|  |
| --- |
|  |

|  |
| --- |
| **Mentor On Demand** |
| Case Study |
|  |
| This document covers Software Requirements of Mentor On Demand, along with list of Technologies to be used to develop this Software System, and also includes some details on the Architecture |
|  |
|  |
|  |
|  |

Table of Contents

[1. Business Requirement(Mentor On Demand) 2](#_Toc22425676)

[1.1. Retail Use cases 2](#_Toc22425677)

[1.1.1. User(Retail) Use Cases. 2](#_Toc22425678)

[1.1.2. Mentor Use Cases 3](#_Toc22425679)

[1.1.3. Admin Use Cases(optional) 4](#_Toc22425680)

[2. Design Inputs 4](#_Toc22425681)

[3. UI/UX – Angular Components and Templates – Front End 4](#_Toc22425682)

[4. Model Classes – Mid Tier 6](#_Toc22425683)

[5. Entity Classes – Mid Tier 8](#_Toc22425684)

[6. Microservices & Architecture Diagram – Mid Tier 13](#_Toc22425685)

[7. Development of individual Microservices 14](#_Toc22425686)

[8. Database Tables 15](#_Toc22425687)

[9. Microservices Integration and Security 16](#_Toc22425688)

[10. Spring Microservices Tools to be used 16](#_Toc22425689)

[11. JWT Authentication 17](#_Toc22425690)

[12. Architecture/Design(with Netflix OSS Tools) 17](#_Toc22425691)

[13. Training Progress Scheduler 17](#_Toc22425692)

[14. DevOps Activity 18](#_Toc22425693)

[Diagram 18](#_Toc22425694)

[15. Jenkins CI/CD 19](#_Toc22425695)

[16. Configure Jenkins and Docker for the Project 19](#_Toc22425696)

[17. Perform CI/CD 19](#_Toc22425697)

[CI/CD Diagram 20](#_Toc22425698)

[18. Full Stack Technologies 20](#_Toc22425699)

[19. Technical Spec – Solution Development Environment 20](#_Toc22425700)

[19.1. Front End Layer 20](#_Toc22425701)

[19.2. Middle Tier Layer 21](#_Toc22425702)

[19.3. Database & Integration Layer 21](#_Toc22425703)

[19.4. Ancillary Layer 21](#_Toc22425704)

[19.5. Security 21](#_Toc22425705)

[19.6. Deployment & Infrastructure 21](#_Toc22425706)

[19.7. Editors 21](#_Toc22425707)

[20. Assessment Deliverables 22](#_Toc22425708)

[21. Important Instructions 22](#_Toc22425709)

# Business Requirement(Mentor On Demand)

Build a software system which lets users search for a Mentor for a specific Technical Course, User can search, select, propose & finalize a Mentor of his Choice, after which Training gets started.

## Retail Use cases

Below are the different roles, which need to be supported by above Software System.

#1. User

#2. Mentor

#3. Admin

Below are the Use Cases which need to be supported by each of above Roles

### User(Retail) Use Cases.

User should be able to Signup. Email ID can be used as username, which need to be confirmed by User to complete Signup process.

User should be able to Signin.

User can search for Mentors for a specific 2hrs Time slot (optional) and Technology(mandatory), even without signing in.

Search results need to list Mentors having expertise in specific Technology, who are available in that specific time slot(Mentor's Calendar can be checked to display search results)

Each search result has to display Name of Mentor, years of experience, no. of trainings delivered in total and in specific technology, Fee charged (including commission).

After viewing Mentor's Profile/History/Rating, User can select a specific Trainer from above Search Results and should be possible to send Proposal, to the Mentor.

Once the User Proposes, Propose request is received by a specific Mentor. A Mentor can Confirm or Reject a Proposal. Proposal Confirmation or Rejection need to be sent to User.

User can Finalize a Confirmed Proposal response(from Mentor), after which user need to make payment, for Training Course.(Integration with any Payment Gateway is optional)

Payment is collected in advance from User. But Mentor will be paid in 4 slots, as the Training Progress reaches 25%, 50%, 75% and 100%

Should be possible to Rate any Mentor, of the Trainings which are in Progress

User should be able to view list of Current Trainings in Progress.

Trainings in any of below states are considered to be in Progress.

1. Proposed.
2. Confirmed Proposals
3. Trainings started, and not completed

User should be able to view List of Trainings already availed and Completed.

### Mentor Use Cases

Mentor need to SignUp to provide Mentoring service through the Portal

During SignUp process Mentor need to provide timezone/working timings, list of technologies, facilities provided –reading material, examples/cloud labs, email/mobnum verification, linkedin URL, mentor profile, number of years of experience. Email id can be used as Mentor’s username.

View History of Mentor - Trainings delivered, ratings, etc...

Payment will be fixed based on Technology/Complexity, etc... and based on Trainers experience, proposal

Trainer will be paid in 4 slots, based on progress

Payment will get accumulated based on Progress, which can be withdrawn by Trainer.(Payment Gateway Integration is optional)

E-Mail Notifications need to be automatically sent to User and Mentor, during appropriate Use Cases.(Optional)

### Admin Use Cases(optional)

Implementation of Admin functionality is optional. Below are the Use Cases

Admin can add/remove List of Technologies

Block or Unblock User or Mentor

Admin should be able to search the Payments made, and display reports

Admin should be able to edit parameters such as Payment Commission percentage(which will be deducted from the payments done by User).

Contact details should be confidential, and should not be shared with each other.

# Design Inputs

Next sections in this document provides inputs on designing the solution for above requirements.

Design inputs provided in this document are just for your reference purpose, Associates can make changes or additions to the Design, based on their analysis.

# UI/UX – Angular Components and Templates – Front End

In this Phase you will develop, UI/UX of the SPA application using HTML5, Angular, Javascript and Jquery where required.

