Software Requirements Specification (SRS) & API Documentation

Knowledge Delivery Platform

Software Requirements Specification (SRS) & API Documentation 1
Knowledge Delivery Platform 1
1. Problem Statement 4
2. Scope 2
2.1 Business Objectives 3
2.2 In Scope (MVP) 3
2.2.1 Event Lifecycle (CRUD) 3
2.2.2 Data Model & Validation (Baseline) 4
2.2.3 Error Semantics 4
2.2.4 API Behavior & Contract 4
2.2.5 Operability & Deployment 4
2.2.6 Documentation (Developer-Facing) 4
2.3 Out of Scope (Deferred / Future Enhancements) 5
2.4 Stakeholders 5
2.5 Assumptions & Constraints 6
2.6 Success Criteria (MVP Acceptance) 6
2.7 External Interfaces (High-Level) 7
2.8 Out-of-Scope Risks & Mitigations (Planning) 7
3. Functional Requirements 11
4. System Description 11
4.1 Backend (Core Services) 11
4.2 Database Layer (MongoDB) 12

4.3 Authentication & Authorization 12
4.4 Frontend Clients 13
4.5 Real-Time & Extended Features 13
4.6 Deployment & Infrastructure 13
4.7 Workflow Example 14
5. Detailed Requirements 14
5.1 Add Course 14
5.2 Get All Courses 14
5.3 Enroll Student 14
5.4 Update Progress 15
5.5 Get Quiz 15
5.6 Submit Quiz 15
6. Non-Functional Requirements 15
6.1 Performance 15
6.2 Scalability 16
6.3 Security 16
6.4 Reliability 16
6.5 Usability 17
6.6 Maintainability 17
7. Roles and Responsibilities 17
8. RBAC (Role-Based Access Control Matrix) 18
9. Entity Relationship (ER) Design 18
10. API Documentation 19
Add Course 19
Get All Courses 19
Enroll Student 19
Update Progress 19
Get Quiz 19
Submit Quiz 20

11. Appendix 20
Abbreviations 20
12. Existing Solutions 20
Moodle 20
Blackboard 20
Google Classroom 20
13. References 21

1. Problem Statement

Education and professional training are rapidly shifting from traditional classrooms to digital platforms. While online learning offers **flexibility, accessibility, and scalability**, it also presents significant challenges for institutions, instructors, and learners when there is no structured system in place.

Without a centralized Knowledge Delivery Platform:

- **Course Delivery Issues:** Instructors struggle to distribute materials, assignments, and quizzes consistently. Learners often miss resources or face confusion regarding course structure.
- **Student Enrollment & Tracking Gaps:** Institutions cannot effectively manage which students are enrolled in which courses, leading to administrative errors and poor monitoring of participation.
- **Progress Monitoring Challenges:** Instructors find it difficult to track student progress, measure performance, and provide timely interventions. Learners also lack clarity about their completion percentage and learning outcomes.
- **Assessment Inefficiency:** Conducting quizzes, evaluating results, and maintaining student performance records manually is time-consuming and error-prone.
- **Scalability Problems:** Traditional methods or fragmented tools do not scale efficiently when thousands of students need simultaneous access.
- **Security Concerns:** In the absence of role-based access, unauthorized users may modify course content or access sensitive student performance data.

The **Knowledge Delivery Platform** is designed to overcome these limitations by:

- Providing a **centralized digital platform** for course creation, enrollment, content delivery, and assessments.
- Enabling **real-time progress tracking** and performance analysis for students.
- Supporting **secure role-based access** to ensure only authorized users (Admin, Instructor, Student) perform specific operations.
- Offering **quiz and evaluation modules** for structured assessments.
- Ensuring **scalability and availability** so institutions can handle thousands of learners concurrently.

In summary, an LMS addresses the pressing challenges of **course management, enrollment, progress tracking, and assessment** by providing a unified, secure, and

scalable platform that enhances learning outcomes for students and reduces administrative overhead for institutions.

2. Scope

This section defines the functional boundaries, stakeholders, assumptions, and success criteria for the **Knowledge Delivery Platform** . It clarifies what the MVP will deliver using the implemented controllers and what is deferred for future releases.

