System Requirements Specification Index

**For**

Professional Consultancy Services

### Version 1.0

**IIHT Pvt. Ltd.**

[**fullstack@iiht.com**](mailto:fullstack@iiht.com)

Table of Contents

[Version 1.0 1](#_bookmark0)

[Professional Consultancy Services 2](#_bookmark1)

1. [PROJECT ABSTRACT 2](#_bookmark2)
2. [CONSTRAINTS 3](#_bookmark3)

[Common Constraints 3](#_bookmark4)

1. [System Requirements 3](#_bookmark5)
2. [FAULT TOLERANCE 5](#_bookmark6)
3. [MICROSERVICES COMMUNICATION 6](#_bookmark7)
4. [Rest Endpoints 6](#_bookmark8)
5. [Execution Steps to Follow 9](#_bookmark9)

# Professional Consultancy Services

**System Requirements Specification**

## PROJECT ABSTRACT

Professional Consultancy Services (PCS) is a business consultancy that has established itself as a renowned service provider of a wide range of business services to its clients. PCS is planning to provide business- and employment-oriented service through a skill mapping application that operates via online recruiting website. This application is having spring boot microservices (employee, skills and certificates) which are using different databases and communicating to each other.

**Following is the requirement specifications**:

|  |  |
| --- | --- |
|  | Professional Consultancy Services |
|  |  |
| Microservices |  |
| 1 | Employee-service |
| 2 | Skills-service |
| 3 | Certificates-service |
|  |  |
| Employee  Microservice |  |
|  |  |
| 1 | Register Employee |
| 2 | Get all the skills for an employee |
| 3 | Get all the certificates for an employee |

|  |  |
| --- | --- |
|  |  |
| Skills Microservice |  |
| 1 | Create a skill for an employee |
| 2 | Get all skills for an employee |
| 3 | Get all certificates generated for the given skill |
|  |  |
| Certificates Microservice |  |
| 1 | Generate a certificate for an employee |
| 2 | Get all certificates for an employee |
| 3 | Get all certificates generated for the given skill |
|  |  |
|  |  |
|  |  |

## CONSTRAINTS

### Common Constraints

* Do not change, add, remove any existing methods in service layer.
* In Repository interfaces, custom methods can be added as per requirements.
* All RestEndpoint methods and Exception Handlers must return data wrapped in

**ResponseEntity**

# System Requirements

* 1. **EUREKA-NAMING-SERVER**

This is a discovery server for all the registered microservices. Following implementations are expected to be done:

1. Configure the Eureka server to run on port: 8761.
2. Configure the Eureka server to deregister itself as Eureka client.
3. Add appropriate annotation to Enable this module to run as Eureka Server.

**You can launch the admin panel of Eureka server in browser preview option.**

* 1. **API-GATEWAY**

This microservice is an api gateway to all the microservices. All the microservices can be accessed by using this common gateway. Following implementations are expected to be done:

1. Configure API Gateway to run on port: 8765.
2. Implement the routes and logging in this api-gateway.
   1. **EMPLOYEE-SERVICE**

The employee microservice is used to perform all the operations related to the employee. In this microservice, you have to write the logic for EmployeeServiceImpl.java and EmployeeRestController.java classes. Following implementations are expected to be done:

1. Configure this service to run on port: 8001.
2. You are required to configure 2 feign proxy to fetch (Get Certificate by Employee ID and get Skills by Employee Id)
   1. **SKILLS-SERVICE**

The skills microservice is used to perform all the operations related to employee skills. In this microservice, you have to write the logic for SkillsServiceImpl.java and SkillsRestController.java classes. Following implementations are expected to be done:

1. Configure to run this module on port: 8090
2. You are required to configure feign proxy to fetch : Certificate by Skill name
3. **Running multiple instance of this module**
   * **Run 2 more instance of Skill-Service on port 8091 and 8092, to that feign client would automatically uses the multiple running instances.**
   * **While launching the additional instance of skill-service use following option:** 
     + **-Dserver.port=<port number>**
   1. **CERTIFICATES-SERVICE**

The certificates microservice is used to perform all the operations related to employee certificates. In this microservice, you have to write the logic for CertificatesServiceImpl.java and CertificatesRestController.java classes. Following implementations are expected to be done:

1. Configure to run this module on port: 9001
   1. **Kindly follow the sequence and run your commands through all the folders separately.**
   2. **To build your project use command:**

**mvn clean package -Dmaven.test.skip**

* 1. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

**java -jar <jar-name>-0.0.1-SNAPSHOT.jar**

## FAULT TOLERANCE

Configure your skill microservice DB operation for Hystrix circuit breaker and create fallback function in case database is not connected. Details as below:

1. Add the required Hystrix dependency in pom.xml file of skill microservice
2. Configure Hystrix dashboard and Circuit Breaker for this microservice
3. Create a fallback method for findAll() service method which should return an empty collection if connection with database is not available.

# MICROSERVICES COMMUNICATION

Communication among the microservices needs to be achieved by using FeignClient. Feign configuration class is created in the project, but you are required to implement the feign client method. You can check in the proxy package of the microservice.

# You are required to configure 2 feign proxy to fetch (Get Certificate by Employee ID and get Skills by Employee Id) (Employee-Service)

# You are required to configure feign proxy to fetch: Certificate by Skill name (Skill-Service)

## REST ENDPOINTS

Rest Endpoints to be exposed in the controller along with method details for the same to be created

* 1. **EmployeeRestController**

|  |  |  |
| --- | --- | --- |
| **URL**  **Exposed** | | **Purpose** |
| 1. /api/employees | | Creates a new employee |
| Http Method | POST |
| Parameter | - |
| Return | EmployeeDto |
| 2. /api/employees/skills/{id} | | Fetches skills of an employee by Employee id |
| Http Method | GET |
| Parameter | Id |
| Return | List<SkillsDto> |

|  |  |  |  |
| --- | --- | --- | --- |
| 3. /api/employees/certificates/{id} | | Fetches all certificates of an employee  by id |  |
| Http Method | GET |  | |
| Parameter | Id |
| Return | List<CertificatesDto> |

* 1. **SkillsRestController**

**Purpose**

**URL**

**Exposed**

|  |  |  |
| --- | --- | --- |
| 1. /api/skills | | Creates a new skill |
| Http Method | POST |
| Parameter | - |
| Return | SkillsDto |
| 2. /api/skills/by/{employeeid} | | Fetches all skills of Employee by employee id |
| Http Method | GET |
| Parameter | {employeeId} |

|  |  |  |
| --- | --- | --- |
| Return | List<SkillsDto> |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 3. /api/skills/certificates-by-skill-name/{name} | | Fetches all certificates of by Skill Name |  |
| Http Method | GET |  | |
| Parameter | {name} |
| Return | List<CertificatesDto> |

* 1. **CertificateRestController**

|  |  |  |
| --- | --- | --- |
| **URL**  **Exposed** | | **Purpose** |
|  | 1. /api/certificates |  |

|  |  |  |
| --- | --- | --- |
| Http Method | POST | Creates a new certificate |
| Parameter | - |
| Return | CertificatesDto |
| 2. /api/certificates/employee/{employeeid} | | Fetches all certificates of employee by Employee ID |
| Http Method | GET |
| Parameter 1 | {employeeid} |
| Return | List<CertificatesDto> |
| 3. /api/certificates/skills/{skillName} | | Fetches all certificates by  Skill Name |
| Http Method | GET |
| Parameter | {skillname} |
| Return | List<CertificatesDto> |

## EXECUTION STEPS TO FOLLOW

* 1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
  2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal**
  3. **Kindly follow the sequence and run your commands through all the folders separately.**
  4. **To build your project use command:**

**mvn clean package -Dmaven.test.skip**

* 1. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

**java -jar <jar-name>-0.0.1-SNAPSHOT.jar**

* 1. **This editor Auto Saves the code**
  2. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
  3. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
  4. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**
  5. **To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
  6. **Default credentials for MySQL:**
     1. **Username: root**
     2. **Password: pass@word1**

1. **To login to mysql instance: Open new terminal and use following command:**
   1. **sudo systemctl enable mysql**
   2. **sudo systemctl start mysql**
   3. **mysql -u root -p**

**The last command will ask for password which is ‘pass@word1’**

1. **Mandatory: Before final submission run the following command for each of the module: mvn test**
2. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality**