Vedagyan — End‑to‑End EdTech Platform with Integrated LMS (Proposal & SRS)

Client: Vedagyan  
Date: 14-08-2025

0) Executive Summary

Vedagyan will evolve into a scalable, secure, and engaging ed‑tech platform tailored to Indian classical arts, devotional learning, and public speaking. Built on a Next.js + Node.js (Express) + MongoDB stack, the platform enables learners to discover, purchase, and consume structured courses, while empowering instructors to create, manage, and monetize their content.

This proposal adapted to Vedagyan’s course catalog and branding. It includes student, instructor, and admin panels, robust LMS features (courses, modules, lessons, quizzes, ratings, certificates), a clean, mobile‑first UI, secure payments, and analytics. (Per scope, no live classes or calendar are included in this phase; can be added later.)

Key Outcomes

* Immersive learning for students with clear paths and progress tracking.
* Efficient course creation & management for instructors.
* Operational control, quality assurance, and insights for admins.

1) Scope & Course Catalog (Vedagyan)

Course Categories

1. Classical Dance — *Bharatanatyam, Kuchipudi, Kathak*
2. Classical Music — *Carnatic Vocals, Hindustani Vocals*
3. Devotional & Cultural Arts — *Bhajans & Slokas, Mythological Storytelling*
4. Public Speaking — *Articulation & Confidence*

Each course supports: trailer video, rich description, learning outcomes, prerequisites, instructor bio, syllabus (sections → lessons), downloadable resources (PDF/notes), practice tasks, quizzes, ratings/reviews, and completion certificates.

2) System Architecture (High Level)

Client–Server model with REST APIs.

* Front‑end: Next.js (React), App Router, SSR/SSG for SEO, Client Components for interactivity. Styling: Tailwind CSS. State: Redux Toolkit (or Zustand where suitable). Forms: React Hook Form + Zod. Media players for video/audio.
* Back‑end: Node.js (Express), REST APIs, JWT auth, role‑based access (Student/Instructor/Admin), file streaming, rate‑limiting.
* Database: MongoDB Atlas with Mongoose ODM.
* Media: Cloudinary for images/videos/documents (upload, transform, secure delivery).
* Payments: Razorpay checkout, webhooks for order → enrollment.
* Caching/Queues (Phase‑2 optional): Redis for caching hot reads; BullMQ for jobs (emails, transcoding triggers, webhooks).
* Email/OTP: Nodemailer + transactional provider (e.g., Resend/SMTP).
* Observability: Winston + Morgan logs; Sentry for error tracking; analytics events pipeline.
* Security: HTTPS, JWT + refresh tokens, bcrypt password hashing, CORS, input validation, file type whitelisting, signed URLs, audit trails.

Deployment

* Front‑end: Vercel (Next.js) with environment‑based configs.
* Back‑end: Render/Railway (auto‑deploy from Git, zero‑downtime).
* DB: MongoDB Atlas.
* Media: Cloudinary.
* CI/CD: GitHub Actions (lint, test, build, deploy).

3) Role‑Based Product Features

A) Student Panel

* Discovery: Home, search, filters (category, level, language, rating), trending/new, instructor cards.
* Course Pages: trailer, syllabus, outcomes, resources, reviews, FAQs, instructor profile.
* Commerce: wishlist, cart, coupon support, Razorpay checkout, GST invoice PDF.
* Learning Space (LMS): continue learning, sections/lessons, video/audio player (resume position), transcripts/lyrics, downloadable PDFs, practice tasks, lesson notes, bookmarks.
* Assessments: lesson quizzes, module tests, final quiz; instant feedback; retakes policy.
* Progress: lesson completion toggles, overall % progress, streaks, certificates on completion.
* Community: per‑course Q&A, comments, rating/review.
* Profile: personal info, password, order history, enrolled courses, certificates, settings.
* Notifications: email + in‑app for enrollments, new lessons, replies.
* Support: help center, ticket form.

B) Instructor Panel

* Dashboard: revenue, enrollments, course ratings, top courses, recent reviews.
* Course Builder: create/edit course → sections → lessons; drag‑drop reorder; markdown/RTF content; upload video/audio/thumbnail/PDF to Cloudinary; set pricing, coupons, tags, difficulty; preview.
* Content Types:
  + Video (dance demos, vocal lessons),
  + Audio (alap, swaras, slokas/chanting),
  + PDFs/notations/lyrics (swaras, bols, sahityam),
  + Text/markdown (context, assignments).
* Assessments: quiz bank, timed quizzes, question types (MCQ, single/multi‑select, true/false), explanations, randomization.
* Student Engagement: respond to Q&A, pin answers, announcements.
* Monetization: pricing, coupons, payouts summary (admin‑managed settlement).
* Profile: bio, expertise, links, media gallery.

C) Admin Panel

* Overview: users, enrollments, revenue, course approvals, top categories.
* Catalog Ops: review/approve courses, category & tag management, feature courses, reorder home sections.
* Quality: content moderation (flagged content/reviews), takedown flow.
* Users & Roles: create/suspend users, reset passwords, instructor onboarding KYC.
* Payments: orders, refunds, coupons, payout reports.
* CMS: pages (About, FAQs, Policies), banners, emails.
* Analytics: funnel (visit → view → add‑to‑cart → purchase), retention, course performance, NPS.
* Settings: tax/GST, email provider, media limits, rate limits, feature flags.

4) Front‑end Architecture & UI/UX

* Next.js App Router with mixed rendering (SSR for catalog/SEO, SSG for static pages, CSR for LMS player).
* Design System: Tailwind CSS utility‑first, accessible components, responsive grid, dark mode.
* State & Data: Redux Toolkit Query for API caching; React Query alternative acceptable.
* Forms & Validation: React Hook Form + Zod schema validation, inline errors.
* Internationalization (optional): i18n ready.
* Accessibility: WCAG‑AA contrast, focus states, keyboard navigation, captions/subtitles.
* Performance: image optimization, code‑splitting, lazy media, prefetching, CDN.

Core Screens (Students)  
Home, Category/Catalog, Course Details, Cart/Checkout, Dashboard (My Courses), Player (sections/lessons), Quiz, Notes, Profile, Orders, Certificates.

Core Screens (Instructors)  
Dashboard, Courses list, Course Builder (details, curriculum, media, pricing, publish), Quiz manager, Q&A, Profile.

Core Screens (Admin)  
Overview, Users, Courses (approval), Categories/Tags, Orders/Payments, Reviews/Flags, Coupons, CMS Pages, Settings, Analytics.

5) Back‑end Architecture

* Framework: Node.js (Express).
* Auth: Email/password with OTP verification; JWT access + refresh tokens; password reset; role guard middleware.
* Payments: Razorpay orders, verification via signature; webhooks for capture/refund; server‑side invoice creation; map to enrollment.
* Media: Cloudinary SDK uploads from server (signed) or client (signed preset); MIME/type checks; size limits.
* Search & Filters: text index on title/description; filter by category/level/price/rating.
* Reviews: 1 review per enrolled student per course; rating aggregate stored/updated.
* Progress Tracking: per‑lesson completion, last position (ms), quiz scores.
* Certificates: generate PDF with unique ID & QR verification.
* Notifications: templated emails, optional in‑app notifications collection.
* Security: input sanitization, rate limits, helmet, CORS, audit logs.

Data Models (high‑level)

* User {name, email, role, passwordHash, otp, avatar, bio, expertise[], social, isActive}
* Course {title, slug, category, subcategory, description, outcomes[], level, language, price, thumbnail, trailer, instructor, sections[], tags[], status}
* Section {title, lessons[]}
* Lesson {title, type(video|audio|pdf|text), mediaUrl, duration, transcript, resources[], freePreview}
* Quiz {courseId, sectionId, questions[], passingScore}
* Question {type(MCQ|MSQ|TF), prompt, options[], answer, explanation}
* Enrollment {userId, courseId, orderId, startedAt, progress%, completedAt, certificateId}
* Order {userId, items[], subtotal, tax, total, status, provider, providerRef}
* Review {courseId, userId, rating, text, createdAt}
* Certificate {id, courseId, userId, issuedAt, verificationCode}
* Announcement/QnA {courseId, authorId, body, replies[]}
* Coupon {code, discountType, value, expiry, usageLimit}

6) API Design (REST)

Auth

* POST /api/auth/signup — create user (student/instructor)
* POST /api/auth/login — issue JWT
* POST /api/auth/verify-otp — verify email
* POST /api/auth/forgot-password — email reset link
* POST /api/auth/reset-password — set new password

Catalog

* GET /api/courses — list with filters/sort/pagination
* GET /api/courses/:id — course details (public fields)
* GET /api/categories — categories & subcategories

Instructor

* POST /api/courses — create course (instructor)
* PUT /api/courses/:id — update
* DELETE /api/courses/:id — delete
* POST /api/courses/:id/publish — submit/approve (admin gate)
* POST /api/courses/:id/sections — add section
* POST /api/courses/:id/lessons — add lesson (signed upload)
* POST /api/courses/:id/quiz — manage quizzes

Commerce & Enrollment

* POST /api/cart — add/remove items
* POST /api/orders — create Razorpay order
* POST /api/payments/webhook — provider webhook
* GET /api/enrollments/me — my enrollments

Learning & Engagement

* GET /api/player/:courseId — secured lesson manifest
* POST /api/progress — update lesson completion/time
* POST /api/courses/:id/reviews — add review (enrolled‑only)
* GET /api/certificates/:id — verify certificate
* POST /api/courses/:id/qna — questions & replies

Admin

* GET /api/admin/overview
* POST /api/admin/courses/:id/approve|reject
* GET /api/admin/orders — refunds, payouts
* POST /api/admin/coupons
* GET/PUT /api/admin/settings

Conventions: JSON, standard HTTP verbs, pagination via limit/offset (or cursor), 201 on create, meaningful error codes, input schemas via Zod/JOI.

7) Testing Strategy

* Unit Tests: services, controllers, utils (Jest).
* API Tests: supertest for endpoints; schema validation snapshots.
* Integration: MongoDB memory server for flows (signup→purchase→enroll).
* E2E: Playwright for core journeys (discover, checkout, learn, quiz, review).
* Accessibility: axe automated checks; manual keyboard tests.
* Performance: Lighthouse budgets; player page LCP/CLS targets.
* Security: dependency checks, OWASP linters, penetration test (phase‑2).

8) AI Models in we can use :

1. AI Learning Agent (Personal AI Mentor)

Description:  
An intelligent agent that acts as a personal coach for every student, understanding their pace, learning style, and strengths/weaknesses.

Key Capabilities:

* Continuously tracks learning progress via LMS activity data.
* Adapts teaching styles — e.g., visual aids for visual learners, practice tests for analytical learners.
* Motivates students with reminders, challenges, and rewards.
* Provides personalized lesson summaries and reinforcement exercises.

System Architecture:

* Frontend: Student dashboard with AI Mentor interface (React + WebSockets for real-time updates).
* Backend: AI engine (Python, FastAPI/Flask) integrated with LMS database.
* Data Sources: Student progress records, assessment scores, time spent per module, engagement metrics.
* Model Type: Reinforcement Learning (RL) + Natural Language Processing (NLP) for dialogue.
* Storage: PostgreSQL for structured data, MongoDB for unstructured notes.

Dataflow:

1. Input: Student activity logs, quiz results, interaction history.
2. Processing: AI Mentor analyzes patterns using RL + user profile embeddings.
3. Decision: Suggests next learning activity, modifies content delivery style.
4. Output: Personalized recommendations and guidance to student UI.