2.1 Business Objectives

- **Centralized Learning Hub**: Provide a single, reliable source of truth for courses, progress, and assessments.
- **Efficiency**: Reduce administrative overhead for instructors by standardizing course creation, enrollment, and quiz evaluation.
- **Integration-Ready**: Expose clean REST APIs for use by mobile/web learning clients.
- **Foundation for Growth**: Lay groundwork for advanced features (certificates, grading systems, role-based access, analytics) without major redesign of the core data model.

2.2 In Scope (MVP)

The MVP strictly aligns with the implemented controller endpoints:

- **Course CRUD (partial)**
  - `GET /api/courses`
  - `POST /api/courses`
- **Enrollment & Progress**
  - `PUT /api/courses/:id/enroll`
  - `PUT /api/courses/:id/progress`
- **Quizzes**

- `GET /api/courses/:id/quiz`
- `POST /api/courses/:id/quiz`

2.2.1 Course Lifecycle

Create Course (POST /api/courses)

- Accepts course details (title, description, duration, modules).
- Persists the new course in MongoDB.
- Returns `201 Created` with the stored document.
- Validation: Required fields must be present; duration must be positive integer.

Read Courses (GET /api/courses)

- Returns all courses as JSON array (`200 OK`).
- If no courses exist, returns empty array.

2.2.2 Enrollment

Enroll Student (PUT /api/courses/:id/enroll)

- Accepts course ID (`:id`) and student identifier (`student` query param).
- Adds student to course's enrollment list.
- Returns updated course object (`200 OK`).
- Returns `404` if course not found.
- Validation: Student identifier required.

2.2.3 Progress Tracking

Update Progress (PUT /api/courses/:id/progress)

- Accepts course ID and student ID (`student` query param).
- Updates progress percentage (`progress` query param, integer 0–100).

- Returns updated course object (`200 OK`).
- Returns `400 Bad Request` for invalid progress values.
- Returns `404` if course/student not found.

2.2.4 Quizzes

Get Quiz (GET /api/courses/:id/quiz)

- Retrieves quiz linked to the given course.
- Returns JSON containing quiz questions.
- Returns `404` if no quiz is available.

Submit Quiz (POST /api/courses/:id/quiz)

- Accepts student ID and score (`student` and `score` query params).
- Records quiz submission result for the student.
- Returns JSON containing success confirmation and updated student record.
- Returns `400` if score is not numeric.
- Returns `404` if course not found.

2.2.5 Data Model & Validation

- **Course Model** (MongoDB via Mongoose):
  - `_id`: ObjectID (auto-generated)
  - `title`: String (required)
  - `description`: String
  - `duration`: Number (hours, required)
  - `students`: Array of student identifiers
  - `progress`: Mapping of student → progress %
  - `quiz`: Object (questions, answers, scores)
- Validation rules:
  - Required fields enforced at model level.
  - Numeric fields (duration, progress, score) validated for proper ranges.
- Error Handling:

- **400** for validation errors or malformed requests.
- **404** if `:id` not found.
- **500** for DB or server errors.

2.2.6 API Behavior & Contract

- Request/Response format: JSON only.
- Time representation: All timestamps in ISO 8601 (UTC).
- Idempotency:
  - `GET` is idempotent and safe.
  - `PUT` is idempotent at the resource level.
  - `POST` is non-idempotent by default.

2.2.7 Operability & Deployment

- Containerized **Node.js + Express** microservice.
- Deployment: AWS EKS (Kubernetes).
- Configurations (ports, DB URI, CORS origins) managed via environment variables.
- Basic logging: request errors, DB failures (stdout + CloudWatch/ELK integration).
- Health check endpoint: `GET /health` returns service status.

2.2.8 Documentation

- Developer-facing API documentation recommended via **Swagger/OpenAPI**.
- Example request/response schemas provided for all endpoints.
- Include success and failure responses.

2.3 Out of Scope (Deferred / Future Enhancements)

- **User Accounts & Authentication**: JWT/OAuth, RBAC (instructor/student).

- **Certificates & Badges**: Award completion certificates or achievements.
- **Search, Filters & Pagination**: Advanced course discovery features.
- **Notifications**: Push/email alerts for deadlines or new content.
- **Payments & Monetization**: Paid courses, subscriptions, refunds.
- **Advanced Quiz Engine**: Question pools, adaptive assessments, grading rubrics.
- **Analytics**: Progress dashboards, performance insights.
- **Internationalization**: Multi-language course content.
- **Multi-Tenancy**: Multiple institutions sharing same platform.