This Phase includes development of

1. Angular components with Templates
2. Angular Routing
3. Angular Services, to invoke Spring Boot REST End points.

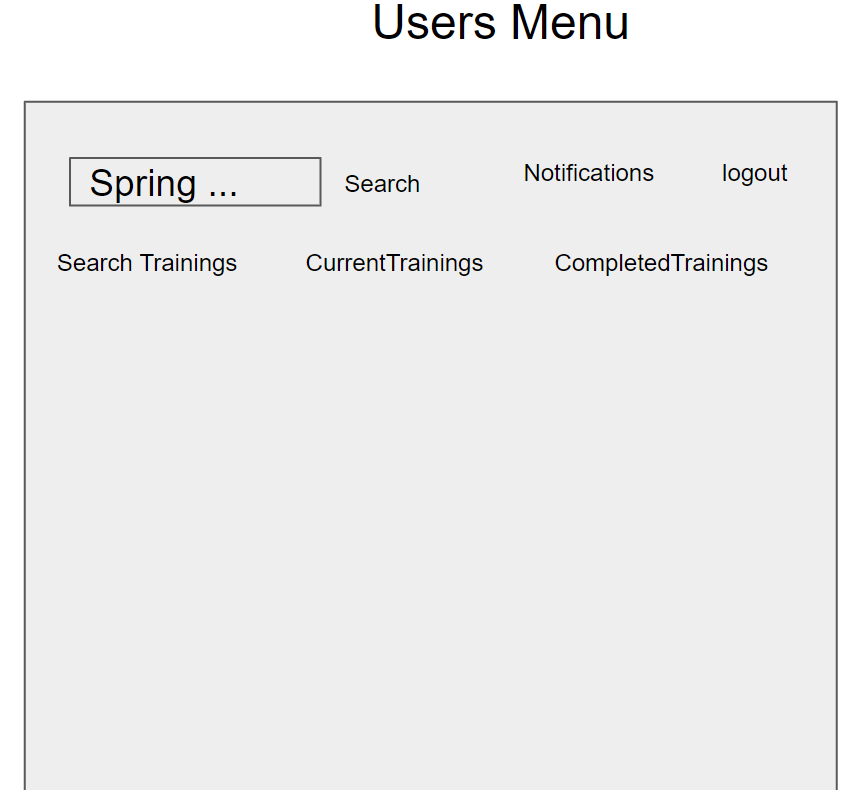
Below can be various Angular Components

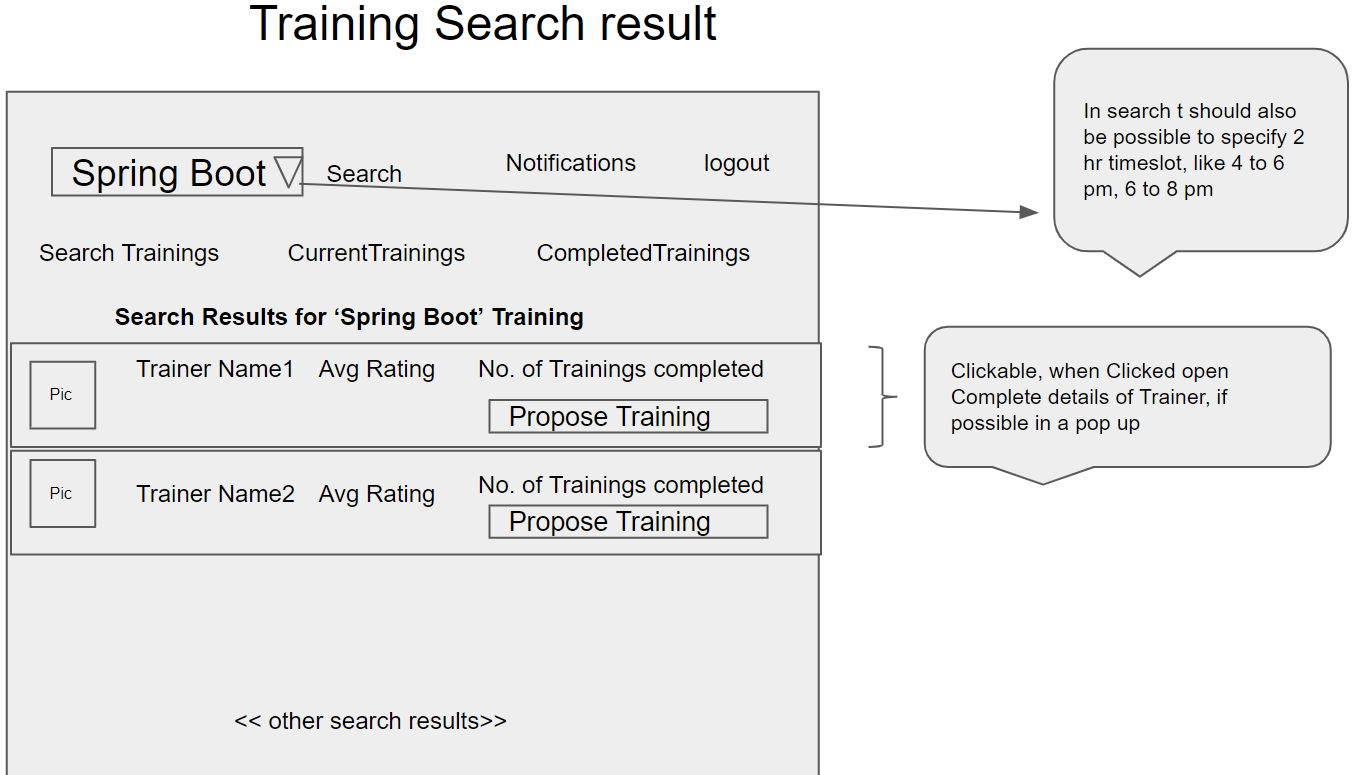
1. MentorSearch – to search for a Mentor for a specific Technology and Time slot(optional), and display search results as a list
2. MentorTrainings – to view Trainings(in Completed or in any other state) related to a specific Mentor
3. MentorCalendar–to view Mentor’s Calendar
4. Training –to display details(like user, Trainer, Current State, etc…) of a specific Training, % completion.
5. MentorProfile – to viewMentor’s profile
6. MentorSkills – which lets Mentor to specify Skills, experience, etc…
7. UserTrainings – to View Trainings(in Completed, Proposed, Approved, In Progress states)
8. UserLogin – Login for both user, mentor
9. UserSignup – Signup for both user, mentor

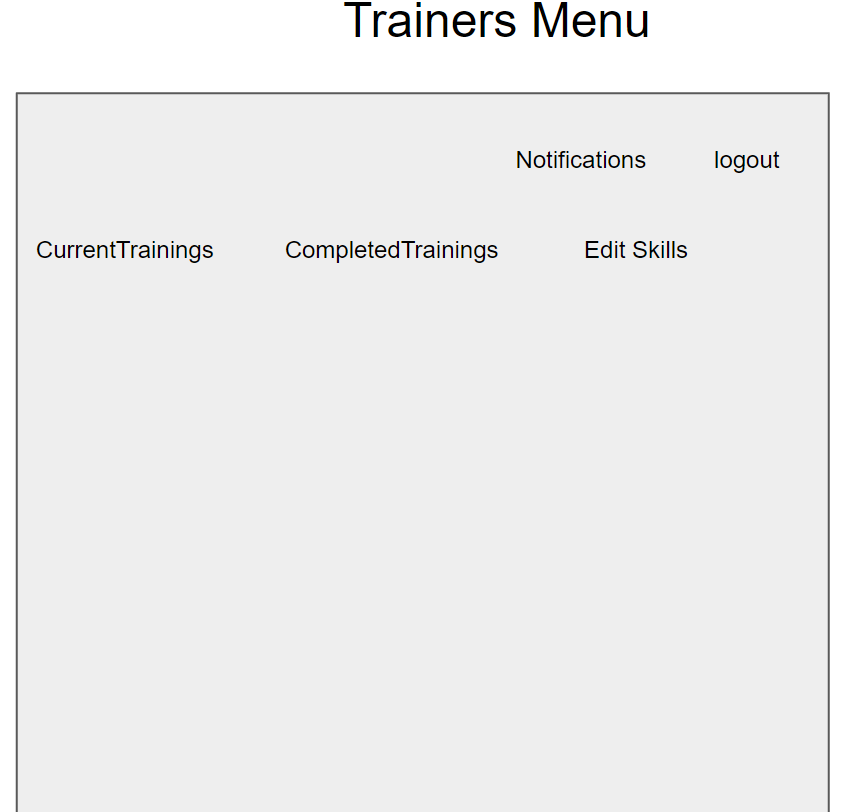
Angular Routing feature is used to display Navigation links on Web Pages. When User clicks a Navigation link, the corresponding Component is displayed.

As known, Angular Services areused to interact with the REST end points, using Observable or Promises.

Below are the Wireframes which need to be made available as part of Angular Components, responsive UI of the application using HTLM5, HTML5 API, CSS3 and Bootstrap/Material.







Below are wireframes

1. Login/Signup for User/Mentor
2. User Landing Page and Navigation Links
3. Mentor Landing Page and Navigation Links
4. User/Trainer Trainings - Completed
5. User Trainings - Progress
6. Training Search Results
7. Trainer Profile
8. Trainer Payments – View History & Withdraw
9. Trainer Trainings - Progress
10. Trainer Edit Skills
11. Admin – Block User/Mentor
12. Admin- Add/Edit Technology and change Fee

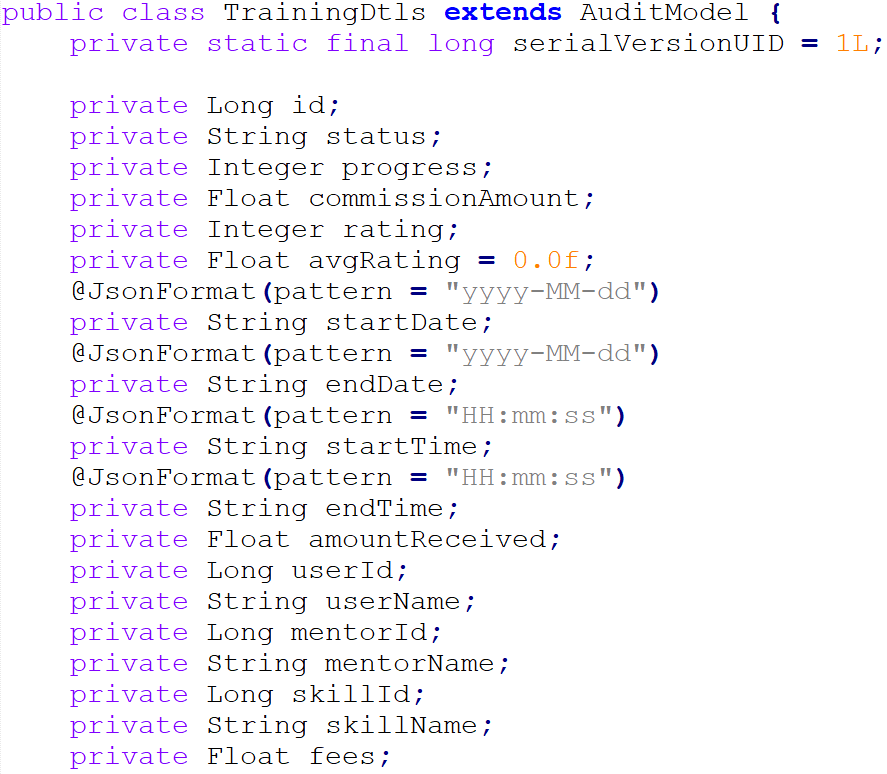
Based on the requirements, Front End need to be divided into multiple components to accommodate above Wireframes. Angular Routing can be used to create navigation Links. For Authentication, store JWT token in Local or Session Storage(on Client). REST APIs are invoked from the corresponding Services,

As known JWT token is generated on the Server side and received by Client on successful authentication. Angular’s HttpInterceptor can be used to automatically send JWT Token thru Header of every HttpRequest to REST API end points.

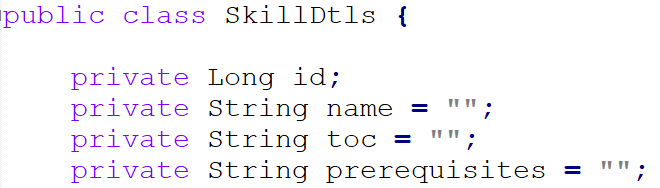
# Model Classes – Mid Tier

This Phase comprises developing Mid Tier’s Model and Entity classes(in Java) which are required to be used in the next Phases. Below are some sample Model classes

