auto-detected (Beginner, Free, Hindi)
- Badges: Free, Certificate, University-backed
- Compare table: syllabus, duration, price, rating, certificate
- AI summaries of pros/cons per course (Pro plan)

---

14) Next Steps

1) Create repo from this structure 2) Add Docker & run Postgres/Redis locally 3) Wire Clerk; scaffold Prisma + run migrations (pgvector + FTS) 4) Build search Server Action + scoring; instrument logging 5) Implement first connector E2E; ship internal MVP