## TekGain E-Learning (Spring Boot + Microservices)

# 1.0Introduction

# 1.0Functional Requirements

TekGain is one of the popular training institutes that have branches all over India. Due to covid restrictions, TekGain planned to initiate E-Learning activities that help candidates all over the world to gain benefit. As an initiate they are in need to of developing an application that supports the below business activities.

* Course Service
* Associate Service
* Admission Service

The client wishes to have microservices to be created for Course , Associate and Admission , so that each team can work on independent services which will help to deploy the services more quickly.

Help them to automate the above process by developing Rest Service using Maven and incorporate microservices for the same. The front end should be designed using Angular.

You are also required to apply security for the above services. Develop the security as separate service and apply the security.

# 2.0 Technical Specifications

**REST SERVICE LAYERING**

1. **Rest Services:**

# 2.0.1 Course Service

The provided CourseController which is a RestController, should be created with the below services. All the below endpoints should return the ResponseEntity with below status:

1. For success – HttpStatus.OK
2. For Exception – HttpStatus.NOT\_FOUND
3. For Authentication and Authorization failure- HttpStatus.UNAUTHORIZED

|  |  |  |
| --- | --- | --- |
| **Request Method** | **Request Url** | **Description** |
| Post | /course/addCourse | This service should add the course details by using the addCourse method of the CourseServiceImpl class which internally uses the CourseRepository to persist the data. |
| Put | /course/update/{courseId,courseFees} | This service should modify the course fees of the given course id and should return the updated course details, by using the updateCourse method of the CourseServiceImpl class which internally uses the CourseRepository |
| Get | /course/viewByCourseId/{courseId} | This service should retrieve the course information of the given courseId by using the viewByCourseId method of the CourseServiceImpl which internally uses the CourseRepository class |
| Get | /course/viewFeedback/{courseId} | This service should retrieve the feedback rating entered by all the associates as a list for the given course. |
| PUT | /course/calculateAverageFeedback/{courseId,rating} | This service should calculate the average feedback for the given course and update the feedback rating for the given course by using the calculateAverageFeedbackAndUpdate method of the CourseServiceImpl class which internally uses the CourseRepository |
| DELETE | /course/deactivate/{courseId} | This service deactivates the given course by using the deactivateCourse method of the CourseServiceImpl class which internally uses the CourseRepository.  **Note:** While deactivating the course, it should deactivate the admission that happened for that course. **This service should be implemented by performing intermicroservice communication with admission service for deactivating the admission** |
| GET | /course/viewAll | This service returns the list of courses using the viewAll method the CourseServiceImpl class which internally uses the CourseRepository |

# 2.0.2Associate Service

The provided AssociateController which is a RestController, should be created with the below services. All the below endpoints should return the ResponseEntity with below status:

1. For success – HttpStatus.OK
2. For Exception – HttpStatus.NOT\_FOUND
3. For Authentication and Authorization failure- HttpStatus.UNAUTHORIZED

|  |  |  |
| --- | --- | --- |
| **Request Method** | **Request Url** | **Description** |
| Post | /associate/addAssociate | This service should add the associate details by using the addAsscoiate method of the AssociateServiceImpl class which internally uses the AssociateRepository to persist the data. |
| Put | /associate/updateAssociate/{associateId,associateAddr} | This service should modify the associate address of the given associate id and should return the updated associate details , by using the updateAssociate method of the AssociateServiceImpl class which internally uses the AssociateRepository |
| Get | /associate/viewByAssociateId/{associateId} | This service should retrieve the associate information of the given associate id by using the viewByAssociateId method of the AssociateServiceImpl which internally uses the AssociateRepository class |
| GET | /associate/viewAll | This service returns the list of associates using the viewAll method the AssociateServiceImpl class which internally uses the AssociateRepository |

**2.0.3 Admission Service**

The provided AdmissionController which is a RestController, should be created with the below services. All the below endpoints should return the ResponseEntity with below status:

1. For success – HttpStatus.OK
2. For Exception – HttpStatus.NOT\_FOUND
3. For Authentication and Authorization failure- HttpStatus.UNAUTHORIZED

|  |  |  |
| --- | --- | --- |
| **Request Method** | **Request Url** | **Description** |
| Post | /admission/register/{associateId}/{courseId} | This service should register the given associate for the given course by using the registerAssociateForCourse method of the AdmissionServiceImpl class which internally uses the AdmissionRepository to persist the registration info as regNo, associateId, courseId and the fees  **Note:**  **1.This service should be implemented by performing intermicroservice communication with the Associate Service and the Course Service for checking the existence of the Associate and the Course for the given associate Id and course Id.**  **2. Once the registration is done, the registration details like the associateId, courseId and the registration Number should be sent to the queue using the Rabbit MQ Message Broker. Please refer to Message-Broker Section for further technical details to implement this functionality.**  **3. After posting the messages in the queue, the mail should be sent to the associate email-id with the registration details. Use JavaMailSender API to send mail** |
| Put | /admission/calculateFees/{associateId} | This service should calculate fees for all the courses registered by the given associate and return the fees, by using the calculateFees method of the AdmissionServiceImpl class which internally uses the AdmissionRepository. |
| Post | /admission/feedback/{regNo,feedback,feedbackRating} | This service should add the feedback for the course associated with the given regNo by using the addFeedback method of the AdmissionServiceImpl which internally uses the AdmissionRepository class. Also, for the given regNo fetch the corresponding courseId and pass the courseId and feedbackRating to the service calculateAverageFeedback of Course service  **Note:** **This service should be implemented by performing intermicroservice communication with course service for calculating the average feedback.** |
| Get | /admission/highestFee/{associateId} | This service should return the course for which the associate has paid the highest fee among the registered courses by using the highestFeeForTheRegisteredCourse method of the AdmissionServiceImpl which internally uses the AdmissionRepository class |
| Get | /admission/viewFeedbackByCourseId/{courseId} | This service should return the feedback as a list for the given course id by using the viewFeedbackByCourseId method of the AdmissionServiceImpl which internally uses the AdmissionRepository class. |
| Delete | /admission/deactivate/{courseId} | This service should deactivate the admissions made for the given course by using the deactivateAdmission method of the AdmissionServiceImpl which internally uses the AdmissionRepository class. |
| Post | /admission/makePayment/{registartionId, fees} | This service should make the payment for the given registration by using the makePayment method of the AdmissionServiceImpl. |
| GET | /admission/viewAll | This service returns the list of admissions using the viewAll method the AdmissionServiceImpl class which internally uses the AdmissionRepository |

**2.0.4 AuthenticationAuthorization Service**

The provided AuthController which is a RestController, should be created with the below services:

|  |  |  |
| --- | --- | --- |
| **Request Method** | **Request Url** | **Description** |
| Post | /app/signin | This service should validate the given user details, if valid, it should generate the token and return the token. This service should internally use the UserRepository to check the availability of the user in the database. If the username and password is valid, get the JWT token as the response. The JWT response will contain the token,token type, username and the role |
| Get | /app/validateToken/{authorization} | This service should validate the given token, if valid it should allow to access the services available in Course, Associate and Admission service. **Note**: The Authorization should hold the token prefix by Bearer.  **Example:** Bearer <<token>> |

**Please refer the Spring Security section for more details**

**Note: CourseService should use WebClient for intermicroservice communication with the securityservice for token validation**

**Note: AssociateService should use RestTemplate for intermicroservice communication with the securityservice for token validation**

**Note: AdmissionService should use FeignClient for intermicroservice communication with the securityservice for token validation**

1. **Service Layer**

# Service Layer for Course Service

Refer to the ICourseService interface provided as part of the code skeleton. The CourseServiceImpl class which is provided as part of the code skeleton must realize all the methods in the ICourseService interface.