2.4 Stakeholders

- **Instructor/Admin**: Creates and manages courses, tracks student progress.
- **Student**: Enrolls in courses, tracks own progress, completes quizzes.
- **Developer/Maintainer**: Extends APIs, bug fixes, monitors deployments.
- **Community/Institution**: Sponsors courses, governs platform priorities.
- **DevOps Engineer**: Ensures deployment, monitoring, backups, failover.

2.5 Assumptions & Constraints

- Requires internet access (offline mode out of scope).
- Server stores times in UTC; clients render in local timezone.
- MVP assumes thousands of students/courses scale; large-scale sharding deferred.
- PII minimization: Only basic student identifiers, no sensitive data.
- Single-tenant for MVP (one institution/community).

2.6 Success Criteria

Functional:

- All six endpoints respond with correct status codes and payloads.
- Enrollment, progress, and quiz submissions are persistent.

Performance:

- Median latency ≤ 300 ms, p95 ≤ 600 ms under normal loads.

Reliability:

- No data loss in create/enroll/progress/quiz flows.
- Consistent reads after writes.

Operability:

- Service runs in containerized environment.
- `/health` endpoint returns `"OK"`.
- Logs available for debugging.

Documentation:

- OpenAPI/Swagger published with endpoint definitions and sample responses.

2.7 External Interfaces

- **Client Interfaces**: Web or mobile apps consuming JSON over HTTPS.
- **Database**: MongoDB (via Mongoose ODM).
- **Observability**: Logs to stdout/CloudWatch; health check endpoints for K8s readiness/liveness probes.

2.8 Risks & Mitigations

- **No Authentication in MVP**: Limit exposure to trusted environments; add auth in next iteration.
- **No Pagination**: Course lists could grow large; mitigate via frontend caps until backend pagination is added.
- **No Certificate/Badges**: Clearly state learning validation is informational only.
- **Limited Quiz Logic**: Only score storage supported; advanced grading deferred.

3. Functional Requirements

1. **Course Management**
    a. Add, update, delete, and view courses.
2. **Student Enrollment**
    a. Enroll students in a course.
    b. Manage student lists per course.
3. **Progress Tracking**
    a. Update and track student course completion progress.
    b. View detailed progress reports.
4. **Quiz Management**
    a. Retrieve quizzes linked to a course.
    b. Allow students to attempt and submit quizzes.
    c. Store quiz results for evaluation.
5. **Access Control**
    a. Ensure different roles (Admin, Instructor, Student) have defined permissions.

4. System Description

The **Knowledge Delivery Platform** is architected as a modular, scalable, and secure application designed to support digital education, training programs, and online course delivery. Its architecture separates concerns into backend services, persistent storage, authentication, and client applications, ensuring flexibility and maintainability.

4.1 Backend (Core Services)

- **Framework:** Node.js with Express.js provides RESTful APIs for handling all requests related to courses, users, enrollments, assessments, and progress.
- **API Layers:**
    - **Course Management:** Create, update, and fetch course details (titles, descriptions, modules, resources).

- **Enrollment Management:** Register learners for courses, track enrollment status, and manage batch allocations.
- **Assessment & Quizzes:** APIs to create, fetch, and evaluate quizzes. Results are tied to student profiles for performance analysis.
- **Progress Tracking:** Tracks completion rates, time spent, and assessment scores per learner.

- **Error Handling:** Implements HTTP status code standards with meaningful error messages (400 for invalid input, 404 for missing resource, 500 for system failures).
- **Scalability:** APIs are stateless, enabling easy scaling using Kubernetes clusters in cloud environments.

4.2 Database Layer (MongoDB)

- **Data Storage:**
  - **Courses:** Stores metadata (title, category, duration, description, media links).
  - **Users:** Profile details (name, email, roles).
  - **Enrollments:** Links users to courses with enrollment date and status.
  - **Quizzes:** Stores questions, options, correct answers, and submissions.
  - **Progress Metrics:** Tracks lesson completion, quiz scores, and overall course completion percentage.
- **Advantages of MongoDB:**
  - Schema flexibility for diverse course structures.
  - Horizontal scaling for handling large datasets.
  - Indexing to optimize query performance (e.g., fetching courses by category or user enrollments).

