|  |
| --- |
| **Website administration** |
| Use Case document for Full Stack development |
| This document covers Software Requirements of Website administration, 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 (Website Administration) 3](#_Toc74906881)

[2. Technical Spec – Solution Development Environment 6](#_Toc74906882)

[*Front End Layer* 6](#_Toc74906883)

[*Middle Tier Layer* 6](#_Toc74906884)

[*Database & Integration Layer* 6](#_Toc74906885)

[*Ancillary Layer* 6](#_Toc74906886)

[*Security* 6](#_Toc74906887)

[*Deployment & Infrastructure* 6](#_Toc74906888)

[*Editors* 7](#_Toc74906889)

[Coding Standards and Guidelines 7](#_Toc74906890)

[Designing for Testability 7](#_Toc74906891)

[3. UI Components 8](#_Toc74906892)

[3.1. UI Wireframes 8](#_Toc74906893)

[3.2. UX – Angular Components 9](#_Toc74906894)

[4. Mid-Tier Java Components 10](#_Toc74906895)

[4.1. Architecture Diagram 10](#_Toc74906896)

[4.2. Database Constructs, Model Classes and Entity classes 10](#_Toc74906897)

[4.3. Development of individual Microservices 11](#_Toc74906898)

[4.4. Microservices Integration and Security 12](#_Toc74906899)

[4.5. Spring Microservices Tools to be used 12](#_Toc74906900)

[4.6. JWT Authentication 13](#_Toc74906901)

[5. DevOps Activity 14](#_Toc74906902)

[5.1. Diagram 14](#_Toc74906903)

[5.2. Jenkins CI/CD 14](#_Toc74906904)

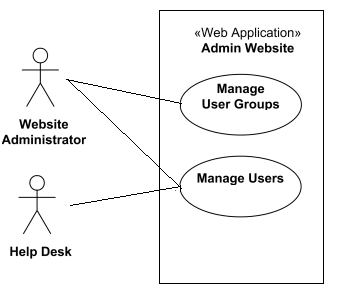
[5.3. Configure Jenkins and Docker for the Project 14](#_Toc74906905)

[5.4. Perform CI/CD 14](#_Toc74906906)

[6. Important Instructions 15](#_Toc74906907)

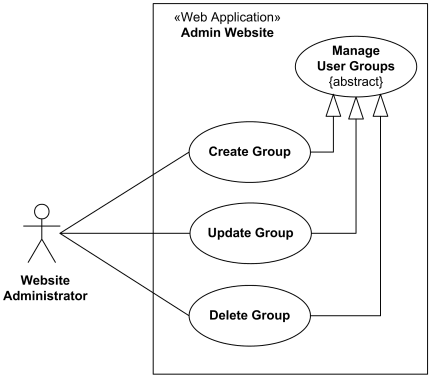
# Business Requirement (Website Administration)

Build a software system which lets website administrator to manage users, user groups, user sessions. Along with administrators, some part of the administrative interfaces should be also available to the Help desk staff, as they need to be able to assist customers having issues while using the customer oriented website.



*Top level use case diagram for the administration website.*

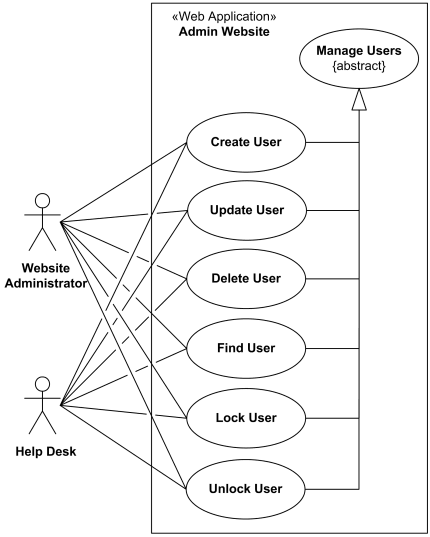
**Manage User Groups** abstract use case is specialized by **Create Group**, **Update Group**, and **Delete Group** use cases. The idea is that website administrator could create different user groups, for example, having different privileges or options, and later some user groups could be modified or even deleted.