For each method, use SLF4J along with Spring AOP to log the success and failure message into the log file named “log/mylog.log”. For the success case, log the message as “The method <<methodname> has completed successfully and the log level should be info. For the Exception scenario, log the message given in each method’s exception column and the log level should be error. Note: **Use Lombok annotations for logging**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | **Method in Service** | **Description** | **Exception** |
| Requirement 1 | addCourse | This method should add the course details to the database by using the CourseRepository and return the Course object. The course id should get generated automatically using the sequence. The sequence for the courseid should from 200 and increment by 1. | If the given course id already exist, then throw CourseInvalidException with the message **“CourseId already exists”.** |
| Requirement 2 | updateCourse | This method should update the course fees for the given course id by retrieving the existing course from the database. The method should then return the course information with the updated fees. | If the given course id does not exist, then throw CourseInvalidException with the message **“CourseId does not exists”.** |
| Requirement 3 | viewByCourseId | This method should return the course information available in the database for the given courseId | If the given course id does not exist, then throw CourseInvalidException with the message **“Invalid Course Id”.** |
| Requirement 4 | findFeedbackRatingForCourseId | This method should return the feedback rating for the given course id | If the given course id does not exist, then throw CourseInvalidException with the message **“CourseId does not exists”.** |
| Requirement 5 | calculateAverageFeedbackAndUpdate | This method should calculate the average feedback for the given course and update the feedback rating for the given course in the database | If the given course id does not exist, then throw CourseInvalidException with the message **“CourseId does not exists”.** |
| Requirement 6 | deactivateCourse | This method should deactivate the given the course in the database | If the given course id does not exist, then throw CourseInvalidException with the message **“CourseId does not exists”.** |
| Requirement7 | viewAll | This method should return the list of courses available in the database |  |

# Service Layer for Associate Service

Refer to the IAssociateService interface provided as part of the code skeleton. The AssociateServiceImpl class which is provided as part of the code skeleton must realize all the methods in the IAssociateService interface.

For each method, use SLF4J along with Spring AOP to log the success and failure message into the log file named “log/mylog.log”. For the success case, log the message as “The method <<methodname> has completed successfully and the log level should be info. For the Exception scenario, log the message given in each method’s exception column and the log level should be error. Note: **Use Lombok annotations for logging**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | **Method in Service** | **Description** | **Exception** |
| Requirement 1 | addAssociate | This method should add the associate details to the database by using the AssociateRepositoryand return the Associate object.  The associate id should get generated automatically using the sequence. The sequence for the associate id should from 100 and increment by 1. | If the given associate id already exists, then throw AssociateInvalidException with the message **“AssociateId already exists”.** |
| Requirement 2 | updateAssociate | This method should update the associate address for the given associate id by retrieving the existing associate information from the database and update the same to the database. The method should then return the associate information with the updated address. | If the given associate id does not exist, then throw AssociateInvalidException with the message **“AssociateId does not exists”.** |
| Requirement 3 | viewByAssociateId | This method should return the associate information available in the database for the given associateId | If the given associate id does not exist, then throw AssociateInvalidException with the message **“Invalid Associate Id”.** |
| Requirement 4 | viewAll | This method should return the list of associates available in the database |  |

# Service Layer for Admission Service

Refer to the IAdmissionService interface provided as part of the code skeleton. The AdmissionServiceImpl class which is provided as part of the code skeleton must realize all the methods in the IAdmissionService interface.

For each method, use SLF4J along with Spring AOP to log the success and failure message into the log file named “log/mylog.log”. For the success case, log the message as “The method <<methodname> has completed successfully and the log level should be info. For the Exception scenario, log the message given in each method’s exception column and the log level should be error. Note: **Use Lombok annotations for logging**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | **Method in Service** | **Description** | **Exception** |
| Requirement 1 | registerAssociateForCourse | This method should add the valid associate id , valid course id, fees along with the unique registration number to the database by using the AdmissionRepository and return the Admission object. The admission id should get generated automatically using the sequence. The sequence for the associate id should from 300 and increment by 1. | If the given registration number already exists, then throw AdmissionInvalidException with the message **“AdmissionId already exists”.** |
| Requirement 2 | calculateFees | This method should calculate fees for all the registered courses for the given associate Id. | If the given associate id does not exist, then throw AdmissionInvalidException with the message **“AssociateId does not exists”.** |
| Requirement 3 | addFeedback | This method should add the feedback for the given registration number into the database and return the Admission object | If the given registration number does not exist, then throw AdmissionInvalidException with the message **“Invalid Registration Id”.** |
| Requirement 4 | highestFeeForTheRegisteredCourse | This method should retrieve all the registered courses from the db for the given associate Id. This method should then find the course that has highest fee and return as a List.  **Note:**  There can be more than one course with the highest fee. | If the given associate id does not exist, then throw AdmissionInvalidException with the message **“AssociateId does not exists”.** |
| Requirement 5 | viewFeedbackByCourseId | This method should return the feedback from the database for the given course id. | If the given course id does not exist, then throw AdmissionInvalidException with the message **“Invalid Course Id”.** |
| Requirement 6 | deactivateAdmission | This method should deactivate the admission registered for the given course in the database. | If the given course id does not exist, then throw AdmissionInvalidException with the message **“CourseId does not exists”.** |
| Requirement 7 | makePayement | This method should make the payment for the given registration . |  |
| Requirement 8 | viewAll | This method should return the list of admissions available in the database |  |

