Software Requirements Specification

for

MelodyHub – Music Education Platform

Version 1.0 approved

Prepared by Group 31:
Amrik Bhadra(202201040021)
Essak Dasari (202201040013)
Raviraj Tekle (202302040010)
Harish Chavankde (0120200177)
Omkar Shinde (202201040190)
Ankur Dome (202201040191)

30-05-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

● Introduction

●.1 Purpose

This document outlines the Software Requirements Specification (SRS) for **MelodyHub v1.0**, a full-featured music education platform. The goal of this platform is to provide structured, interactive, and personalized music learning experiences. This SRS covers the entire software system, including backend services, frontend interfaces, and integrations with third-party APIs (e.g., video streaming, payment gateways). It is intended for use throughout the project lifecycle, from planning and design to development and testing.

●.2 Document Conventions

- **Bold text** is used to highlight section titles, sub-headings & important keywords/information.
- Each functional requirement is uniquely labeled using the format **REQ-#** (e.g., REQ-1, REQ-2).
- **Each requirement is assigned an explicit priority level** (High, Medium, Low).
- Technical terms and abbreviations are defined in Appendix A: Glossary

●.3 Intended Audience and Reading Suggestions

This document is intended for the following stakeholders:

- **Developers** – to understand functional and non-functional system requirements.
- **Testers/QA Engineers** – to develop test cases and test plans based on defined requirements.
- **Project Managers and Product Owners** – to monitor progress and ensure deliverables align with objectives.
- **UX/UI Designers** – to align designs with functional flows and use cases.
- **Client and Investors** – to get a clear overview of product capabilities and expected outcomes.

●.4 Product Scope

MelodyHub is a web-based music education platform aimed at democratizing access to high-quality music learning. The platform delivers structured courses in music theory and practice, supports live sessions, tracks progress, and integrates AI-based feedback. It enables global collaboration between students and instructors and supports monetization via subscription and course-based payments. MelodyHub aligns with modern educational trends by offering an engaging, scalable, and personalized learning environment.

Key goals include:

- Enhancing music learning through interactive, engaging content.
- Supporting instructors with tools to create and manage courses.
- Empowering learners with tailored learning paths and progress tracking.
- Supporting platform monetization through subscriptions and certification*s.*

●.5 References

- IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- JWT Authentication Guide – https://jwt.io/introduction
- React.js Official Docs – https://reactjs.org/docs/getting-started.html
- Node.js Official Docs – https://nodejs.org/en/docs
- MongoDB Manual – https://www.mongodb.com/docs/manual/

● Overall Description

●.1 Product Perspective

MelodyHub is a new, standalone web-based platform designed to deliver comprehensive music education through interactive, personalized learning tools. It is not an extension or replacement of any previous system and is being developed from scratch using the **MERN stack** (MongoDB, Express.js, React.js, Node.js). While the product itself is self-contained, it integrates with several third-party services such as **RazorPay** for payments, **WebRTC** for live sessions, and AI feedback engines for music analysis.

●.2 Product Functions

At a high level, MelodyHub will enable the following functionalities:

- **User Authentication & Management**: Secure signup/login, role-based access (student, instructor, admin), and profile settings.
- **Course Management**: Creation, browsing, enrollment, and progress tracking of music courses.
- **Content Delivery**: Video-based lessons, quizzes, assignments, and interactive tools.
- **Live Sessions**: Schedule and attend real-time video classes and webinars.
- **Feedback & Analytics**: AI-driven performance analysis and progress reports.
- **Communication**: In-app chat, discussion forums, and community engagement.
- **Monetization**: Support for subscriptions, course-based payments, and certifications

●.3 User Classes and Characteristics

MelodyHub will cater to multiple user types:

- **Students**
  - Primary users of the platform
  - May vary in age and musical experience
  - Use basic to intermediate features like course consumption, quizzes, and progress tracking

- **Instructors**
  - Skilled musicians or educators
  - Require advanced tools for course creation, feedback, and live session management

- **Administrators**
  - Responsible for content moderation, user management, and platform oversight
  - Require full backend access and analytics

- **Visitors/Guests**

  - Can browse available courses but need to register to access learning conten*t*

●.4 Operating Environment

MelodyHub will be accessible on **standard web browsers** (Chrome, Firefox, Safari, Edge) and optimized **for desktop and mobile**.

