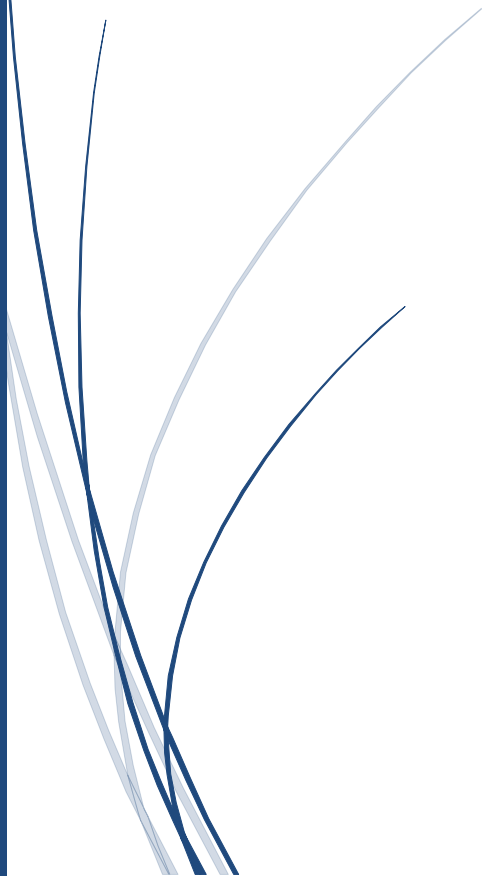
10/1/2025

Shekhai LMS

Backend Documentation (REST API, MongoDB)

Comprehensive Developer Guide, Data Models, APIs, and Admin Operations (Updated)



Generated for: Shekhai (shekhai.ngengroup.org)
Tech stack: Node.js, Express, Mongoose (MongoDB), JWT

Contents

1. Executive Summary 3

2. Architecture Overview 3

3. Authentication & Authorization 5

4. Users (Students & Instructors) 5

5. Courses, Modules & Lessons 6

   5.1 Course Data Model 6

   5.2 Module/Lesson Data Model 7

6. Categories & Tags 7

7. Enrollments & Progress 7

8. Quizzes, Assignments & Grading 8

   8.1 Quiz Model 8

   8.2 Assignment Model 8

9. Project Showcase Module 8

   9.1 Data Model (projects) 9

   9.2 Endpoints 9

   9.3 Admin & Mentor Actions 9

   9.4 Business Logic & Notes 10

10. Mentor Room Module 10

   10.1 Data Model (mentor_rooms & bookings) 10

   10.2 Booking Model 10

   10.3 Endpoints 11

   10.4 Admin & Mentor Actions 11

   10.5 Business Logic & Notes 11

11. Webinar Room Module 11

   11.1 Data Model (webinars) 12

   11.2 Endpoints 12

   11.3 Business Logic & Notes 12

12. Community Module 12

   12.1 Data Models (posts, comments, reactions) 13

   12.2 Endpoints 13

   12.3 Moderation & Business Rules 13

13. Payments & Orders 13
14. Media & Assets 13
15. Notifications & Real-time 13
16. Admin Panel & Operations 14
17. Search, Filtering & Indexing 14
18. Backups, Migrations & Seeding 14
19. Security & Compliance 14
20. Deployment & Environment Variables 14
21. API Reference — Selected Endpoints 15
22. Example Mongoose Schemas 16

1. Executive Summary

This document provides a comprehensive backend specification for Shekhai LMS (shekhai.ngengroup.org).
It is designed for developers, architects, and DevOps engineers responsible for implementing and maintaining the Node.js + Express REST API powered by MongoDB (Mongoose). The specification covers data models, CRUD endpoints, validation rules, admin workflows, payment integrations, and real-time communication strategies.

This release includes extended modules such as:

- Project Showcase (for student portfolio presentation)
- Mentor Room (for mentor-student collaboration and sessions)
- Webinar Room (for live and recorded sessions)
- Community Module (for peer interaction and discussions)

The goal is to ensure a scalable, maintainable, and secure backend infrastructure ready for production deployment.