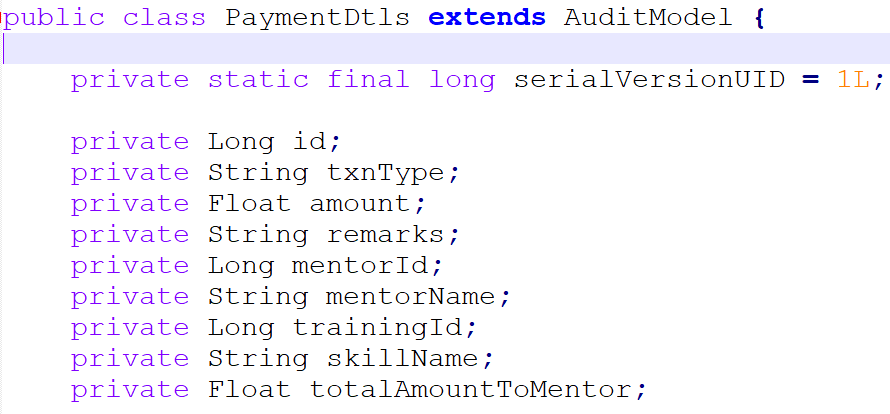
Training Details class



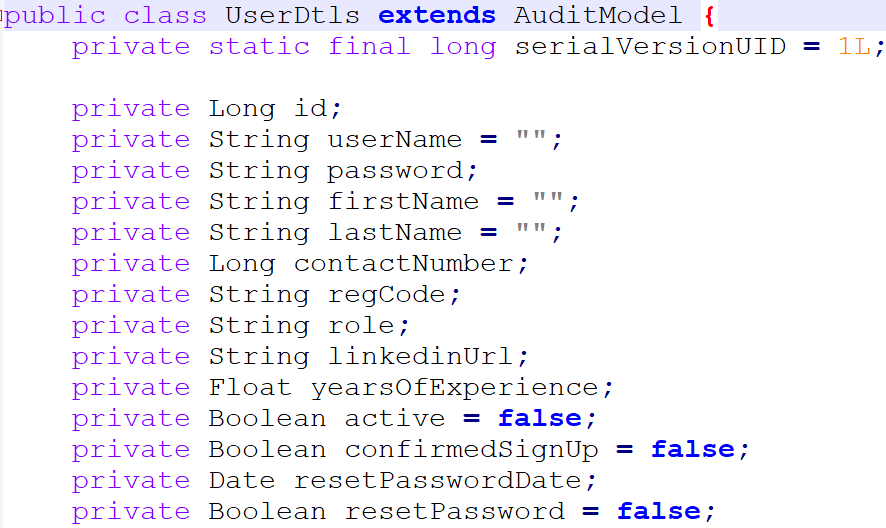
Skills Classes



Payment Details



User Details Class



Skills Classes

# Entity Classes – Mid Tier

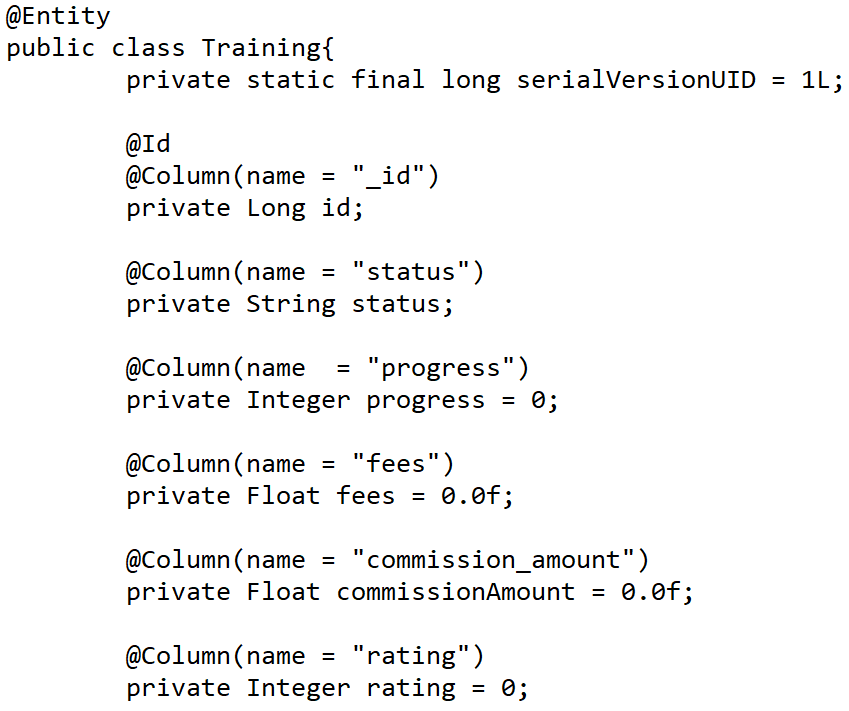
Entity classes are the ones which are mapped with underlying DB table(DB tables for reference are provided in next Sections of this document). Entity classes also includes corresponding mappings between them, such as below(It is not mandatory to use below Mappings)

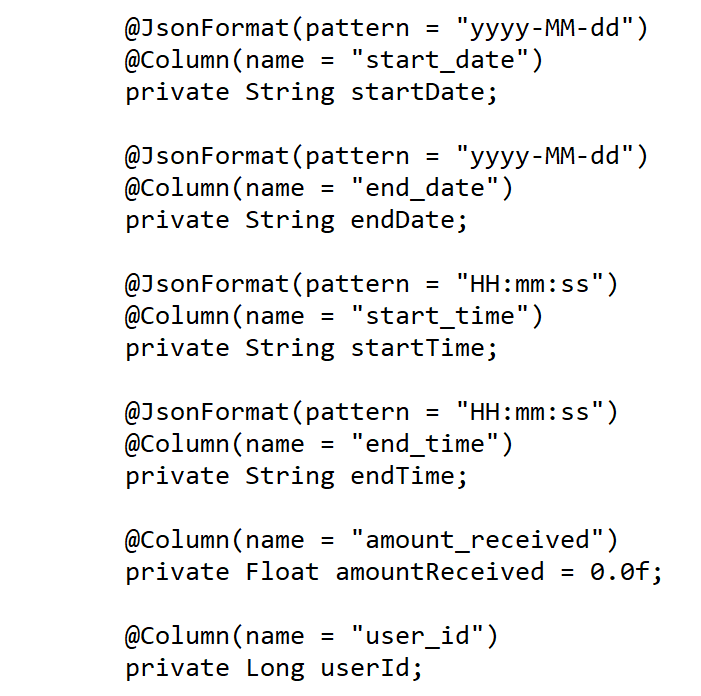
1. One to One Mapping
2. One to Many Mapping
3. Many to One Mapping
4. Many to Many Mapping

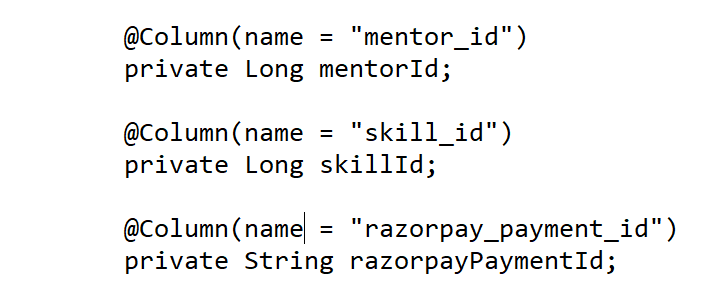
Entity classes can also include validation related details, such as @NotNull

Below are some sample Entity Classes. Similarly identify all the Entity classes based on the requirements.