- **Frontend**: React.js (HTML5, CSS3, JavaScript)
- **Backend**: Node.js with Express.js
- **Database**: MongoDB (Cloud-hosted)
- **Hosting**: Deployed via platforms like Vercel (frontend) and AWS/GCP/Render (backend)
- **Third-party Integrations**: RazorPay, WebRTC, Firebase for email services, and optional AI/ML APIs

●.5 Design Principles for Scalability & Performance

Microservice-Oriented Architecture (Future-Ready)

- Decompose major components such as User Management, Course Delivery, Payments, Notifications, and AI Feedback into independent services (Node.js with Express or NestJS).

- Use containerization (Docker) and orchestration tools (Kubernetes or AWS ECS).

Asynchronous and Non-blocking Communication

- Use message queues (RabbitMQ, Kafka) for background tasks like email dispatch, progress tracking, and AI feedback generation.

Database Optimization

- Use indexing on frequently queried fields (e.g., `user_id`, `course_id`, `lesson_progress`).
- Separate read/write loads with read replicas.
- Use PostgreSQL for relational modules (courses, enrollments) and MongoDB for document-oriented data (media, metadata).

Caching for High-Read APIs

- Use Redis for caching frequently accessed data: course catalogs, user profiles, certificates, and video metadata.
- Use CDN (e.g., CloudFront, Cloudflare) to serve static assets (videos, thumbnails, PDFs).

Load Balancing

- Use Application Load Balancer (ALB) or Nginx to distribute traffic evenly.
- Enable auto-scaling groups based on CPU, memory, or queue depth.

Frontend Performance Optimization

- Implement code splitting with lazy loading.
- Compress video and static assets (Gzip, Brotli).
- Use service workers for offline support and caching.

API Rate Limiting and Throttling

- Prevent abuse using API gateway solutions (e.g., Kong) or Express middleware like `express-rate-limit`.

CDN Usage

- Offload static content such as images, videos, and documents to a CDN for edge caching and performance improvement.

●.6 Security Principles

Access Control & RBAC

- Enforce Role-Based Access Control at both frontend routes and backend APIs.
- Implement middleware for role checks at the controller level.

Secure Authentication

- Use JWTs for access and refresh tokens.
- Implement token revocation and rotation.
- Require Multi-Factor Authentication (MFA) for admin and instructor accounts.

Secure Data Transmission

- Enforce HTTPS for all APIs using TLS 1.2 or higher.
- Use HTTP security headers (HSTS, Content-Security-Policy).

Dependency Security

- Use tools like `npm audit`, Snyk, or GitHub Dependabot for vulnerability scanning.
- Lock dependency versions using `package-lock.json`.
- Regularly patch and review dependencies.

Input Validation & Sanitization

- Use validation libraries like Joi/Zod (backend) and Yup/Formik (frontend).
- Prevent common attacks such as XSS, SQL injection, and CSRF.

Secure File Uploads

- Restrict uploads to specific MIME types.
- Scan files using antivirus tools (e.g., ClamAV).
- Use signed URLs for direct-to-S3 uploads to bypass backend load.

Secure Payments (PCI-DSS Compliance)

- Use hosted payment pages from Stripe or Razorpay.
- Do not handle card data directly.
- Enable logging of transactions and secure configurations using MFA and audit logs.

Decision Logging (RAID Logs)

- Log key business logic events (e.g., "Certificate issued", "Role changed").
- Include `user_id`, timestamp, affected resource, and reason.
- Store logs securely with rotation using Winston or ELK/EFK stack.

Access Monitoring and Audit

- Log login attempts, token creation, and course actions.
- Centralize logs using CloudWatch, Graylog, or ELK.
- Set alerts for suspicious activity like brute-force login attempts.

Defense in Depth

- Restrict database user privileges.
- Use a separate VPC for the database, accessible only by backend services.
- Enforce IAM roles and isolate environments (dev/staging/prod).

●.7 Maintainability, Portability, & Reliability

Component Modularity

- Maintain clean folder structures based on Domain-Driven Design.
- Separate concerns: core logic, services, DB models, and controllers.

CI/CD Pipeline

- Automate build, lint, test, and deploy using GitHub Actions, GitLab CI, or Jenkins.
- Use ESLint/Prettier for code quality and run Jest tests before deployment.

Unit & Integration Testing

- Ensure unit test coverage exceeds 80%.
- Use Postman/Newman or Supertest for API integration testing.