2. Architecture Overview

The Shekhai LMS platform follows a modular microservice-style architecture built on a RESTful Node.js + Express API, communicating with a React/Tailwind frontend. It's optimized for horizontal scalability, security, and developer productivity.

Layer	Technology	Responsibility
Client (Frontend)	React + TailwindCSS	SPA/UI consuming REST APIs
API Layer	Node.js + Express	Core business logic and route handling
Database	MongoDB (via Mongoose ODM)	Data persistence and relational mapping
Auth	JWT (Access + Refresh tokens)	Role-based access control and secure sessions
Storage	Amazon S3 (or MinIO)	File and media storage (images, videos, docs)
Background Jobs	Redis + BullMQ (or RabbitMQ)	Asynchronous task processing
Real-time Communication	Socket.IO + WebRTC	Webinars, chat, notifications
CDN / Edge Delivery	CloudFront or equivalent	Optimized media delivery
Monitoring	PM2 + Winston + LogDNA	Logging, uptime, and analytics

Service boundaries and responsibilities include Auth, Course Management, Media Service, Payment Service, Notification Service, and Reporting (future).

Integration Map:

- /api/v1/auth/* → Authentication & Authorization
- /api/v1/courses/* → Course Management
- /api/v1/mentor/* → Mentor Room
- /api/v1/webinar/* → Webinar Room
- /api/v1/community/* → Community Features
- /api/v1/projects/* → Project Showcase

3. Authentication & Authorization (enhanced structure)

Authentication & Authorization
The system employs a **JWT-based access control mechanism** with role-based permissions. Each user has a role defining access privileges within the LMS.

Roles

- superadmin: Full access to all modules, users, and settings
- admin: Manage courses, users, and financial records
- instructor: Create courses, host webinars, manage enrolled students
- student: Enroll in courses, submit assignments, attend webinars
- guest: Limited access to public content only

Authentication Flow

1. **Registration:** via email/password or social OAuth (Google, Facebook)
2. **Login:** generates accessToken (short-lived) and refreshToken (long-lived)
3. **Protected Routes:** require header Authorization: Bearer <accessToken>
4. **Refresh Token:** /api/v1/auth/refresh regenerates a valid token pair
5. **Logout:** allows single-session or all-device logout

Security Recommendations

- Enforce HTTPS for all routes
- Use salted bcrypt for password hashing
- Rotate JWT secrets periodically
- Implement request rate limiting on all /auth/* endpoints
- Use strong validation (Joi or Zod) for all user input

3. Authentication & Authorization

Roles:

- superadmin — full access to all modules and settings
- admin — manage users, courses, and financial operations
- instructor — create/manage courses, host webinars, mentor students
- student — enroll in courses, submit assignments, attend webinars
- guest — limited access to public content only

Authentication Flow:

1. **Register:** via email/password or OAuth providers (Google, Facebook, etc.)
2. **Login:** returns accessToken (short-lived) and refreshToken (long-lived)
3. **Protected Routes:** require header Authorization: Bearer <accessToken>
4. **Refresh:** endpoint to exchange refreshToken for a new accessToken
5. **Logout:** option to revoke tokens for current session or all devices

Security Recommendations:

- Enforce **HTTPS** for all API endpoints
- Use **bcrypt** with salt for password hashing
- Rotate **JWT secrets** periodically
- Implement **rate limiting** on authentication endpoints
- Apply **strong password policies** and input validation

4. Users (Students & Instructors)

All users are stored in a **single user's collection**, with additional fields for instructors. This allows a unified schema while supporting role-based access and features.