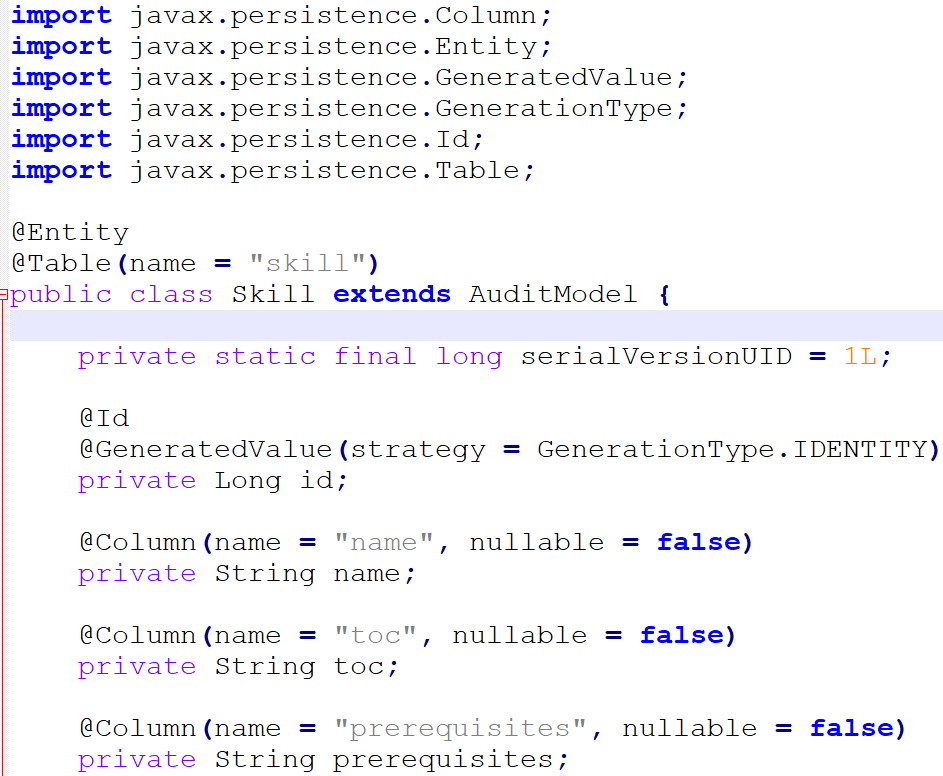
Training Entity class



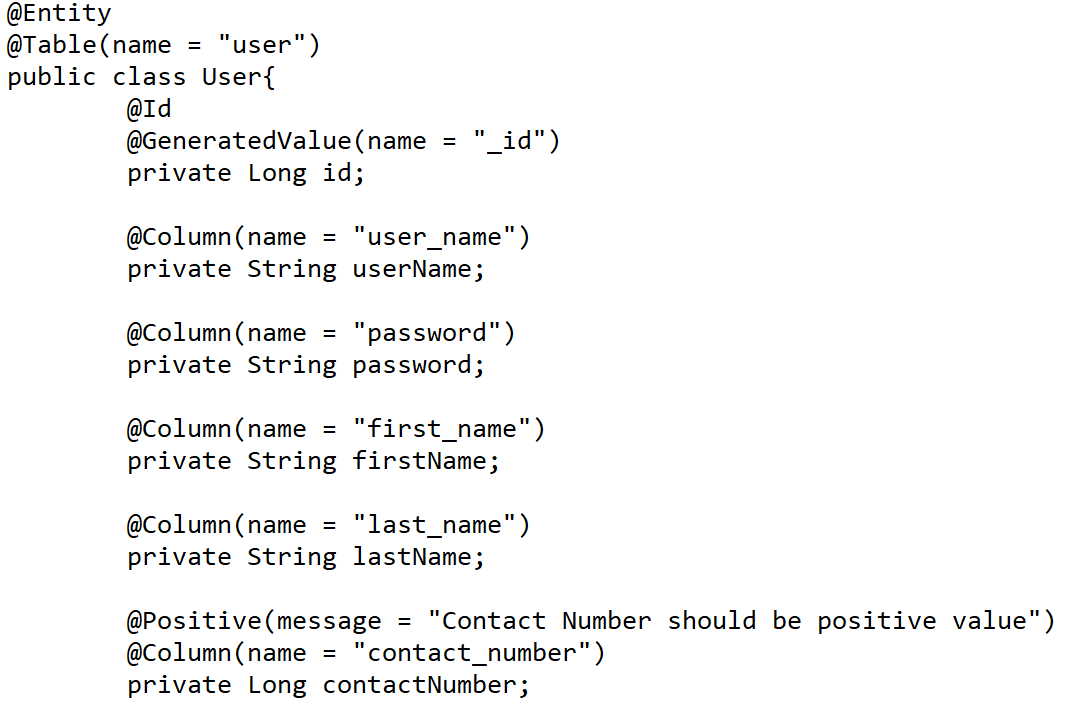


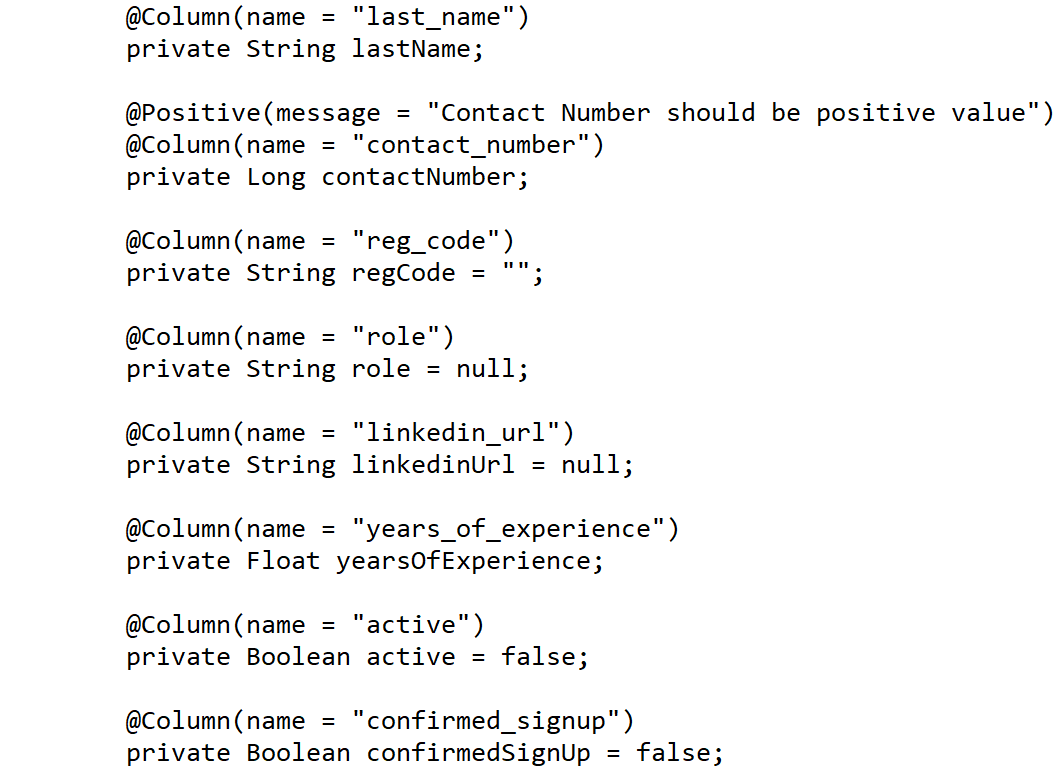


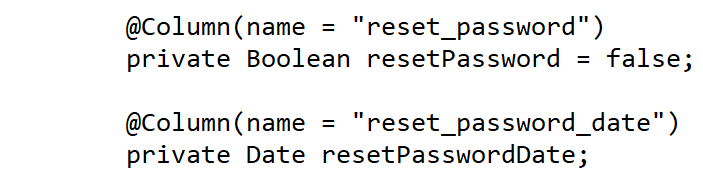
Skill Entity class



User Entity class







# Microservices &Architecture Diagram – Mid Tier

This project need to be developed using Microservice Architecture. Divide the Mid Tier functionality into multiple Microservices



Architecture of a Single Microservice with REST Controller, Service, Model & Entity Classes and Repository classes



# Development of individual Microservices

This specific Phase is to design/develop individual Microservices. Analyze the requirement and divide Mid Tier functionality into multiple Microservices. Based on the Mentor on Demand requirements, below can be possible Microservices

1. User Microservice
2. Training Microservice
3. Technology Microservice
4. Search Microservice
5. Security Microservice
6. Payment Microservice

Each of above Microservicesneeds to comprise below functionality, which need to be developed

1. REST Controllers
2. Service Layer
3. Entity & Model classes, including appropriate relationship(like One-One, Many-One, etc…) between Entity Classes. (Entity and Model classes have been developed in the Previous Phase)
4. In case specific Entity or Model classes are required across multiple Microservices, it is recommended to maintain separate copy of Entity or Model classes for each Microservice, for flexibility reasons.
5. Microservice interaction with corresponding DB tables or Databases it owns.
6. It is possible that one Microservice need to interact with other Microservice(using RestTemplate or FeignClient)
7. Repository class which implements JPA or CrudRepository, if RDBMS is used
8. Usage of Custom Queries using @Query where ever custom functionality required
9. Feign Client can be used to invoke one Microservice, from another Microservice
10. Send Email Notifications to Users/Mentors in Use Cases wherever appropriate.
11. Use Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice
12. Unit Testing code can be developed using JUnit, Mockito, and perform Unit Testing