Monitoring and Alerting

- Integrate Prometheus with Grafana for system and custom metrics.

- Use uptime monitoring tools like Pingdom or UptimeRobot.
- Notify engineering via Slack or Discord on error trends.

Environment Configuration

- Manage secrets and configs using environment variables.
- Securely store in AWS Systems Manager (SSM) or Vault.

●.8 Design and Implementation Constraints

- Must use the **MERN stack** as per team expertise
- **JWT-based** authentication is required for secure role-based access control
- **Real-time features** (chat, live sessions) must scale with user volume
- **AI-based feedback systems** may depend on available open-source models or APIs
- Integration with **Razorpay** is mandatory for handling payments

●.9 User Documentation

The following documentation will be provided with the platform:

- **User Manual** – for students and instructors (PDF + HTML format)
- **Admin Guide** – covering backend controls and reports
- **Onboarding Tutorials** – guided flows for new users inside the app
- **Help Center/FAQ Page** – accessible from the app interface
- **API Documentation** – for internal/external use (Swagger/OpenAPI format)

●.10 Assumptions and Dependencies

- Assumes continuous internet connectivity for all real-time and content-heavy features
- Dependent on third-party APIs (Stripe, Zoom/WebRTC) for core functionalities
- Assumes instructors will be responsible for uploading content and managing their courses
- Depends on React and Node.js libraries being compatible across all modules
- Scalable cloud hosting will be available for deployment and data storage

● External Interface Requirements

●.1 User Interfaces

MelodyHub will follow a clean, responsive, and intuitive design with a focus on user experience across various roles—students, instructors, and administrators.

- **General Layout**: The platform will adopt a consistent top navigation bar for global access to dashboard, courses, profile, notifications, and help.
- **Theme and Style**: Uses a modern UI design system (such as Material UI or Tailwind CSS standards). Consistent use of color palette and iconography is enforced.
- **Screen Layout**: Layouts will be optimized for mobile, tablet, and desktop views. Major screens include Home, Course Explorer, Course Detail, Interactive Lessons, Instructor Dashboard, and Admin Panel.
- **Standard Components**:
    - Search bars on top of course listings
    - “Play Lesson” buttons for interactive content
    - “Mark Complete” checkboxes for progress tracking
    - “Upload Content” buttons for instructors
    - Toast or modal-based error/success notifications
- **Error Messages**: All validation and error messages will be displayed contextually with user-friendly messages (e.g., "Please enter a valid email address").

●.2 Hardware Interfaces

MelodyHub is a web-based application accessible through a browser; however, it may also include a mobile version in future phases.

- **Client Devices**: Compatible with standard personal computers, laptops, tablets, and smartphones.
- **Input Devices**: Supports keyboard, mouse, touch, and microphone (for voice-based interaction features).
- **Audio Devices**: Headphones/speakers required for listening to audio lessons; microphone recommended for singing practice modules.
- **Server Hardware**: Hosted on a cloud server (e.g., AWS EC2) with scalable CPU/RAM resources depending on load.

●.3 Software Interfaces

Operating Systems:

- Backend: AWS Linux
- Client: Any OS with a modern browser (Windows, macOS, Linux, Android, iOS)

Databases:

- MongoDb for non-relational data (users, courses, enrollments)

Libraries and Frameworks:

- Frontend: React.js (v18+), Tailwind CSS
- Backend: Node.js (v18+), Express.js
- JWT for auth, Multer for uploads, WebRTC for Realtime video conference

APIs and Protocols:

- RESTful API (v1) over HTTPS
- Cloud storage via AWS S3 for storing lesson content
- Payments integration via Razorpay

Shared Data:

- User profile info, enrolled courses, lesson progress, and certificate data are shared between frontend and backend via REST APIs.

●.4 Communications Interfaces

Protocols:

- HTTP/HTTPS for all client-server communication
- WebSocket support (TBD) for live class modules or real-time chat

Security:

- TLS encryption for all API endpoints
- OAuth2.0 support for external logins (Google/GitHub)

Data Transfer:

- JSON format for request/response payloads
- Average payload < 500KB for lesson content metadata

Browser Compatibility:

- Latest two versions of Chrome, Firefox, Edge, and Safari

4. System Features

4.1 Course Enrollment and Learning Progress

4.1.1 Description and Priority

This feature allows users to browse available music courses, enroll, and track their learning progress.

