**Scenario:**

The Automat ion team of a leading Learning Organization is planning to come up with a "Learning Platform/App"

Architect, senior developers and other stakeholders got into several huddle up sessions last week to come up with a high-level overall Functional/ Non-Funct ional/Architectural requirement.

Though not structured at this point of time,the essence/summary of those discussions are as below:

1. We can have 3 or 4 types of users:

# Content Authors/ Content Creators:

* + People who come, sign-up on your site and create content. (Content authors/Content creators)
  + They can create a complete course, which can have multiple

modules/lessons/v ideos/biogs/art icles etc. or create some short learning nuggets.

# Users:

* + General anonymous users who come to the site to view details about the different courses/packages available and may purchase courses/package
* First time they make a purchase, they need to sign-up. Once they sign-up they become customers of the site (Learners).
* Signing up without making an immediate purchase also is possible.

# Learners:

* A User becomes a Learner if they purchase a course/package.
* After making a purchase, they can view their courses and access them.
* Multiple purchases is possible.

# Site Admin:

* Is the user who will do analysis of the types of users who are part of this site.
* They Create packages,set prices on them, Apply/Offer discounts, block/unblock users etc.,

(Dual/multiple roles are also possible. E.g.: "Content Creator" might create a course and also buy another course as a "Learner" viz.

# Probable Functionalities:

**a. Registration and Login**

1. **Payment:** For Users, payment is a critical functionality that can be run as a stand-alone service because at any cost, this service should not go down. Although the traffic might not be very high,the funct ionality is critical,and we cannot afford it to go down at any cost. Letting it go down is sure business loss.
2. **Content Creation:**Content Authors should be able to create content while logged in as "Content Creator",like create a course,add modules into the course etc

**d. Course Access:**Registered Users (Learners) access their courses, watch videos etc., the videos are heavy, they consume band-width and other comput ing resources, so this could be spun off as another stand-a lone service.

1. Relevant Reports should be made available for each role as deemed necessary.

**Probable Microservices:**

The team agrees that the Microservices architecture needs to be used here and few probable microservices could be:

* 1. Payment
  2. Registration and Login can be created as a stand-a lone service or keep it as part of the e-commerce piece itself (coming onto the site,looking for products, looking for course details,adding courses to cart, signing-up ,logging in can also be put into one single service.
  3. Course Creation
  4. Course Access - viewing/watching the course (Assuming that this requires its own bandw idth and may have to be scaled independently of the other funct ionalities)

**Technology Stack/Architectural selection:**

1. Java based Microservices with Spring Boot
2. Database - Any SQL/No-SQL Database.

**You, playing the role of an Architect & Senior Developer of this project need to:**

* 1. Architect the entire learning portal,its funct ionalities based on the above discuss ions.

1. Architect and implement it as Java based Microservices with Spring Boot.
2. Use relevant additional distributed projects to manage the distributed nature of the application
   1. Service Discovery and Registry (Hint: Eureka)
   2. Client-side load balancing (Hint: Ribbon)
   3. Fault Tolerance and circuit breaker (Hint: Hystrix)
   4. Centralized configurat ion (Hint: Config Server

Step 1: Architectural Design

1.1 Identify Users and Roles:

Content Authors/Content Creators

Users

Learners

Site Admin

1.2 Define Functionality Overview:

Registration and Login:

Standalone service or part of e-commerce.

Payment:

Critical standalone service.

Content Creation:

Microservice for Content Authors.

Course Access:

Microservice for Learners.

Reports:

Generate relevant reports for each role.

1.3 Identify Microservices:

Payment Service:

Critical for business; standalone.

User Service:

Manages registration and login.

Content Creation Service:

Allows Content Authors to create and manage content.

Course Access Service:

Manages access to courses for Learners.

1.4 Additional Distributed Projects:

Service Discovery and Registry (Eureka):

Enables services to discover and register with each other.

Client-side Load Balancing (Ribbon):

Distributes load among instances of services.

Fault Tolerance and Circuit Breaker (Hystrix):

Ensures system reliability and resilience.

Centralized Configuration (Config Server):

Manages configuration centrally.

Security (OAuth/JWT):

Implements secure authentication and authorization.

API Gateway (Spring Cloud Gateway):

Acts as the entry point for microservices, handles routing, and provides security.

REST Consumers (Feign):

Simplifies RESTful service consumption.

Secrets Management (Hashicorp Vault):

Safely stores and manages secrets.

Step 2: Implementation

2.1 Develop Microservices:

Implement each microservice using Java and Spring Boot.

Configure Eureka for service registration and discovery.

Set up Ribbon for client-side load balancing.

Implement Hystrix for fault tolerance.

Use Config Server for centralized configuration.

Implement security using OAuth or JWT.

Utilize Spring Cloud Gateway as the API Gateway.

Implement Feign for simplified REST consumption.

Integrate Hashicorp Vault for secrets management.

Step 3: Test Cases

Develop unit tests for each microservice.

Implement integration tests to ensure proper communication between microservices.

Test security mechanisms.

Validate load balancing and fault tolerance under stress conditions.

Ensure proper functionality of each service individually and in collaboration.

1. Security (Hint: Basic Auth/OAuth/JWT}
2. API Gateway (Hint: Spring Cloud Gateway/Zuul)
3. REST Consumers (Hint: Feign}
4. Store secrets in Vaults (Hint: Hashicorp Vault}

**Final Deliverables expected:**

1. Overall architectura l design document with appropriate support documents if any { Include "why" such architectural decisions were made}

1. Final source code of the implemented solution .
2. Test Cases
3. Screen shots/grabs of applicat ion flow Role wise capturing important functional ities of the application.