# Database Tables

Below are list of Database Tables, which can be used by corresponding Microservice. Though, ideally each Microservice need to use separate database, it should be fine to place all below DB Tables in a single database

|  |  |
| --- | --- |
| Table Name | Purpose |
| Users | Stores list of User, Mentor, Admin – usernames and login credentials |
| Technologies | Store list of all Technologies, whose Trainings are supported by Portal |
| Trainings | Stores details of each Training in progress or completed |
| MentorSkills | Stores skills of Mentor |
| MentorCalendar | Stores Calendar slots of a Trainer |
| Payments | Stores payments released to Trainer, based on Training Progress |

# Microservices Integration and Security

Assuming that you are done with developing individual Microservices in previous Phase, current Phase includes creating and integrating Zuul gateway, Eureka Server and Eureka client in each Microservice. This is shown in architecture Diagram, in next section.

Zuul Gateway(create a Zuul based Project using Spring Initilaizer or STS IDE), add required annotation(s).

Authentication and JWT Token validation can be performed in Zuul’s Pre Filter.

Add below details to yml or property file of Zuul Gateway

1. add route configurations
2. port number & url of eureka Server

Eureka Server(create a Eureka Discovery Server using Spring Initializer or STS IDE), add required annotation & port number in yaml configuration file

Add Eureka Discovery Client to all the Microservice

Now open Eureka Server Dashboard by opening and crosscheck if all Microservices are registered in the dashboard

Now start sending the requests to Zuul Gateway which further routes to a specific Microservice based on the url pattern

Develop code for Unit Testing

PostMan, to test REST end points

# Spring Microservices Tools to be used

As already specified under Full Stack Technologies Microservice Architecture need to be followed. Ensure that the Application is divided into multiple Microservices, along with database/tables each Microservice Manages. Below Spring Microservices Tools need to be used