4.3 Authentication & Authorization

- **Authentication:** Implemented using **JWT (JSON Web Tokens)**.
  - Users log in with credentials → Token issued.
  - Token required for all protected endpoints (e.g., course enrollment, quiz submission).

- **Authorization (RBAC – Role-Based Access Control):**
  - **Admin:** Manage courses, users, and system-wide analytics.
  - **Instructor:** Create/manage courses, add quizzes, view student progress.
  - **Student:** Enroll in courses, attempt quizzes, view progress.
  - **Guest:** Limited access to public courses or previews.
- **Security Practices:** Input validation, HTTPS in production, token expiration, and refresh logic.

4.4 Frontend Clients

- **Web Client (React/Angular):**
  - Interactive dashboards for students, instructors, and admins.
  - Features: Course listings, enrollment forms, quiz interfaces, and progress charts.
- **Mobile Clients (React Native/Flutter):**
  - Focused on accessibility and mobile-first learning.
  - Supports offline learning (future scope) with local storage sync.
- **API Consumption:** Both web and mobile clients consume REST APIs for course data, authentication, progress tracking, and assessments.

4.5 Real-Time & Extended Features

- **WebSockets / Socket.IO (Future Scope):** For live classes, collaborative learning, and real-time chat.
- **Notifications:** Push/email notifications for enrollments, due assignments, and results.
- **Analytics Layer:** Aggregates data to provide reports on learner performance, course popularity, and system utilization.

4.6 Deployment & Infrastructure

- **Containerization:** Docker images used for portability.

- **Orchestration:** Kubernetes (AWS EKS/GCP GKE) for scaling and resilience.
- **Database Hosting:** MongoDB Atlas for secure, managed storage.
- **CI/CD Pipeline:** Automated testing and deployment using GitHub Actions/Jenkins.
- **Monitoring & Logging:** Integrated with Prometheus + Grafana for monitoring and ELK stack for centralized logging.

4.7 Workflow Example

1. **Admin/Instructor** creates a course → Stored in MongoDB.
2. **Student** enrolls via `POST /api/enrollments` → Enrollment entry created.
3. **Student** starts a quiz → Submission stored, score calculated → Progress updated.
4. **Student/Instructor** views dashboard → APIs aggregate course, enrollment, and quiz data.
5. **Admin** monitors analytics → API retrieves reports from aggregated data.

5. Detailed Requirements

5.1 Add Course

- **Input:** JSON object with course details (`title`, `description`, `instructor`).
- **Output:** Newly created course record.

5.2 Get All Courses

- **Input:** None.
- **Output:** List of all available courses.

5.3 Enroll Student

- **Input:** Course ID (param), Student ID/Name (query).
- **Output:** Updated course object with enrolled student list.

5.4 Update Progress

- **Input:** Course ID (param), Student ID (query), Progress % (query).
- **Output:** Course record with updated progress for the student.

5.5 Get Quiz

- **Input:** Course ID (param).
- **Output:** Quiz details for the course.

5.6 Submit Quiz

- **Input:** Course ID (param), Student ID (query), Quiz Score (query).
- **Output:** Quiz submission result with score recorded.

6. Non-Functional Requirements

Non-Functional Requirements define **how well the system performs**, rather than what the system does. For the **Knowledge Delivery Platform**, these requirements ensure the platform is efficient, scalable, secure, and maintainable.

6.1 Performance

- **Requirement:** API responses must be delivered within **2 seconds** for at least **95% of requests** under standard operating conditions.
- **Rationale:** In an LMS, students and instructors frequently interact with APIs for viewing courses, enrolling, or submitting quizzes. Delayed responses can negatively impact the learning experience. Performance optimizations will be achieved through:
    - Database indexing for faster queries.
    - Caching frequently accessed data (e.g., course lists, student profiles).
    - Load balancing across multiple servers.

6.2 Scalability

- **Requirement:** The system must support **100,000+ concurrent student users** without service degradation.
- **Rationale:** LMS platforms are often deployed at **university or enterprise scale**, where thousands of learners may attempt quizzes or access courses simultaneously. Scalability will be achieved using:
    - Horizontal scaling via **cloud auto-scaling groups (AWS, Azure, GCP)**.
    - Container orchestration (e.g., Kubernetes/EKS).
    - Database sharding or replication to distribute load.