# Repository

CourseRepository

CourseRepository should be a MongoRepository to perform the CRUD operations as per the requirements specified in the case study

AssociateRepository

AssociateRepository should be a MongoRepository to perform the CRUD operations as per the requirements specified in the case study

AdmissionRepository

AdmissionRepository should be a MongoRepository to perform the CRUD operations as per the requirements specified in the case study.

**Exception Class**

Refer to the CourseInvalidException, AdmissionInvalidException, AssociateInvalidException class provided as part of the code skeleton. This class should extend the Exception class.

**GLOBAL EXCEPTION HANDLING**

To incorporate the messages related to Exception as ResponseEntity, you are provided with the below classes

* Make use of ExceptionResponse class that is already provided as part of the code skeleton to send the customized response error message.
* CustomizedResponseEntityExceptionHandler class to handle all the exceptions that has occurred in the application(both user defined and pre defined)

|  |  |
| --- | --- |
| **Method Name** | **Explanation** |
| handleAllExceptions | This method should generally handle all the exceptions. It should return ResponseEntitywhich include ExceptionResponse object and the status as HttpStatus.INTERNAL\_SERVER\_ERROR |
| handleNotFoundException | This method should handle and provide customized error message using Exception Response class for user defined exception CourseInvalidException for CourseService, AdmissionInvalidException for AdmissionService and AssociateInvalidException for AssociateService  It should return ResponseEntity which include ExceptionResponse object and the status as HttpStatus.NOT\_FOUND |

**Note :**

**In CustomizedResponseEntityExceptionHandler class, log all the error messages using SLF4J.Use Lombok annotations for logging**

# Model Class

Refer to the Course class provided as part of the code skeleton.

|  |  |
| --- | --- |
| **Attributes** | **DataType** |
| courseId | String |
| courseName | String |
| fees | Integer |
| duration | Integer |
| courseType | String |

Refer to the Associate class provided as part of the code skeleton.

|  |  |
| --- | --- |
| **Attributes** | **DataType** |
| associateId | String |
| associateName | String |
| associateAddress | Integer |
| associateEmailId | String |

Refer to the Admission class provided as part of the code skeleton.

|  |  |
| --- | --- |
| **Attributes** | **DataType** |
| registrationId | Integer |
| courseId | Integer |
| associateId | Integer |
| fees | Integer |
| feedback | String |

**Note:**

* **Use Lombok to generate no-arg-constructor getters and setters for the attributes.**
* **Do not change the datatype or the attribute name provided as part of the code skeleton.**

# MICROSERVICE SPECIFICATION:

* Spring boot application should contain all the REST services implementation as per the case study specification.
* The Spring rest service applications should have the below application names as below in the application.properties:
  + Course Service should have the name as **courseapp** running in 9091
  + Associate Service should have the name as **associateapp** running in 9092
  + Admission Service should have the name as **admissionapp** running in 9093

**Eureka Registry**

* The AssociateService , CourseService and AdmissionService should be registered in the Eureka Registry. The Registry should run in the port number 8761.