Field	Type	Description
name	String	Full name of the user
email	String	Unique and indexed for login and notifications
password	String	Hashed with bcrypt; omitted in API responses
role	String	`'student'
bio	String	Short user biography
avatarUrl	String	Link to profile picture
socialLinks	Object	{ Facebook, LinkedIn, website }

Field	Type	Description
skills	Array[String]	User skills or expertise
location	Object	{city, country}
isVerified	Boolean	Indicates if email is verified
isInstructorVerified	Boolean	Indicates if instructor is verified by admin
docs	Array[assetId]	Uploaded verification documents
createdAt	Date	Record creation timestamp
updatedAt	Date	Record last update timestamp

Important Endpoints:

- **GET /api/v1/users** — List users (with filters for role, status, etc.)
- **GET /api/v1/users/:id** — Fetch user profile
- **PUT /api/v1/users/:id** — Update profile or role (role changes restricted to admins)
- **POST /api/v1/users/:id/verify-instructor** — Admin endpoint to verify instructor status

5. Courses, Modules & Lessons

Courses are the primary learning units in Shekhai LMS. Each course can contain multiple modules/lessons, quizzes, assignments, and resources, allowing structured content delivery and progress tracking.

5.1 Course Data Model

Field	Type	Description
title	String	Required course title
slug	String	Unique, SEO-friendly identifier
shortDescription	String	Brief summary of the course
longDescription	String	Detailed description (Markdown or sanitized HTML)
category	ObjectId	Reference to course category
tags	Array[String]	Tags for search/filtering
level	String	`'Beginner'
language	String	Course language

Field	Type	Description
price	Object	{ amount:Number, currency:String, isFree:Boolean }
thumbnailUrl	String	Course thumbnail image URL
coverUrl	String	Cover image URL
instructor	ObjectId	Reference to the user (instructor)
modules	Array [ObjectId]	References to lesson/module documents
durationMinutes	Number	Approximate total duration
published	Boolean	Indicates if course is live
stats	Object	{ studentsCount, rating, reviewsCount }
createdAt	Date	Record creation timestamp
updatedAt	Date	Record last update timestamp

5.2 Module/Lesson Data Model

- courseId: ObjectId — parent course
- title: String
- type: String — 'video'|'article'|'quiz'|'assignment'
- content: String — markdown or structured JSON for quizzes
- media: Array[Object] — {url,type,meta}
- duration: Number — minutes
- order: Number — sort order
- isLocked: Boolean — locked if not enrolled
- createdAt: Date

6. Categories & Tags

Categories and tags help users discover courses. Category supports hierarchy (parent-child).
Category fields: {name, slug, description, parentCategory, order, isPublished}

7. Enrollments & Progress

Enrollments store the relationship between a student and a course, including progress tracking.
Note: Certificate module has been removed per request.

- Enrollment model fields:

- userId: ObjectId — student
- courseId: ObjectId
- status: String — 'active'|'completed'|'cancelled'
- purchasedAt: Date
- progress: Object — {completedModules: [ObjectId], percent:Number, lastAccessed:Date}
- gradeSummary: Object — aggregate quiz/assignment scores
- createdAt: Date

Endpoints include creating enrollments, updating progress, and listing user's enrollments.

8. Quizzes, Assignments & Grading

Supports MCQ quizzes, timed quizzes, assignments with file upload, instructor grading and feedback.

8.1 Quiz Model

- courseId: ObjectId
- moduleId: ObjectId (optional)
- title: String
- questions: Array[{question, type, choices, correctAnswer, points}]
- timeLimitMinutes: Number (optional)
- passingScore: Number (percentage)
- isPublished: Boolean

8.2 Assignment Model

- courseId: ObjectId
- moduleId: ObjectId (optional)
- title: String
- description: String
- attachments: Array[assetId]
- dueDate: Date
- maxPoints: Number
- submissions: Array[{userId, files, submittedAt, grade, feedback}]

9. Project Showcase Module

Purpose:

The Project Showcase module allows students and instructors to publish completed projects or portfolios. It serves as a gallery of student work, supporting images, videos, links to repositories/live demos, tags, and mentor endorsements.

