Project Title: Microservices-Based Learning Platform for Course Content Management and Learner Progress Tracking System

By Simon Gawar Dak

Table of Contents

Figures

1.0 Document Revision History

Document Owner Name	Date	Document Version
Simon Gawar Dak	2025-09-25	1.0.0

2.0 Project Overview

The Microservices-Based Learning Platform is proposed to manage course content and track learner progress. It will provide modular services for instructors, learners, and administrators to interact with educational materials, assessments, and analytics. The platform supports scalability, flexibility, and integration with modern learning standards.

3.0 Stakeholders

- **Learners**: Consume content, complete assessments, and monitor progress.
-**Instructors**: Create and manage content, evaluate learners.
- **Course Administrators:** Oversee operations and ensure content quality.
- **Software Development Team:** Build and maintain the platform.
- **Educational Institutions:** Deploy and evaluate platform effectiveness.

4.0 Capabilities

4.1 Course Management

- **Create, edit, and delete courses:** Provide instructors and administrators to create, update, and remove courses dynamically, ensuring the curriculum stays current and relevant.
- **Assign instructors and manage metadata:** Facilitates the assignment of instructors to specific courses and the management of course metadata (e.g., tags, categories, prerequisites) for better organization and discoverability.
- **Organize modules and learning paths:** Supports the breakdown of courses into modules and the creation of structured learning paths that guide learners through content based on progression rules or competencies.

4.2 Content Delivery

- Upload various forms of multimedia content.
- Deliver content via web and mobile.
- Support shareable Content Object Reference Model (SCORM)/Experience API (xAPI) standards.

4.3 Learner Progress Tracking

- Monitor time spent and completion rates.
- Generate progress reports.

- Visual dashboards for learners and instructors.

4.4 Assessment and Grading

- Create quizzes and assignments.
- Auto-grade objective questions.
- Manual grading for subjective responses.

4.5 Notifications and Feedback

- Automated alerts for deadlines and grades.
- Feedback forms and surveys.
- Push notifications for mobile devices.

4.6 Analytics and Reporting

- Generate performance and engagement reports.
- Visualize trends and export data.
- Predictive analytics for learner outcomes.

5.0 Project Theme

This Microservices-Based Learning Platform project aims to create a highly scalable, modular, and resilient educational technology (EdTech) system. Its central goal is to effectively manage course content and provide detailed tracking of learner progress by decoupling critical business functions into independent microservices.

5.2 Canonical Model and Bounded Contexts

Microservices-Based Learning Platform for Course Content Management and Learner Progress Tracking system is structured around the principles of Domain-Driven Design (DDD) to manage complexity.

5.2.1 Canonical- Data Model (CDM)

The CDM defines the **shared language** for communication between services, ensuring that core entities are understood consistently across the platform. UUIDs are critical for linking data across independent service databases.

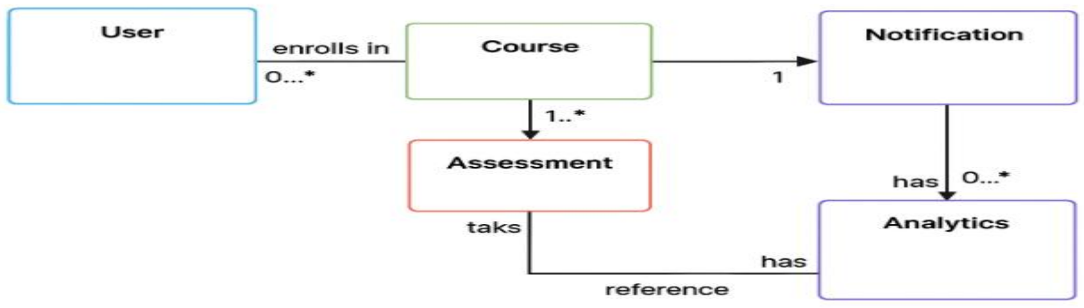

Figure 1: Canonical Data Model

The table summarized the data model above with the involving entities: **User, Course, Assessment, Notification, and Analytics**, including their descriptions and relationships.