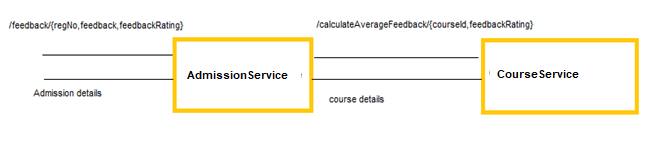
**API Gateway:**

* Implement API Gateway using Spring cloud API Gateway
* The API gate way server port(8777)must be specified in the “application.properties” file.
* The Gateway should be configured to route only the services of courseapp, admissionapp and associateapp.

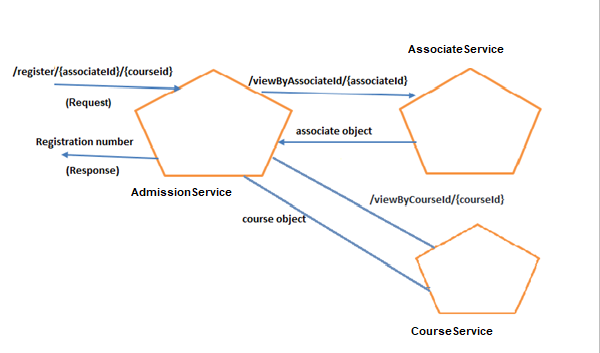
**Intermicroservice Communication**

AdmissionService and CourseService perform intermicroservice communication to handle the below client requests:

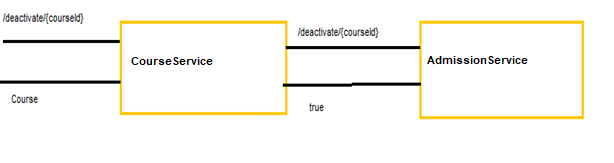
* viewFeedback request for AdmissionService



* register request for AdmissionService



* delete request for Course.



**Fault Tolerance**

When deactivate end point of CourseService is invoked, it deletes the course details of the particular course id in the Course service and then performs intermicroservice communication with the admission by invoking the /deactive end point of the AdmissionService to delete all the admissions made for the given course id. Handle intermicroservice communication failure by returning the status **HttpStatus.SERVICE\_UNAVAILABLE**

**PAYMENT GATEWAY**

1.Use Paypal to provide payment gateway for the making payment.

2.You need PayPal Account. Follow below steps:

2.1 Create official PayPal account

2.2 Login to PayPal’s developer portal by using this link: https://developer.paypal.com/developer/applications

2.3 Create new App by using this link: https://developer.paypal.com/developer/applications/create

2.4 Get ClientID and ClientSecret that we'd like in our program to get paypalContext

Please find below the various classes involved and its need

com.payment.Order – DTO class

com.payment.PaypalConfig – For Loading the paypal configuration details

com.payment.PaymentController – Controller to initiate the payment

com.payment.PaymentService - Service to create and process the payment

|  |  |
| --- | --- |
| **API** | **Explanation** |
| com.payment.Order | Includes all the attributes such as the price, currency,method,intent and descripted entered by the user |
| com.payment.PaypalConfig | Used for loading the client details like client id, client secret,mode  **paypalSdkConfig**() – Configure the mode  **oAuthTokenCredential**()- Create AuthToken by using client details  **apiContext**() - create APIContext and pass Authentication token and set the mode |
| com.payment.PaymentService | Inject API context  **createPayment**()-create the Amount,Transaction and Payer.  Intiate the Payment and redirect to paypal payment api  **executePayment**()-It invokes the paypal payment processing method |
| com.payment.AdmissionController | Inject PaymentService  use /makePayment/{registartionId} end point to initiate the payment |

**MESSAGE BROKER – RABBITMQ**

**Diagram

Description automatically generated**

After successful registration by using the /register service, the registration details such as the registration number , courseId, associate id and the date of registration should be send as a message to the message broker, which can be later consumed by the mailer system to send an email notification to the associate

**Specifications:**

* To publish an object into a queue using rabbit MQ, you are provided with the RabbitMQConfig class which is responsible for creating the connection, queue, exchange and binding the exchange with the queue.
* Inside the method connectionFactory() establish the connection by providing the host,port,username,password and virtualhost and return the ConnectionFactory.
* Inside the queue() method, create a queue with the name given in the application.properties file. The details about the host,port,username,password, virtualhost and queuename are given in the application.properties file. Please do not alter the same.
* Use only RabbitMQ API to perform the messaging.