Key responsibilities:

- Allow authenticated users to submit projects with rich media and descriptions.
- Enable instructors/mentors to feature, endorse, or review projects.
- Provide public-facing project browsing, search, and filters.
- Track likes, comments, and view counts.

9.1 Data Model (projects)

- title: String — project title
- slug: String — unique for public URL
- description: String — detailed project description (markdown/html)
- author: ObjectId (users) — student or team lead
- contributors: Array[Object] — [{userId, role}]
- mentorIds: Array[ObjectId] — mentors associated
- techStack: Array[String]
- images: Array[String] — asset URLs or assetIds
- videos: Array[String] — asset URLs or assetIds
- repoUrl: String — GitHub/GitLab URL
- liveDemoUrl: String — deployed demo
- tags: Array[String]
- likesCount: Number
- commentsCount: Number
- status: String — 'draft'|'published'|'archived'
- isFeatured: Boolean
- createdAt: Date
- updatedAt: Date

9.2 Endpoints

1. GET /api/v1/projects — List projects with filters (q, tags, techStack, mentor, author, page, limit)
2. GET /api/v1/projects/:slug — Public project detail
3. POST /api/v1/projects — Create project (authenticated users)
4. PUT /api/v1/projects/:id — Edit project (owner or admin)
5. DELETE /api/v1/projects/:id — Delete project (owner or admin)
6. POST /api/v1/projects/:id/like — Toggle like for authenticated user
7. POST /api/v1/projects/:id/comment — Add comment (authenticated)
8. POST /api/v1/projects/:id/feature — Admin: mark project as featured

9.3 Admin & Mentor Actions

- Admins can feature, archive, or remove projects.
- Mentors can endorse projects (endorsement record stored) and add review notes.
- Project moderation: flagging flow for inappropriate content.

9.4 Business Logic & Notes

- Media uploads should use presigned URLs and store metadata in assets collection.
- Projects can be single-user or team-based; contributors capture the team.
- Consider pagination and caching for high-traffic project listings.

10. Mentor Room Module

Purpose:

Mentor Room is a private workspace for mentors to manage their students, hold 1:1 sessions, give feedback, and track mentee progress. It centralizes mentor-student interactions outside of course content (mentorship, office hours, reviews).

Key responsibilities:

- Mentor profile and availability management.
- Scheduling 1:1 mentorship sessions (bookings) with calendar integration.
- Private chat or notes between mentor and mentee.
- Mentor-led resources and small-group rooms.

10.1 Data Model (mentor_rooms & bookings)

- roomId: String — unique identifier
- mentorId: ObjectId — user who owns room
- title: String — e.g., 'John's Mentor Room'
- description: String
- availability: Array[{day, startTime, endTime, timeZone}]
- officeHours: Array[{date, startAt, endAt, capacity}]
- bookings: Array[ObjectId] — references to bookings collection
- members: Array[ObjectId] — mentees added to room
- resources: Array[assetId] — mentor-shared files
- createdAt: Date
- updatedAt: Date

10.2 Booking Model

- mentorId: ObjectId
- studentId: ObjectId
- roomId: String
- scheduledAt: DateTime
- durationMinutes: Number
- status: String — 'pending'|'confirmed'|'cancelled'|'completed'

- notes: String
- createdAt: Date

10.3 Endpoints

9. GET /api/v1/mentor-rooms — List mentor rooms (public listing of mentors offering mentorship)
10. GET /api/v1/mentor-rooms/:id — Get mentor room details
11. POST /api/v1/mentor-rooms — Create/update mentor room (mentor)
12. POST /api/v1/mentor-rooms/:id/book — Create booking (student)
13. GET /api/v1/mentor-rooms/:id/bookings — Mentor: list bookings
14. PATCH /api/v1/bookings/:id — Update booking status (mentor or student)
15. DELETE /api/v1/bookings/:id — Cancel booking

10.4 Admin & Mentor Actions

- Mentors can set availability, confirm/cancel bookings, and add session notes.
- Admins can manage mentors, override bookings, and view mentor activity logs.