4.1.2 Stimulus/Response Sequences

1. User clicks on a course → System displays course details.
2. User clicks "Enroll" → System registers enrollment and updates the dashboard.
3. User completes lesson → System marks it “Completed” and updates progress bar.

4.1.3 Functional Requirements

- **REQ-COURSE-1**: The system shall display all available courses on the Course Explorer page.
- **REQ-COURSE-2**: The system shall allow logged-in users to enroll in any course after successful payment.
- **REQ-COURSE-4**: The system shall update the user’s course dashboard on enrollment.
- **REQ-COURSE-5**: The system shall track and store progress for each enrolled course.
- **REQ-COURSE-6**: The system shall restrict lesson access for non-enrolled users.

4.2 Certification Generation and Download

4.2.1 Description and Priority

This feature generates downloadable certificates and badges for users upon successful course completion.

4.2.2 Stimulus/Response Sequences

1. User completes final lesson → System verifies all lessons completed.
2. System triggers certificate generation → User notified via dashboard and email.

4.2.3 Functional Requirements

- **REQ-CRT-1**: The system shall verify course completion before issuing a certificate.
- **REQ-CRT-2**: The system shall generate a PDF certificate with the user's name and course title.
- **REQ-CRT-3**: The system shall allow users to download and share certificates.

4.3 Collaboration & Communication

4.3.1 Description and Priority

This feature supports direct interaction between users—specifically between students, instructors, and the broader learning community. It includes real-time chat, asynchronous discussions via forums, and live video communication using integrated platforms like Zoom or WebRTC. This feature is of High priority as it enhances learner engagement, peer support, and instructor guidance, which are critical in an e-learning environment.

4.3.2 Stimulus/Response Sequences

Scenario 1 – Live Chat
Stimulus: A student opens the course page and initiates a chat with the instructor.
Response: The system checks if the instructor is online. If yes, it opens a live chat session; if not, a message is queued. The instructor responds when available.

Scenario 2 – Community Forums
Stimulus: A student navigates to the community forum of a course and posts a question.
Response: The system adds the post to the forum and notifies other enrolled students and the instructor.

Scenario 3 – Virtual Rooms / Video Calls
Stimulus: An instructor schedules a virtual session using Zoom/WebRTC.
Response: The system creates a meeting link and sends notifications to enrolled students. Students can join the session via browser.

4.3.3 Functional Requirements

- **REQ-COM-1**: The system shall provide a real-time text chat interface for students and instructors within a course context.
- **REQ-COM-2:** The chat system shall support presence detection (online/offline/away) for instructors.
- **REQ-COM-3**: The system shall store chat history and allow users to review previous conversations.
- **REQ-COM-4**: The system shall provide a community forum for each course with support for threads, replies, and likes/upvotes.
- **REQ-COM-5:** The forum shall support markdown or rich text formatting for better readability.
- **REQ-COM-6**: The system shall integrate with Zoom API or WebRTC to allow instructors to schedule and host live video sessions.
- **REQ-COM-7**: The system shall generate calendar events and email notifications for scheduled virtual sessions.
- **REQ-COM-8:** Only enrolled students shall be able to participate in the chat, forum, or video sessions for a given course.

4.4 Certification Generation and Download

4.4.1 Description and Priority

This feature enables **authorized users (admins or instructors)** to create new courses on the MelodyHub platform. Each course can include structured content such as lessons, video material, descriptions, quizzes, and supplementary resources.

Priority: High — This feature is critical for maintaining and growing the content repository of the platform.

4.2.2 Stimulus/Response Sequences

Sequence 1: Course Creation

- Admin/Instructor logs into the dashboard.
- Navigates to the “Create Course” panel and fills out course metadata (title, description, category, difficulty level).
- System validates the input and creates a new course entry in the database.
- System displays a success message and redirects to “Add Lessons” screen.

Sequence 2: Adding Lessons to a Course

- User selects the newly created course from the dashboard.
- User uploads lesson materials (video, text, PDF) and assigns lesson order.
- System stores the files and metadata, then links lessons to the course.
- User saves the course structure → System confirms and publishes the course if all required fields are completed.