**SPRING SECURITY**

**STEPS:**

* Before invoking any endpoint of the service, invoke the AuthenticationAuthorizationService and get the token
* **Set the token in the header of each request. In the header add the key as “Authorization” and the value as “Bearer <<token>>”**
* Each service should first validate the token.
* Once the token is authenticated and authorized, the appropriate service (end point) should be invoked, and the response should be returned to the user.
* If the token is not authenticated and authorized, then each end point should return the status HttpStatus.UNAUTHORIZED

**UserDetailsServiceImpl** -[UserDetailsServiceImpl returns the UserDetails object that can be used by Spring for Authentication and Validation].  Override the below method

 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException

Use the UserRepository which is a MongoRepository . The user details -username,password and the role is already persisted in the mongo db.

 Check if the user is already present in the db if so, return the user if not throw UsernameNotFoundException with the message User Not Found with username: " + username

Invoke the getUser method of UserDetailsImpl and return the UserDetails that has the username , password and authorities

**Note**: The users table has been created already and the records in the users table are:

|  |  |  |
| --- | --- | --- |
| Username | Password | Role |
| Admin | admin123 | ROLE\_ADMIN |
| John | Johnpeter | ROLE\_USER |

**Authorization:**

To perform Authorization, you are provided with AdminController and UserController. All those endpoints that should be accessed via user token has to be configured in the UserController with the class level mapping /api/user and all those endpoints that should be accessed via the admin token has to be configured in the AdminController with the class level mapping /api/admin. Please refer to the end point names in the appropriate Service description.

**For example**, there is a endpoint course /course/addCourse in CourseService, then the corresponding end point in the AdminController for Authorization in the AuthenticationAuthorization will be

/api/admin/course/addCourse.

**Script File**

**Use the below script file contents to create users with the role in the mongo database**

db.users.drop();

db.users.insert([{"username" : "admin", "email" : "admin@info.com", "password" : "admin123", "role" : "ROLE\_ADMIN", "\_class" : "com.model.Login"}]);

db.users.insert([{"username" : "john", "email" : "john@info.com", "password" : "johnpeter", "role" : "ROLE\_USER", "\_class" : "com.model.Login"}]);

**UNIT TESTING**

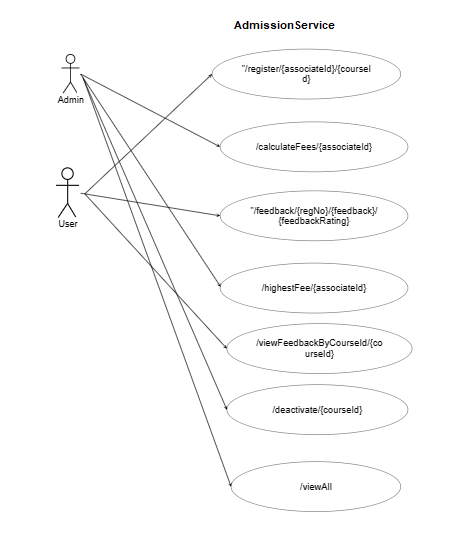
# Perform unit testing for various methods inside the Service Layer for each Service by writing Junit test cases. Test class is provided as part of the code skeleton

|  |  |
| --- | --- |
| **Service** | **Test class** |
| CourseService | CourseServiceImplTest |
| AssociateService | AssociateServiceImplTest |
| AdmissionService | AdmissionServiceImplTest |

**SPRING TESTING**

# Perform Spring test for various service method inside the Controller class. Test class is provided as part of the code skeleton.

|  |  |
| --- | --- |
| **Controller Class** | **Test class** |
| CourseController | CourseControllerTest |
| AssociateController | AssociateControllerTest |
| AdmissionController | AdmissionControllerTest |