10.5 Business Logic & Notes

- Integrate calendar sync (Google Calendar) for booked sessions if desired.
- Use timezone-aware scheduling and send email/in-app reminders.
- Booking conflicts should be prevented via atomic checks (DB transaction or unique slot reservations).

11. Webinar Room Module

Purpose:

Webinar Room supports scheduled live events (one-to-many) such as live classes, workshops, or guest lectures. It integrates with a video provider (Agora/Twilio/Jitsi) and manages registrations, capacity, recordings, and Q&A.

Key responsibilities:

- Schedule webinars with host(s), description, capacity, and pricing.
- Registration flow and attendee management.
- Integration with third-party video provider for live stream.
- Store recording links and playback for registered users.

11.1 Data Model (webinars)

- title: String
- slug: String
- description: String
- hostIds: Array[ObjectId] — mentors or instructors
- startAt: DateTime
- endAt: DateTime
- capacity: Number
- price: Object — {amount, currency, isFree}
- registrationCount: Number
- attendees: Array[Object] — {userId, status, joinedAt}
- joinUrl: String — provider join link (or generated token)
- recordingUrl: String
- status: String — 'scheduled'|'live'|'ended'|'cancelled'
- createdAt: Date

11.2 Endpoints

16. GET /api/v1/webinars — List webinars (upcoming/past)
17. GET /api/v1/webinars/:id — Webinar detail
18. POST /api/v1/webinars — Create webinar (instructor/admin)
19. PUT /api/v1/webinars/:id — Update webinar
20. DELETE /api/v1/webinars/:id — Cancel webinar
21. POST /api/v1/webinars/:id/register — Register attendee
22. POST /api/v1/webinars/:id/join — Generate join token or redirect (protected)
23. GET /api/v1/webinars/:id/recording — Get recording (if available)

11.3 Business Logic & Notes

- Use provider SDKs to create webinar rooms and tokens; do not route video through your servers.
- Enforce capacity limits and waitlists if needed.
- Post-webinar, store recording metadata and notify registrants.

12. Community Module

Purpose:

The Community module is a social layer for learners: forum-style posts, comments, upvotes, tags, and follow/subscribe features. It encourages collaboration, Q&A, and peer support.

Key responsibilities include:

- Creating and moderating posts and threads.
- Comments, replies, and threaded discussions.
- Upvotes/likes, bookmarks, and following authors or topics.
- Moderation tools: pinning, flagging, removing content.

12.1 Data Models (posts, comments, reactions)

- posts: { _id, title, body, authorId, tags, attachments, pinned, views, upvotes, createdAt }
- comments: { _id, postId, parentCommentId?, authorId, body, upvotes, createdAt }
- reactions: { _id, targetType ('post'|'comment'), targetId, userId, type ('upvote'|'bookmark') }

12.2 Endpoints

24. GET /api/v1/community/posts — List posts with filters (tag, author, q, page)
25. GET /api/v1/community/posts/:id — Get post with comments
26. POST /api/v1/community/posts — Create post (authenticated)
27. PUT /api/v1/community/posts/:id — Edit post (owner/mod)
28. DELETE /api/v1/community/posts/:id — Delete post (owner/mod)
29. POST /api/v1/community/posts/:id/comments — Add comment
30. POST /api/v1/community/posts/:id/upvote — Upvote post
31. POST /api/v1/community/comments/:id/upvote — Upvote comment

12.3 Moderation & Business Rules

- Allow anonymous browsing but require authentication for posting/commenting.
- Implement rate-limits and spam detection (CAPTCHA on post creation if suspicious).
- Provide moderator roles and audit logging for removals/edits.

13. Payments & Orders

Payments handled via Stripe or local gateway. Orders track checkout details and payment status.

Order model and payment flows are similar to previous version (see prior doc).