2. AI-Powered Tutor Chatbot

Description:  
An AI assistant available 24/7 to clarify concepts, simulate practice scenarios, and provide contextual explanations.

Key Capabilities:

* NLP-based query understanding (multi-language support).
* Dynamic answer generation using context from LMS materials.
* Scenario-based simulations for practical application of knowledge.
* Integrated with voice and text communication.

System Architecture:

* Frontend: Embedded chat widget in web & mobile apps.
* Backend: LLM (Large Language Model) API + retrieval-augmented generation (RAG) layer connected to course content database.
* Data Sources: Course documents, video transcripts, student queries.
* Model Type: Fine-tuned LLM (e.g., OpenAI GPT-4, Llama-3) with domain-specific corpus.

Dataflow:

1. Input: Student enters question (text/voice).
2. Processing: NLP pipeline → query parsing → RAG fetches relevant materials → LLM generates context-aware answer.
3. Decision: If query matches existing FAQ, fetch direct answer; else generate via model.
4. Output: Rich answer with examples, links, and follow-up prompts.

3. AI Assessment & Feedback System

Description:  
An automated grading assistant that evaluates student work with speed, consistency, and personalized improvement tips.

Key Capabilities:

* Auto-grading MCQs, and subjective answers.
* Feedback that pinpoints exact weaknesses.
* Progress tracking over time to detect learning plateaus.

System Architecture:

* Frontend: Assessment submission panel, feedback reports.
* Backend: Scoring engine with ML models for subjective answers (BERT-based text similarity), test case evaluation for code, statistical scoring for MCQs.
* Data Sources: Submitted answers, reference solutions, grading rubrics.
* Model Type: NLP (text grading), Computer Vision (handwriting recognition if needed), and rule-based scoring.

Dataflow:

1. Input: Student submissions.
2. Processing:
   * Objective questions: direct evaluation.
   * Subjective questions: NLP semantic similarity + rubric matching.
3. Decision: Generate scores and highlight improvement areas.
4. Output: Score report with actionable feedback delivered to LMS dashboard.

4. AI Recommendation Engine

Description:  
A dynamic content suggestion system to guide learners toward optimal resources and skills.

Key Capabilities:

* Learns from past performance, interaction preferences, and peer learning patterns.
* Suggests videos, practice exercises, and advanced topics.
* Updates recommendations in real-time as student progress changes.

System Architecture:

* Frontend: Recommendation feed inside LMS home screen.
* Backend: Collaborative filtering + content-based filtering hybrid model.
* Data Sources: Student profiles, content metadata, engagement analytics.
* Model Type: Matrix factorization (for collaborative) + NLP embeddings (for content matching).

Dataflow:

1. Input: Student’s profile, course completion rate, and activity logs.
2. Processing:
   * Collaborative filtering: compares with similar learners.
   * Content-based filtering: matches content tags to learning goals.
3. Decision: Generate top-N ranked list of recommendations.
4. Output: Displayed as “Suggested for You” section in dashboard.
5. **AI Tutor – Acharya (Multilingual Support)**

The platform will introduce **“Acharya”**, an AI-powered tutor designed to act as a **virtual guide for students learning classical arts, Sanskrit, Slokas, and other specialized subjects**.

**Key Capabilities:**

* **Conversational Tutoring:** Students can interact with Acharya in real-time, asking questions and receiving answers tailored to their level.
* **Multilingual Support:** Acharya will support **Sanskrit, Hindi, English**, and can be extended to other regional languages to make learning inclusive.
* **Cultural Relevance:** For Sanskrit and classical courses, Acharya will incorporate **contextual explanations**, pronunciations, and relevant examples from literature.
* **Adaptive Learning:** Just like the AI Learning Agent, Acharya will adapt teaching style based on student’s proficiency and progress.
* **Voice Interaction:** Optional **speech-to-text & text-to-speech** features will allow students to **speak** in their preferred language and receive **spoken explanations**.