Entity	Description	Relationships
User	Represents a person using the system.	- Enrolls in 0..* Courses - Takes 0..* Assessments - Receives 0..* Notifications - Has 0..* Analytics records
Course	Represents a learning module or subject.	- Has 1..* Assessments - Enrolled by 0..* Users - Referenced in Notifications and Analytics
Assessment	Represents a test or evaluation within a course.	- Belongs to 1 Course - Taken by 0..* Users - Referenced in Analytics
Notification	Represents messages sent to users about updates or actions.	- Sent to 1..* Users - May reference 0..1 Course or Assessment
Analytics	Represents performance or usage data for a user.	- Belongs to 1 User - May reference 0..1 Course - May reference 0..1 Assessment

The explanation of the relationships between the entities in the data model:

Relationship	Explanation
User ↔ Course	A user can enroll in multiple courses (0..*), and each course can have multiple users enrolled. This is a **many-to-many** relationship.
Course → Assessment	Each course contains one or more assessments (1..*). This is a **one-to-many** relationship from Course to Assessment.
User ↔ Assessment	A user can take multiple assessments, and each assessment can be taken by multiple users. This is a **many-to-many** relationship.

User → Notification	A user can receive multiple notifications (0..*). Each notification is sent to one or more users. This is a **one-to-many** relationship.
Notification → Course/Assessment	A notification may optionally reference a course or an assessment (0..1). This allows contextual alerts (e.g., "New assessment available").
User → Analytics	Each user can have multiple analytics records (0..*), tracking their performance or activity. This is a **one-to-many** relationship.
Analytics → Course/Assessment	Analytics may reference a specific course or assessment (0..1), allowing performance tracking per course or per assessment.

5.2.2 Bounded Contexts (BCs)

The Bounded Contexts (BCs) define the functional boundaries of each microservice, encapsulating its own database and business logic to ensure clear separation of concerns and domain consistency.

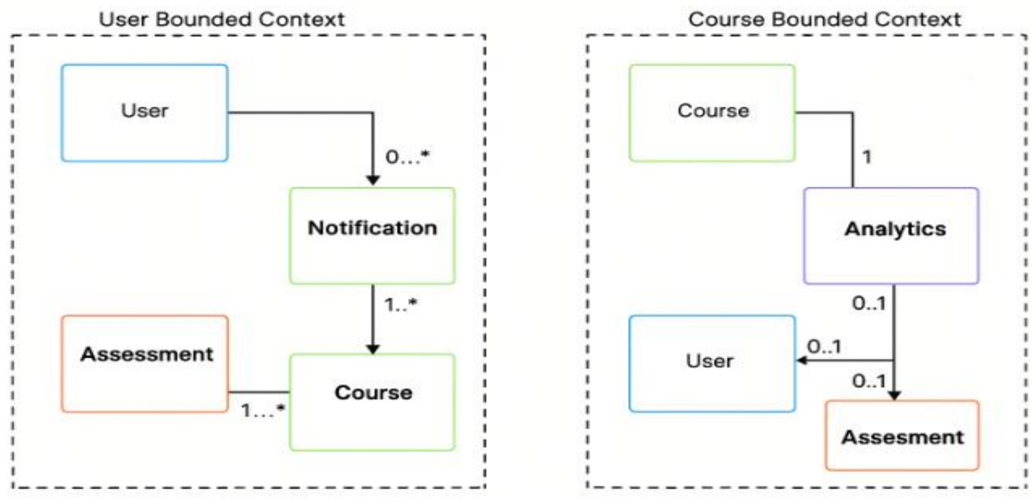

Figure 2: Bounded Contexts

A Bounded Contexts shown above is a central pattern in Domain-Driven Design (DDD). It describes a logical boundary within which a particular domain model is defined and applicable. This boundary inside has:

- The Terms, rules, and logic have specific meanings.
- The model is consistent and cohesive.