14. Media & Assets

Use S3-compatible storage and store metadata in assets collection. Use presigned URL flow.

15. Notifications & Real-time

Notifications stored in collection and pushed via Socket.IO. Real-time features include chat in mentor rooms and live webinar events.

16. Admin Panel & Operations

Admin panel should include management for new modules: Project Showcase, Mentor Rooms, Webinars, Community, alongside Users and Courses.

Audit logs, content moderation, and bulk operations should be available.

17. Search, Filtering & Indexing

Indexes recommended include users.email (unique), courses.slug (unique), projects.slug (unique), text indexes on course and project descriptions, and compound indexes for enrollments.

18. Backups, Migrations & Seeding

Nightly backups, migrate-mongo for migrations, and seed scripts for demo data that now include project, mentor-room, webinar, and community seeds.

19. Security & Compliance

Same recommendations: HTTPS, rate-limiting, XSS sanitization, GDPR endpoints for export/delete, secure webhook verification.

20. Deployment & Environment Variables

- MONGO_URI
- PORT
- JWT_SECRET
- JWT_REFRESH_SECRET
- AWS_S3_BUCKET
- AWS_REGION
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- STRIPE_KEY
- STRIPE_WEBHOOK_SECRET
- SENDGRID_API_KEY
- REDIS_URL

21. API Reference — Selected Endpoints

Examples updated to include Project Showcase, Mentor Room, Webinar, and Community endpoints.

Create Project example:

32. POST /api/v1/projects

```
{
 "title":"Smart Garden",
 "slug":"smart-garden",
 "description":"IoT based smart garden project",
 "author":"<userId>",
 "techStack":["Arduino","Node.js"],
 "repoUrl":"https://github.com/...",
 "liveDemoUrl":"https://demo.example.com"
}
```

Create Mentor Room example:

33. POST /api/v1/mentor-rooms

```
{
 "title":"John Doe Mentor Room",
 "mentorId":"<userId>",
 "description":"Office hours and mentorship sessions",

"availability":[{"day":"Monday","startTime":"10:00","endTime":"12:00","timeZone":"Asia/Dhaka"}]
}
```

Create Webinar example:

34. POST /api/v1/webinars

```
{
 "title":"Live React Workshop",
 "slug":"live-react-workshop",
 "description":"Hands-on workshop",
 "hostIds":["<instructorId>"],
 "startAt":"2025-12-10T10:00:00Z",
 "endAt":"2025-12-10T12:00:00Z",
 "capacity":200,
 "price":{"amount":0,"currency":"USD","isFree":true}
}
```

22. Example Mongoose Schemas

Project Schema (simplified):

```
35. const ProjectSchema = new mongoose.Schema({
      title: { type: String, required: true },
      slug: { type: String, unique: true },
      description: String,
      author: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
      contributors: [{ userId: mongoose.Schema.Types.ObjectId, role: String }],
      techStack: [String],
      images: [String],
      videos: [String],
      repoUrl: String,
      liveDemoUrl: String,
      tags: [String],
      status: { type: String, enum: ['draft','published','archived'], default: 'draft' }
    }, { timestamps: true });
```

MentorRoom Schema (simplified):

```
36. const MentorRoomSchema = new mongoose.Schema({
      roomId: { type: String, unique: true },
      mentorId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
      title: String,
      description: String,
      availability: [{ day: String, startTime: String, endTime: String, timeZone: String }],
      members: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }]
    }, { timestamps: true });
```

Webinar Schema (simplified):

```
37. const WebinarSchema = new mongoose.Schema({
      title: String,
      slug: { type: String, unique: true },
      description: String,
      hostIds: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
      startAt: Date,
      endAt: Date,
      capacity: Number,
      price: { amount: Number, currency: String, isFree: Boolean },
      attendees: [{ userId: mongoose.Schema.Types.ObjectId, status: String, joinedAt: Date }],
      recordingUrl: String,
      status: String
```

```
}, { timestamps: true });
```