**6. Advanced Responsive & Scalable Design**

To ensure a **seamless experience across devices**, the system will be optimized for:

* **Big Screens & Smart TVs:** Students and teachers can access the platform on **large screens** for an immersive classroom-like experience.
* **Adaptive UI:** Interface elements (fonts, layouts, and controls) will **auto-adjust** to fit desktops, tablets, mobile devices, and smart boards.
* **Scalable Infrastructure:** The platform will be capable of handling **up to 1 lakh concurrent users**, with scaling handled dynamically on the cloud.

8) Deployment & Environments

* Envs: Dev, Staging, Production with separate keys (Razorpay, Mongo, Cloudinary).
* CI/CD: PR checks → staging preview (Vercel) → manual promote to prod.
* Backups: MongoDB daily snapshots; artifact retention.
* Monitoring: Uptime checks, Sentry alerts, log retention.

9) Policies & Compliance

* Privacy & T&C pages configurable via Admin CMS.
* Data: GDPR‑style rights (export/delete account), India data residency consideration.
* Payments: GST invoice fields, refund policy workflows.
* Content: copyright attestation in course publish flow; takedown handling.

10) Future Enhancements (Roadmap)

1. Personalized learning paths (high) — adaptive sequencing, recommendations by skill & interest.
2. Gamification (medium) — badges, points, streaks, leaderboards.
3. Mobile Apps (high) — offline playback, background audio for vocals/slokas.
4. ML recommendations (medium‑high) — next lesson/course suggestions.
5. Practice tools (medium) — tala/tabla metronome, pitch drone for vocals.
6. Advanced moderation (medium) — AI‑assisted content/review moderation.
7. AR/VR (low‑medium) — immersive dance posture guidance, virtual classroom showcases.
8. Org/B2B (medium) — cohorts for schools/cultural institutions.

11) Acceptance Criteria (Samples)

* Students can search, purchase via Razorpay, get enrolled, and consume lessons with progress saved across devices.
* Instructors can create a multi‑section course with at least 1 video, 1 audio, and 1 PDF, add a quiz, publish for admin approval, and receive first enrollment.
* Admin can approve/reject courses, issue coupon, see revenue and enrollment metrics.
* Reviews permitted only after enrollment; certificate issued upon passing criteria.

12) Assumptions & Exclusions (Phase‑1)

* No live classes or calendar in Phase‑1.
* Video hosting via Cloudinary (not custom DRM).
* Single‑currency pricing (INR) initially; multi‑currency later.
* English UI; multi‑language optional.

13) Timeline & Deliverables (Indicative)

* Week 1–2: IA/UX, design system, auth & catalog schema.
* Week 3–4: Catalog pages (SSR), course details, cart/checkout (Razorpay).
* Week 5–7: LMS player, progress, quizzes, reviews.
* Week 8–9: Instructor builder, media uploads, publish workflow.
* Week 10: Admin console, CMS, coupons, analytics.
* Week 11: Testing, hardening, accessibility & performance passes.
* Week 12: Staging UAT, production launch.

14) Conclusion

This Proposal & SRS defines a production‑ready Vedagyan LMS, updated for Next.js + Node.js and Vedagyan’s dance, music, devotional, and public speaking catalog. The architecture, APIs, data models, and deployments are designed for scalability, security, and great UX. Subsequent phases can extend personalization, gamification, mobile apps, and immersive practice tools to deepen learner engagement.

Appendix: Sample API Requests/Responses (Abbreviated)

Create Course (Instructor)

POST /api/courses

{

"title": "Bharatanatyam Basics",

"category": "Classical Dance",

"subcategory": "Bharatanatyam",

"price": 1999,

"description": "Foundational adavus and posture.",

"level": "Beginner"

}

Response 201: { "id": "...", "status": "draft" }

Enroll via Razorpay Webhook

POST /api/payments/webhook

{ "event": "payment.captured", "payload": { "order": {"id": "order\_abc"}, "payment": {"id": "pay\_xyz"} } }

Response 200: { "ok": true } → creates Order + Enrollment.