- The bounded context model helps to avoid ambiguity when the same terms are used differently in different parts of a system.

The table below summarizes the Bounded Contexts based on the domain model:

Bounded Context	Entities	Responsibilities
User Context	User, Notification	Manages user profiles, authentication, and delivery of system or course notifications.
Course Context	Course, User	Handles course creation, enrollment, and user-course associations.
Assessment Context	Assessment, Course, User	Manages assessments, their linkage to courses, and user participation.
Notification Context	Notification, User, Course, Assessment	Sends alerts related to course updates, assessments, or system-wide events.
Analytics Context	Analytics, User, Course, Assessment	Tracks user performance, engagement, and assessment outcomes.

We need context maps to divide the entire microservice learning platform complex system into manageable parts, align software design with business domains, enable developer (s) to work independently on different parts of the system and reduce coupling and improve scalability.

5.3 Sketch of the Aimed Deployment

The deployment strategy will be cloud-native, utilizing **Docker** for packaging for this case and a container orchestrator (like Kubernetes or Docker Swarm) for scaling and management.

Component	Technology	Role in Deployment
Microservices	Java/Spring Boot (or similar)	Packaged as individual **Docker images** and deployed as independent services.
API Gateway	Spring Cloud Gateway/Zuul/NGINX	Acts as the **single-entry point**, managing request routing, authentication, and rate limiting.
Service Discovery	Eureka/Consul	Allows services to find each other without hardcoding URLs.
Databases	MySQL	Each service has its own isolated data store, ensuring autonomy.

Deployment Target	Docker Compose (Development) / **Kubernetes (Production)**	Manages the lifecycle, scaling, and networking of all containers.