A picture containing text, diagram, line, font

Description automatically generated

A picture containing text, diagram, line, white

Description automatically generated

# 3.0 Process Flow

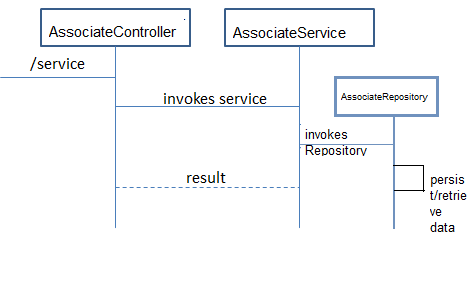
# 3.0.1 Process Flow for Course Service

Diagram

Description automatically generated

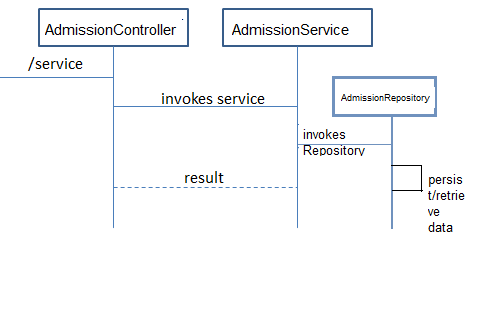
* Client invokes the required service.
* CourseController invokes the method of the CourseServiceImpl.
* CourseServiceImplperforms the service and returns the data back with the help of CourseRepository.
* CourseRepository should a MongoRepository that performs CRUD operations as the case study requirements
* To perform **logging**, you are provided with LogingAspect class. Log the success message and failure message using Slf4j and AOP. Configure logging using **Slf4j** annotation with Lombok in LoggingAspect class ,

# 3.0.2 Process Flow for Associate Service



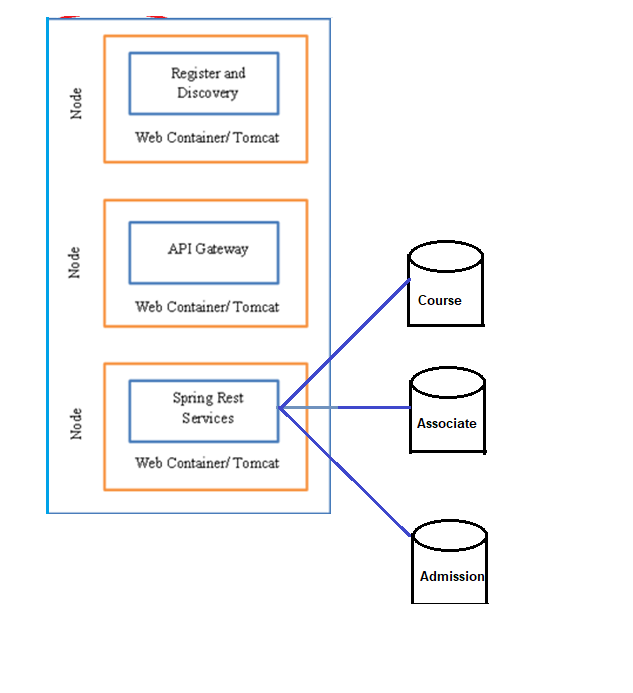
* Client invokes the required service.
* AssociateController invokes the method of the AssociateServiceImpl.
* AssociateServiceImpl performs the service and returns the data back with the help of AssociateRepository.
* AssociateRepository should a MongoRepository that performs CRUD operations as the case study requirements
* To perform **logging**, you are provided with LogingAspect class. Log the success message and failure message using Slf4j and AOP. Configure logging using **Slf4j** annotation with Lombok in LoggingAspect class ,

# 3.0.3 Process Flow for Admission Service



* Client invokes the required service.
* AdmissionController invokes the method of the AdmissionServiceImpl.
* AdmissionServiceImplperforms the service and returns the data back with the help of AdmissionRepository.
* AdmissionRepository should a MongoRepository that performs CRUD operations as the case study requirements
* To perform **logging**, you are provided with LogingAspect class. Log the success message and failure message using Slf4j and AOP. Configure logging using **Slf4j** annotation with Lombok in LoggingAspect class ,

**Architecture diagram:**



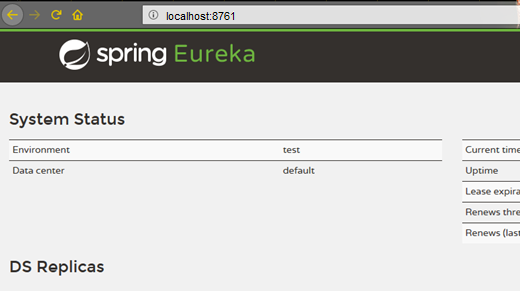
**4.0 Microservices implementation Guidelines**

For Microservices implementation, we will be implementing 4 projects.