Update Progress

POST /api/progress

{ "courseId": "...", "lessonId": "...", "watchedMs": 420000, "completed": true }

Response 200: { "progress": 0.35 }

15) AI System Architecture (All Models)

This section details how all AI features integrate with the Vedagyan LMS: the AI Learning Agent (Mentor), AI Tutor Chatbot, AI Assessment & Feedback, and the AI Recommendation Engine. It includes components, deployment topology, and request/response dataflows.

15.1 Components & Topology

[Client Apps]

Web (Next.js) / Mobile (future)

|

v

[API Gateway]

Express (Node.js) — auth, rate limit, routing

|

+--> [LMS Core Services]

| Courses, Users, Orders, Progress (MongoDB)

|

+--> [AI Orchestrator]

| gRPC/HTTP to AI Microservices, feature flags

|

+--> [Event Bus / Queue]

| BullMQ/Redis — async jobs (ingestion, grading, emails)

|

+--> [File/Media]

| Cloudinary (video/audio/pdf), transcripts

|

+--> [Observability]

Sentry, Logs, Metrics

[AI Microservices]

1) Tutor Chatbot (LLM + RAG)

2) Learning Agent (Mentor)

3) Assessment & Feedback

4) Recommendation Engine

[AI Data Stores]

- MongoDB (users, courses, progress, grades)

- Vector DB (FAISS/Weaviate/Pinecone) for content embeddings

- Object Store (Cloudinary) for transcripts & static artifacts

- Redis for caching session state and short‑lived context

Notes

* RAG Index is built from course materials: descriptions, lesson text, PDF notes, and generated video/audio transcripts.
* AI Orchestrator centralizes auth, quotas, safety filters, and routes to the right AI microservice.
* Event Bus decouples heavy tasks (transcription, embedding, auto‑grading) from real‑time user requests.

15.2 Cross‑Cutting Concerns

* Security & Privacy: PII minimization, field‑level encryption (emails), signed URLs, per‑role data access, audit logs for AI decisions.
* Safety: content filters on LLM outputs; prompt injection defenses; rate limiting per user/IP.
* Evaluation: offline eval sets for grading & chatbot, A/B testing for recommendations, human‑in‑the‑loop flags.
* Internationalization: multilingual tokenization & translation layer (phase‑2).

15.3 Data Pipelines

1. Content Ingestion
   * Trigger: instructor publishes/updates a course.
   * Steps: fetch text (markdown), generate transcripts for video/audio → clean/segment → embed → upsert into Vector DB with metadata (courseId, sectionId, tags, level).
   * Outputs: searchable RAG index + updated content catalog.
2. Learning Signals Stream
   * Events: lesson start/finish, watch time, quiz score, revisit rate, notes created, Q&A posts.
   * Storage: time‑series collection (MongoDB) + aggregate rollups (daily/weekly) for mentor & recommender.
3. Assessment Queue
   * Submissions posted to queue → Assessment service grades → results stored → feedback posted to student & instructor.

15.4 AI Tutor Chatbot (LLM + RAG)

Goal: High‑quality, course‑aware answers for dance, vocals, slokas, and public speaking.

Request Flow

1. User asks a question in chat → API Gateway → AI Orchestrator.
2. Orchestrator builds context: user role, enrolled courses, current lesson, language.
3. Retriever queries Vector DB with query embedding + filters (courseId, sectionId).
4. LLM Generation (provider or self‑hosted) with system & safety prompts; includes retrieved chunks.
5. Safety & citation checks → response streamed to client; feedback buttons log quality.

Key Stores: Vector DB (content), Redis (conversation state), MongoDB (chat logs, feedback).