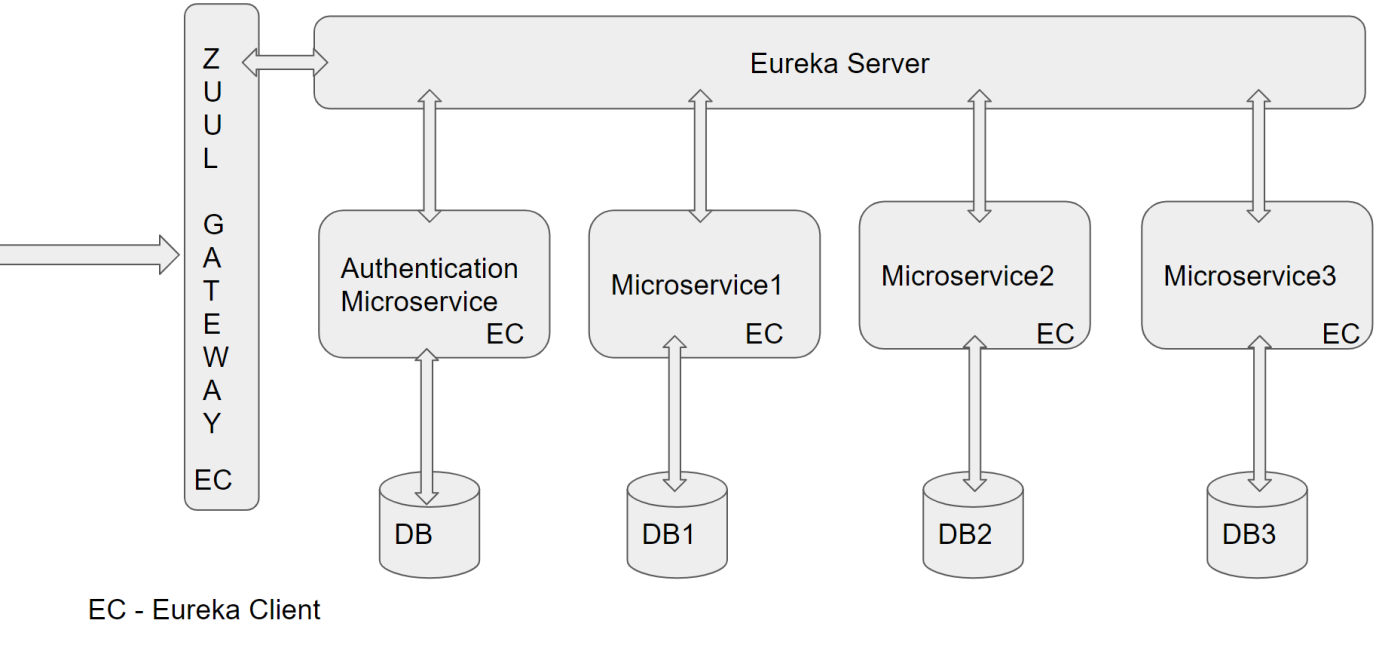
* Zuul API Gateway
* Eureka Service Registry & Discovery
* Ribbon Client side Load Balancer(optional)
* Feign Client(or RestTemplate) – for interaction between Microservices
* Hystrix Circuit Breaker & Fault Tolerant Tool(optional)

# JWT Authentication

Create additional Microservice which takes care of authentication and role activities, and JWT Token validation. Spring Security need to be used for Authentication. On successful authentication or token validation the actual request need to be forwarded to the corresponding Microservice. Invoke authentication REST endpoints from Zuul Gateway. Use PreFilter to perform JWT Token validation by invoking REST endpoint of this Microservice.

Instead of JWT, any other security protocol such as OAuth2 can be used. Authentication data can be stored in MySQL DB or LDAP or any other data source.

# Architecture/Design(with Netflix OSS Tools)



# Training Progress Scheduler

A Scheduler Program may be required which runs periodically (For eg. Daily). This application can be developed using Spring’s @Scheduled annotation. This application need to check current date and Training start date and update Current Progress percentage in Trainings DB Table, and whenever progress is 25%, 50% or 75% corresponding amount need to be added to Mentor’s Wallet.

# DevOps Activity

This phase includes performing below Activities

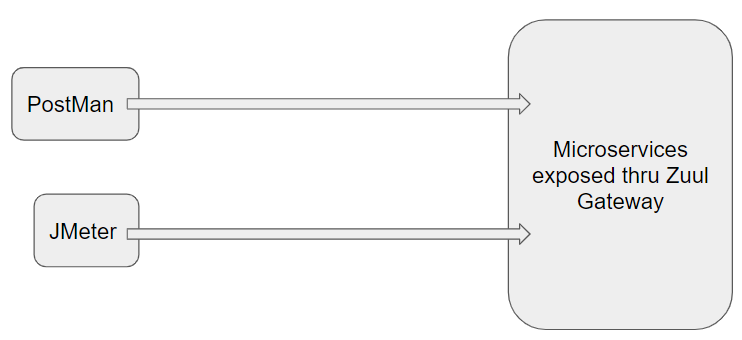
**JMeter:** As already known JMeter is used to perform Performance or Load Testing. Create a JMeter Test Case, which invokes a REST End point, with multiple threads.

Check in jmx file and share the report generated after Performance Testing. Repeat this for atleast two REST end points. If required performance benchmark is not matching consider using appropriate algorithms or Collections to tune the performance.

**Dockerization:** Dockerize atleast Front End or any one Microservice of Mid Tier. Provide Dockerfile and the docker commands used to create image and run Container. Share Screen shots of running Docker.

**Code Coverage:** Code coverage is a Quality Metric to check if sufficient number of Test Cases are created. EclEmma tool can be used as Code Coverage Tool. Code Coverage can be performed on any one Microservice.Ensure that Code Coverage need to be atleast 80%

# Diagram



# Jenkins CI/CD

**Jenkins CI/CD:** As already known Jenkins is popular tool to perform CI/CD. When the code is pushed to GIT, build need to be automatically created and deployed. If possible create a Docker image and run the Container on Docker Host

This Phase also includes completion of Integration of Front end with Mid Tier.

**Deployment on Cloud(optional):** Any of the Microservices or Front End can be deployed on any Cloud(AWS, Azure, etc…) of your choice.