5.3.1 Microservices High-Level Deployment Diagram

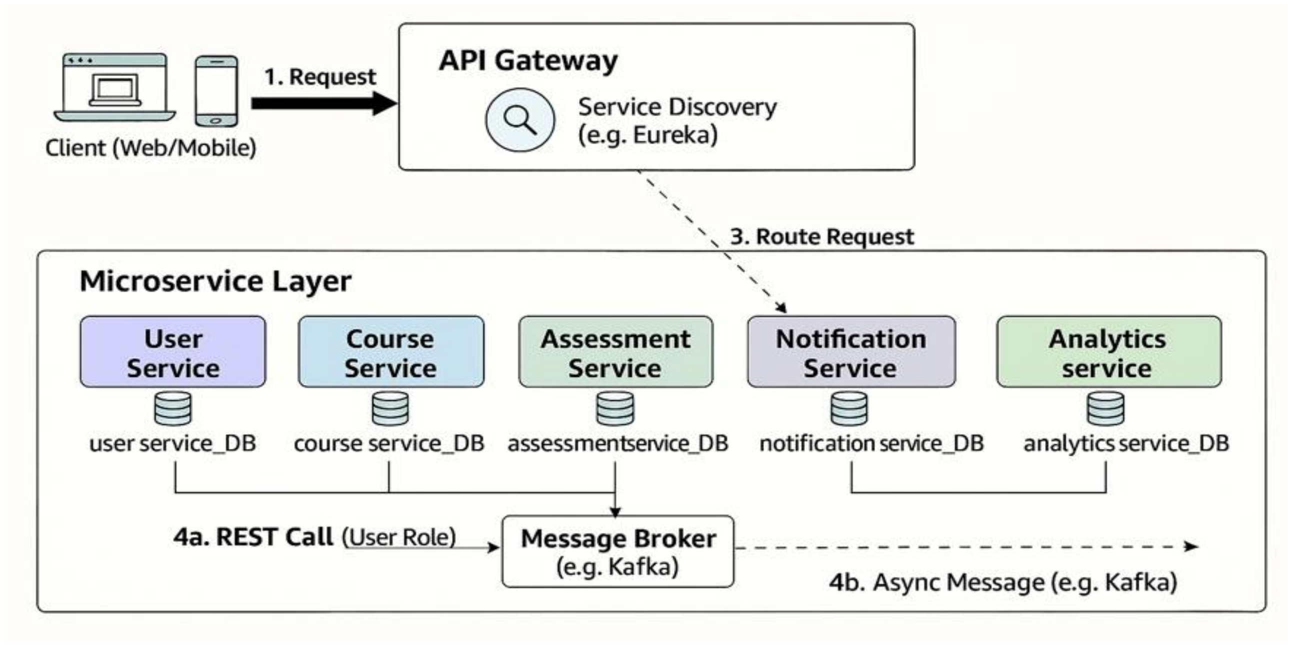

Figure 3: Microservices high-level deployment diagram

The platform uses the **API Gateway pattern** and **Service Discovery** to manage client-to-service communication.

i. **Client (Web/Mobile)** sends a request to the **API Gateway**.
ii. **API Gateway** uses **Service Discovery** to find the correct instance of the target microservice (e.g., Course Service).
iii. Microservice interacts only with its corresponding **database**. This means that each microservice is now explicitly connected to its **own database:**

- User Service → user service_DB
- Course Service → course service_DB
- Assessment Service → assessment service_DB
- Notification Service → notification service_DB

- Analytics Service → analytics service_DB

iv. Inter-service communication (e.g., Course Service needing a user's role) is handled via REST calls or, for asynchronous operations, a **Message Broker** (e.g., Kafka). The **REST calls** for synchronous communication (such as Course Service requesting user roles) whereas **Message Broker (e.g., Kafka)** for asynchronous messaging between services.

5.4 Application Prototype (Postman Workspace)

This is key endpoints and the structure of a Postman Collection required to test the RESTful API contract. The system relies on the **API Gateway** (assumed to run on http://localhost:8080) to route traffic.

Postman Collection Structure and Requests

Service Route	Method	Endpoint	Description	Expected Status
User	POST	/users/register	Create a new Learner account.	201 Created
User	GET	/users/profiles/{{user_id}}	Retrieve a user's profile and role.	200 OK
Course	POST	/courses/create	Instructor creates a new course. **Requires instructor_id**	201 Created
Course	GET	/courses/{{course_id}}	Retrieve full course structure (modules/content).	200 OK
Assessment	POST	/assessments/submit	Learner submits a quiz attempt.	202 Accepted
Analytics	GET	/analytics/progress/{{user_id}}	Retrieve a learner's overall progress dashboard.	200 OK

Postman Pre-requisite: The first step in the collection would be an authentication request to the User Service to get a **JWT Token**, which would then be set as a **Bearer Token** environment variable for all subsequent requests.

5.5 Docker Deployment and Profiles

We use Docker to package the microservices and **Docker Compose** to deploy them locally, using different profiles for distinct environments.

docker-compose.yml Configuration (Structure)

This file defines the services, networks, and volumes. We use **profiles** to manage environment-specific dependencies. These configurations were detailed in the docker-compose yml file for all microservices.

6.0 Demonstrating Two Profiles

The two profiles demonstrate how different environments (Dev vs. Test) can be managed with a single deployment file.

Profile Name	Command to Run	Purpose	Key Difference
dev	docker compose --profile dev up	Local development and rapid iteration. Services connect to the **db-dev** container.	Uses the db-dev container with verbose logging and debug settings enabled in the service containers.
test	docker compose --profile test up	Running automated end-to-end tests. Services connect to the **db-test** container.	Uses the **db-test** container, which can be easily torn down and recreated for fresh test runs, and disables non-critical services (like the Notification service).

This approach allows me to use the same Docker tooling and configuration file while ensuring that data and service configurations are isolated per environment.