Core Prompts: domain style (classical arts), pedagogy tone, structured answers with examples, and optional step‑by‑step practice.

15.5 AI Learning Agent (Personal Mentor)

Goal: Adaptive guidance, motivation, and next‑best‑action for each learner.

Signals: progress %, quiz accuracy, dwell time, replays/rewinds, skipped lessons, preferred modality (video/audio/text), difficulty flags.

Decision Model

* Policy engine (rules + bandits/RL) suggests: revisit fundamentals, attempt quiz, try practice task, switch modality (audio mantra vs. video demo).
* NLP templates personalize messages; goals tracked per learner.

Flow

1. Nightly job aggregates signals → feature vectors per learner.
2. Mentor service computes recommendations & encouragement messages.
3. Notifications saved; shown on dashboard and sent via email if opted‑in.
4. Student actions feed back to improve policy (explore/exploit).

15.6 AI Assessment & Feedback

Goal: Consistent, fast, rubric‑based evaluation for objective & subjective items.

Objective (MCQ/TF): server‑side scoring with item analysis (discrimination, difficulty).

Subjective (text/audio/video):

* Text: semantic similarity to reference, key‑point detection, rubric alignment.
* Audio (Vocals/Chanting): pitch tracking & timing variance (DSP) + rubric comments.
* Video (Dance Posture – phase‑2): pose estimation (keypoints) vs. reference sequences (privacy‑aware, opt‑in).

Flow

1. Submission stored → enqueue grading job.
2. Grader runs per item type → produces score + targeted feedback bullets.
3. Results persisted → LMS notifies learner + instructor.
4. Analytics: mastery by skill, weak‑area clusters for recommender.

15.7 AI Recommendation Engine (Hybrid)

Goal: Suggest the right next lesson, revision, or resource.

Signals: learner features from 15.5, cohort behavior, content metadata (tags: raga, tala, adavus, storytelling themes), recency & novelty.

Model: Hybrid = Collaborative filtering (matrix factorization/nearest neighbors) + Content‑based (embedding similarity). Business rules for diversity & cold‑start.

Flow

1. Batch trainer updates factors nightly; online layer reranks per session.
2. API returns Top‑N items with reasons (“Because you struggled with Alarippu footwork”).
3. Click/complete signals loop back as implicit feedback.

15.8 API Surface (AI)

* POST /api/ai/chat — { message, courseId?, sectionId? } → streaming answer + citations
* GET /api/ai/recs — personalized Top‑N recommendations
* POST /api/ai/mentor/nudge — generate dashboard nudges (admin/instructor)
* POST /api/ai/grade — submit assessment for auto‑grading (async)
* GET /api/ai/grade/:submissionId — grading status/result

Webhooks

* /api/webhooks/transcripts.ready — when transcripts are generated
* /api/webhooks/embeddings.upserted — RAG index updated
* /api/webhooks/payments.captured — link order→enrollment→signals init

15.9 Data Schema Additions (AI)

* ChatSession {userId, context, startedAt, turns[], satisfaction}
* EmbeddingChunk {courseId, sectionId, contentId, vectorRef, tokens, lang, updatedAt}
* SignalEvent {userId, courseId, type, value, ts}
* Recommendation {userId, items[], generatedAt, algoVersion}
* AssessmentSubmission {userId, courseId, items[], mediaRefs[], rubricId, status, score, feedback[]}
* MentorPlan {userId, goals[], nextActions[], lastEvaluatedAt}

15.10 Sequence Diagrams (ASCII)

A) Chatbot RAG Answer

Client -> API Gateway : POST /api/ai/chat

API Gateway -> Orchestrator : validate+route

Orchestrator -> Vector DB : semantic search (filters)

Orchestrator -> LLM : prompt + retrieved chunks

LLM -> Orchestrator : streamed tokens

Orchestrator -> Client : SSE stream (answer+cites)