* Spring Cloud (Eureka-Server) for service registry and discovery
* Spring Boot for REST service creation for CourseService, AdmissionService and AssociateService
* Spring AuthenticationAuthorization Microservice for creating and validating token
* Spring Cloud gateway for API Gateway

**Project 1:**

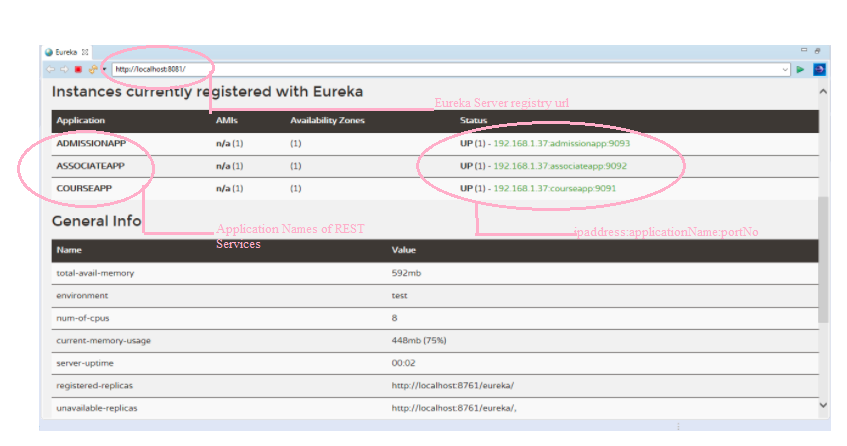
This project should be the Eureka-Server acting as the registry. Run this server in port 8761



**Project 2,3,4:**

* This projects should contain spring boot applications containing all the REST services implemented as per the requirements stated in the case study.
* These services must be registered with Eureka-Server.
* Run the services in port 9091,9092,9093.
* Services should get automatically registered with the Eureka Registry

The below screen depicts the Eureka server home page, after the services are registered,



**Note:**

* **Application names should be given as courseapp ,associateapp,admissionapp as specified in the case study.**
* **Eureka Registry should run only in port 8761**

**Project 3:**

This must be a Spring cloud gateway project that contains the routing implementation to the actual services running in port 9091,9092 and 9093. Run this in port 7777.

**GateWay**

Request

Router

(port :7777)

SecurityService

(9098)

Registry

(port: 8761)

Services

(port :9091,9092 and 9093)

### RestServices EurekaServer

# 5.0 Overall Design Constraints

1. **Do not change the property name given in the application.properties files, you can change the value and you can include additional property if needed.**
2. **In the pom.xml you are provided with all the dependencies needed for developing the application. Do not change or add new dependencies.**
3. Adhere to the design specifications mentioned in the case study.
4. Do not change or delete the class/method names or return types which are provided to you as a part of the base code skeleton.
5. Developer will create 4 applications:
   * Spring Boot (**Name of the application Should be CourseService,AssociateService,AdmissionService,AuthenticationAuthorizationService**)
   * Spring Cloud (Eureka-Server, spring-cloud-starter-eureka-server) (**Name of the application Should be cms-registry-server**)
   * Spring Cloud API GateWay(spring-cloud-starter-gateway) **(Name of the application Should be Gateway**)

1. CORS related issues need to be handled at the application level.
2. Services can be tested using external tools like postman.
3. Eureka Registry should run only in port 8761. You can change the port number of the applications (spring rest) or the Gateway(Router) if required
4. **Ensure before submitting the project the various services are running in the below port**

**Application Name – Port No**

AdmissionService - 9093

AssociateService - 9092

AuthenticationAuthorizationService - 9098

CourseService - 9091

cms-registry-server – 8761

Gateway - 7777