4.4.3 Functional Requirements

- **REQ-COURSE-1:** The system shall allow authenticated instructors or admins to create a new course with metadata.
- **REQ-COURSE-2:** The system shall validate that all required fields (course title, category, level) are filled before submission.
- **REQ-COURSE-3:** The system shall allow users to upload multiple lessons per course including video, text, and PDFs.
- **REQ-COURSE-4:** The system shall support lesson ordering within a course.
- **REQ-COURSE-5:** The system shall confirm successful course creation and return a unique course ID.
- **REQ-COURSE-6:** The system shall allow instructors to preview and edit the course before publishing.
- **REQ-COURSE-7:** The system shall ensure only authorized roles can create or edit courses

5. Nonfunctional Requirements

5.1 Performance Requirements

- Course listing pages must load in under 2 seconds.
- Lesson playback should buffer within 1.5 seconds on a 5 Mbps connection.
- System should handle 100 concurrent users during peak time without service degradation.
- Database operations (enrollments, updates) must complete within 300ms.

5.2 Security Requirements

- **Regular Data Backups**: All user data (progress, content, uploads) must be backed up every 24 hours to ensure data integrity and disaster recovery readiness.
- **Failover and Redundancy**: The system must include failover mechanisms such as redundant storage and mirrored databases to maintain high availability and reliability.
- **Malicious Upload Protection**: All file uploads must be validated using MIME type checking and scanned using antivirus software to block malicious content.
- **Authentication with JWT**: All API and system endpoints must require authentication using JSON Web Tokens (JWT), utilizing both access and refresh tokens.
- **Role-Based Access Control (RBAC)**: User roles such as student, instructor, and admin must be clearly separated with access permissions enforced accordingly.
- **Password Security**: All user passwords must be securely hashed using the bcrypt algorithm before being stored in the database.
- **Data Encryption in Transit**: All data transmitted between clients and the server must be encrypted using TLS version 1.2 or higher to prevent eavesdropping and tampering.

- **PCI DSS Compliance -** To ensure secure handling of payment information in accordance with the Payment Card Industry Data Security Standard (PCI DSS), the following security measures must be implemented in the payment system module:

  Secure Payment Gateway

  - Use PCI DSS-compliant gateways (e.g., Stripe, Razorpay, PayPal).
  - Redirect users to hosted payment pages (no direct card data handling).

  Encryption & Secure Transmission

  - Use TLS 1.2+ for all payment-related data.
  - Enforce HTTPS on auth and payment pages.

  Cardholder Data Protection

- Do **not** store CVV/CVC or full card data.
- Encrypt any stored payment tokens (e.g., AES-256).

Access Control & Authentication

- Restrict access to payment configs/logs.
- Enforce MFA and RBAC for admin access.

Monitoring & Logging

- Log all payment transactions and access attempts.
- Protect logs from unauthorized access/tampering.

Vulnerability Management

- Regularly update payment-related libraries and APIs.

5.3 Software Quality Attributes

- **Availability**: 99.9% uptime expected
- **Usability**: User onboarding process should not exceed 3 steps
- **Maintainability**: Codebase follows ESLint and Prettier for readability
- **Portability**: Compatible across devices and browsers
- **Testability**: 80% unit test coverage with automated CI pipeline

5.4 Business Rules

- Only instructors can create or modify courses.
- Students can enroll in both free and paid courses, but cannot access paid content without payment.
- Certificates are auto-issued only if all lessons are marked as completed and quizzes (if any) are passed.
- Instructors must be verified by the admin before their courses go live.

6. Other Requirements

- **Database Requirements**:
  - Normalize to 3NF, support ACID properties.
  - Store video URLs, not actual video files (videos stored in cloud).
- **Internationalization**:

    - Initial support for English, future support for Hindi.
- **Legal**:
    - GDPR-compliance for EU users.
    - COPPA-compliance if minors use the platform.
- **Reuse Goals**:
    - Common components (e.g., video player, quiz engine) to be designed as reusable React components.

Appendix A: Glossary

UI: User Interface

JWT: JSON Web Token

SRS: Software Requirements Specification

API: Application Programming Interface

CI/CD: Continuous Integration / Continuous Deployment

Appendix B: Analysis Models

DFD Level 1: Course Enrollment Flow

ER Diagram: User ↔ Enrollment ↔ Course ↔ Lesson

Use Case Diagram: Instructor, Student, Admin

Appendix C: To Be Determined List

1. **TBD-01**: Live video class module architecture and timeline
2. **TBD-02**: Integration of gamification mechanics like leaderboard