# Configure Jenkins and Docker for the Project

* Import the project (as discussed above) in Spring Tool Suite and configure it locally to run it as Spring Boot App.
* You may need to configure MySQL credentials and database name.
* Execute the project locally and access the app at http://localhost:portnumber
* Once, it is working fine in local development environment; Configure CI/CD in Jenkins, along with Dockerization
* Push the app source in internal GIT server. Internal GIT is 172.18.2.18 which can be accessed from IIHT VMs only.
* Configure Jenkins locally to pull the source from internal GIT repository
* Jenkins should build the project and create the deployable (war/jar). It should run the unit tests created in ”Maven, GIT, Junit, Tomcat Micro Layer for the Project”
* From Jenkins, invoke Docker commands to perform, below
* Creation of Docker Image(docker build . )
* Create and run Docker Container(docker run <image\_id>)

# Perform CI/CD

1. Make few changes in the project (source code)

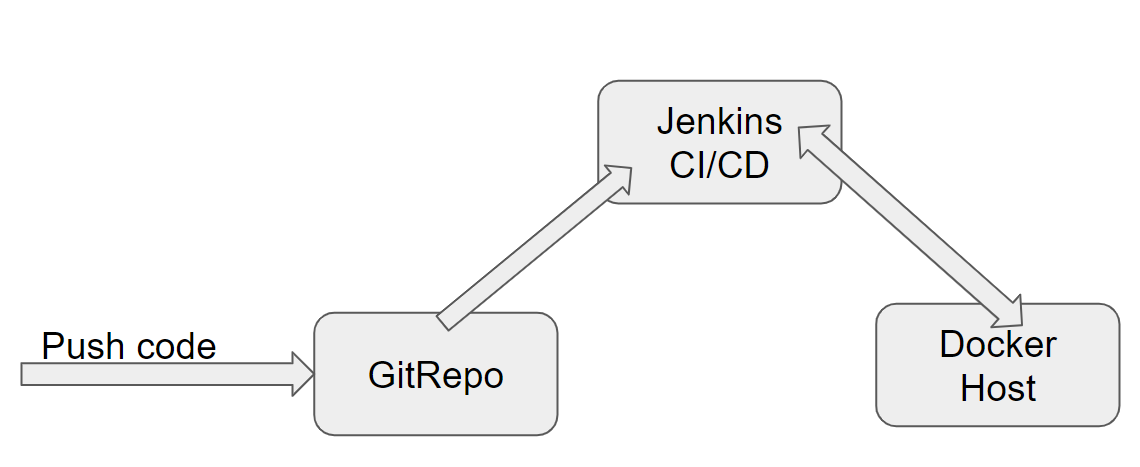
2. Make it sure that project is running locally in development environment without errors.

3. If it running locally without errors, push the changes to the internal GIT repository which was connected

4. If Project was Setup properly, Jenkins will automatically pull the code updates from internal GIT repo and build and deploy the project with updated code.

5. Now, when you visit http://localhost; you should see the changes in the browser window

# CI/CD Diagram



# Full Stack Technologies

The technologies included in Full Stack are not limited to following but may consist of:

* UI Layer (HTML5, CSS3, Bootstrap 4, JavaScript, Jquery, Angular 4/6)
* Middleware Restful API (Spring Boot Restful& MicroServices, JAX-RS, Spring MVC)
* Database Persistence ( Hibernate)
* Database layer (MySQL)
* Ancillary skills (GIT, Jenkins(CI/CD), Docker, Maven) etc.

To complete this case study, you should be comfortable with basic single page web application concepts including REST and CRUD. You may use angular-cli to create your template project. All web pages need to be responsive.

Ref1: https://cli.angular.io/

Ref2: <https://github.com/angular/angular-cli>

# Technical Spec – Solution Development Environment

## Front End Layer

|  |  |
| --- | --- |
| **Framework(s)/SDK/Libraries** | **Version** |
| Angular with TypeScript | 4/6 |
| Bootstrap | 3.0 or above |
| CSS | 3 |
| HTML | 5 |
| JavaScript | 1.8 or above |
| JQuery | 1.3 |

## Middle Tier Layer

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Java Stack | Spring Boot | 2.x |
| Spring MVC | 4.0 or above |
| JDK | 1.8 |
| Maven | 3.x or above |

## Database & Integration Layer

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Java Stack | JPARepository/CrudRepository or Hibernate | 4.0 or above |
| MySQL | 5.7.19 |

## Ancillary Layer

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Source Code Management Tool | GIT | 2.14.2 |
| Build Tool/JAVA Stack | Maven | 3.x |
| Testing Tool/JAVA Stack | JUnit/Mockito | 4.x |
| Testing Tool/JAVA Stack | Spring Test | 4.x |
| Controllers can be tested using Postman Tool | | |

## Security

|  |  |
| --- | --- |
| **Name** | **Version** |
| Spring Boot Security |  |
| JWT |  |

## Deployment & Infrastructure

|  |  |  |
| --- | --- | --- |
| **Technology** | **Framework(s)/SDK/Libraries** | **Version** |
| Docker | - |  |
| Apache Tomcat | - |  |
| Jenkins(CI/CD) | - |  |
| Node | - |  |

## Editors

|  |  |
| --- | --- |
| **Name** | **Version** |
| STS(Spring Tool Suite) |  |
| Visual Studio Code |  |

Agile/Scrum Software development Model can be used

# 

# Assessment Deliverables

Below deliverables need to be checked in(to internal GIT or github)

1. FrontEnd Source code
2. Mid TierSource code of all Microservices
3. Screen shots of Usage of Post Man tool to test each End Point of all Microservices
4. Few Steps on how to run the solution.
5. Test code of Angular and Mid Tier need to be included
6. Jmeter’s JMX file to test atleast one REST End point, and Screenshot of report
7. Dockerfile
8. Jenkinsfile or Jenkins UI ScreenShot
9. URL where the Project is deployed

# Important Instructions

1. Consider using below Java features
2. Lambda Expressions
3. Collection Streams
4. Generics
5. Sample Design provided is just for reference, Associates can make changes over it or follow their own Design.
6. Based on your current work, alternate Technologies can be used, for example ReactJS instead of Angular, etc…, however prior approval from the Mentor is required.
7. Please make sure that your code does not have any compilation errors while submitting your case study solution.
8. The final solution should be a zipped code having solution. Solution code will be used to perform Static code evaluation.
9. Implement the code using best design standards/family Design Patterns.
10. Use Internationalization for all the labels and messages in Rest API Development.
11. Do not use System out statements or console.log for logging in Rest API and FrontEnd respectively. Use appropriate logging methods for logging statements/variable/return values.
12. If you are using Spring Restful or Jersey JAX-RS to develop Rest API, then use Maven to build the project and create WAR file.
13. Write web service which takes input and return required details from database.
14. Use JSON format to transfer the results.

For any further queries you can contact fullstack@iiht.com