B) Auto‑Grading Submission

Client -> API Gateway : POST /api/ai/grade (files+answers)

Gateway -> Queue : enqueue submission

Queue -> Assessment Svc : job pull

Assessment -> Storage : fetch refs/transcripts

Assessment -> LMS DB : write score+feedback

LMS -> Client : notify + GET /api/ai/grade/:id

15.11 Non‑Functional Targets (AI)

* Latency: Chat first token < 1.5s (p95) with streaming; recs API < 250ms (p95).
* Throughput: 50 RPS sustained on chat with autoscaling; grading batch 10k submissions/day.
* Cost Controls: cache hits > 40%; prompt truncation; embedding dedupe; nightly budget caps.
* Quality KPIs: answer helpfulness (CSAT), quiz mastery uplift, completion rate delta.

**16. Project Timeline & Milestones**

**Total Duration:** 3 Months (12 Weeks)  
**Total Project Cost:** ₹6,83,000  
**Payment Structure:** 30% – 40% – 30%

**Milestone 1 – Project Kick-off & Core Foundations**

**Duration:** 4 Weeks  
**Payment:** ₹2,04,900 (30% of total)  
**Deliverables:**

* **Requirement Gathering & Finalization**
  + Final confirmation on all features, course categories, and UI/UX references.
  + Detailed project plan with finalized scope and timelines.
* **UI/UX Design**
  + Wireframes and mockups for all panels (Student, Instructor, Admin).
  + Brand guidelines implementation (colors, fonts, iconography).
* **Frontend Development Setup**
  + Next.js project setup with modular architecture.
  + Common reusable components (Header, Footer, Navigation, Buttons, Inputs).
* **Backend Development Setup**
  + Node.js + Express.js API architecture setup.
  + Database schema design (MongoDB).
  + Authentication & Authorization (JWT + Role-based).
* **Core API Development**
  + User registration/login.
  + Profile management.
  + Course categories listing API.

**Milestone 2 – Feature Development & Integrations**

**Duration:** 5 Weeks  
**Payment:** ₹2,73,200 (40% of total)  
**Deliverables:**

* **Student Panel Features**
  + Course listings & search filters.
  + Course details page.
  + Enrollment & payment integration (Razorpay/Stripe).
  + Video player integration for lessons.
  + Progress tracking & quizzes.
* **Instructor Panel Features**
  + Course creation & management.
  + Upload videos, PDFs, assignments.
  + Student performance analytics.
* **Admin Panel Features**
  + User management.
  + Course approval & moderation.
  + Payment & revenue reports.
* **Additional Integrations**
  + Email notifications & OTP verification.
  + Cloud storage integration for media (AWS S3/Cloudinary).
  + Review & rating system.
  + Discussion forums.
* **Testing Phase 1**
  + Unit testing for backend APIs.
  + Functional testing for major workflows.

**Milestone 3 – Finalization, Testing & Deployment**

**Duration:** 3 Weeks  
**Payment:** ₹2,04,900 (30% of total)  
**Deliverables:**

* **Final UI Enhancements**
  + Responsive design optimization.
  + Accessibility compliance check.
* **Full System Testing**
  + End-to-end testing across all panels.
  + Bug fixing and performance optimization.
* **Security Enhancements**
  + SSL integration.
  + Server hardening and data encryption.
* **Deployment**
  + Deployment on cloud server (AWS/DigitalOcean).
  + Database & API hosting setup.
  + CDN configuration for faster content delivery.
* **Handover & Training**
  + Admin training session.
  + Documentation (API docs + admin usage guide).

**9. Notes**

* The above system architecture is designed to handle **up to 1,00,000 concurrent users** comfortably.
* Actual performance will depend on the **chosen cloud server configuration and scaling strategy**.
* Scope changes or additional features outside of the agreed plan will be **handled as change requests** with separate costing.
* All media content will be stored on **secure cloud storage** to ensure scalability and quick delivery.