6.3 Security

- **Requirement:**
    - All data transmission must be encrypted using **TLS/HTTPS**.
    - Authentication must be implemented via **JWT (JSON Web Tokens)**.
    - **RBAC (Role-Based Access Control)** must enforce different permissions for Admin, Instructor, and Student roles.
- **Rationale:** LMS systems handle sensitive data (student information, progress, grades). Without strong security, data breaches could expose personal and academic records. Security mechanisms include:
    - Input validation to prevent SQL/NoSQL injection attacks.
    - Role-based permissions for sensitive APIs (e.g., only instructors can create quizzes).
    - Regular security audits and penetration testing.

6.4 Reliability

- **Requirement:** The LMS should maintain **99.9% uptime**, with **automatic failover and backup mechanisms**.

- **Rationale:** Students and instructors may access the system at any time, including during live exams or training. Downtime could result in academic disruptions. Reliability measures include:
    - Database replication across multiple nodes.
    - Server clustering and redundancy.
    - Automated daily backups and disaster recovery policies.

6.5 Usability

- **Requirement:** The system must provide **clear, well-documented APIs** and a user-friendly interface for developers and students.
- **Rationale:** Instructors and admins should easily manage courses, while developers integrating the APIs (e.g., for mobile apps) require documentation with:
    - Example requests and responses.
    - Error handling guidelines.
    - Swagger/OpenAPI specifications for easy testing.

6.6 Maintainability

- **Requirement:** The LMS must adopt a **modular service-based architecture**, separating core services such as **Course Management, Quiz Service, Enrollment Service, and Progress Tracking**.
- **Rationale:** Modular architecture improves maintainability by:
    - Allowing independent upgrades or fixes without affecting the whole system.
    - Supporting continuous integration/continuous deployment (CI/CD).
    - Ensuring long-term system adaptability as requirements evolve.

7. Roles and Responsibilities

- **Admin**

- Create, update, and delete courses.
- Manage user accounts and roles.

- **Instructor**
  - Add quizzes to courses.
  - Track and evaluate student progress.
- **Student**
  - Enroll in courses.
  - Access course materials and attempt quizzes.
  - View personal progress reports.

8. RBAC (Role-Based Access Control Matrix)

Feature	Admin	Instructor	Student
Add Course	✓	✗	✗
View Courses	✓	✓	✓
Enroll Student	✓	✓	✗
Track Progress	✓	✓	✓ (own)
Add Quiz	✗	✓	✗
Attempt Quiz	✗	✗	✓
Submit Quiz	✗	✗	✓

9. Entity Relationship (ER) Design

Entities:

- **Course** (course_id, title, description, instructor, students[], quizzes[])
- **Student** (student_id, name, email, enrolled_courses[], progress[])
- **Quiz** (quiz_id, course_id, questions[], max_score)
- **Progress** (student_id, course_id, completion_percentage, quiz_scores)

Relationships:

- A **Course** can have many **Students**.
- A **Course** can have many **Quizzes**.
- A **Student** can attempt many **Quizzes** and track progress per course.

10. API Documentation

Add Course

- **POST** `/api/courses`
- Request Body:

```
{
  "title": "Data Structures",
  "description": "Learn about arrays, stacks, queues, trees, and
graphs.",
  "instructor": "Dr. John Doe"
}
```

- Response: `201 Created`

Get All Courses

- **GET** `/api/courses`

Enroll Student

- **POST** `/api/courses/:id/enroll?student=Alice`

Update Progress

- **PUT** `/api/courses/:id/progress?student=Alice&progress=70`

Get Quiz

- **GET** `/api/courses/:id/quiz`

Submit Quiz

- **POST** `/api/courses/:id/quiz/submit?student=Alice&score=85`

11. Appendix

Abbreviations

- **LMS** – Learning Management System
- **RBAC** – Role-Based Access Control
- **API** – Application Programming Interface
- **CRUD** – Create, Read, Update, Delete
- **JWT** – JSON Web Token

12. Existing Solutions

Moodle

An open-source LMS widely used in universities. Provides course creation, quizzes, and grading features but requires customization for scalability.

Blackboard

A commercial LMS used in higher education and enterprises. Offers extensive integrations, analytics, and advanced assessment tools.

Google Classroom

Simplifies assignment distribution, grading, and communication but lacks advanced RBAC and deep analytics features.