*User group management use case diagram for the administration website.*

User management use cases are available both to the **Website Administrator** and to the **Help Desk**. There is standard user CRUD (Create, Retrieve/Find, Update, Delete) functionality set.

Two other use cases, **Lock User** and **Unlock User** are specific to website security. For example, if during some predefined period of time there were several unsuccessful login attempts using wrong user password, user account should be locked for some predefined time to prevent possible brute force password guessing attack. This locking and unlocking is usually done automatically by intrusion detection or website authentication subsystem, but this functionality needs to be available in the manual mode too, just in case. For example, some user might call and ask to lock his or her account.



*User management use case diagram for the administration website.*

# Technical Spec – Solution Development Environment

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

### Front End Layer

|  |  |
| --- | --- |
| Framework(s)/SDK/Libraries | Version |
| Angular | 8 |
| TypeScript compatible with Angular 8 | 3.4 |
| Bootstrap | 4 |
| CSS | 3 |
| HTML | 5 |
| JavaScript | ECMAScript 2019 / June 2019 |
| JQuery | 3.5.1 / (May 4, 2020) |

### Middle Tier Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Java Stack | Spring Boot | 2 or above |
| Spring MVC | 4.3.x |
| JDK | 1.8.0\_211 |
| Maven | 3.6.3 or above |

### Database & Integration Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Java Stack | Hibernate | Hibernate 4.1.9 Final and 3.3.2.GA or above |
| Jersey JAX-RS / Spring Restful | 2.x |
| MySQL | 8 CE (Community Edition) |
| MongoDB | MongoDB | 3.4 |

### Ancillary Layer

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Source Code Management Tool | GIT | 2.25.1 or above |
| Build Tool/JAVA Stack | Maven | 3.6.3 or above |
| Testing Tool/JAVA Stack | JUnit/Mockito | 4.x.x |
| Testing Tool/JAVA Stack | Spring Test | 4.x.x |
| Controllers can be tested using Postman Tool | | |

### Security

|  |  |
| --- | --- |
| Name | Version |
| Spring Boot Security | 5.x |
| JWT |  |

### Deployment & Infrastructure

|  |  |  |
| --- | --- | --- |
| Technology | Framework(s)/SDK/Libraries | Version |
| Docker | - |  |
| Apache Tomcat | - | 8.5.4.3 |
| Jenkins(CI/CD) | - | 2.204.5 and above  2.54 is Preferred |
| Node | - | 10.16.0 |

### Editors

|  |  |
| --- | --- |
| Name | Version |
| STS(Spring Tool Suite) | 4.6.0 |
| Visual Studio Code | 1.4.2 |
| Eclipse Jee | 4.11.0 |

### Coding Standards and Guidelines

Throughout the implementation of use case, standard naming and coding guidelines needs to be followed for GUI and Java in order to

* produce easily readable,
* consistent code with proper naming conventions
* achieve robustness,
* maintainable code through proper exceptional handling and logging,
* assist software portability,

Refer these links for more details around coding standards.

* <https://docs.google.com/document/d/1XXMvReO8-Awi1EZXAXS4PzDzdNvV6pGcuaF4Q9821Es/pub>
* <https://itnext.io/clean-code-checklist-in-angular-%EF%B8%8F-10d4db877f74>
* <https://docs.oracle.com/cd/E12517_01/back_office/pdf/141/html/pos_impg2/developmentstandards.htm>
* <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

### Designing for Testability

Emphasis should be given to Test Driven Development (TDD) and wherever needed, Integration test cases should also be written.

## 3. UI Components

### 3.1. UI Wireframes

In this Phase you will develop, responsive UI of the application using HTLM5, HTML5 API, CSS3 and Bootstrap/Material. (This Phase does not include development of Angular Components)

#### Below are wireframes which need to be developed in this Phase

1. Login for Website administrator and Helpdesk
2. Website administrator Landing Page and Navigation Links
3. Helpdesk Landing Page and Navigation Links
4. Manage user group pages
5. Manage user pages

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. 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 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>

#### Deliverables of this Phase

1. Wireframes covering menu options and major webpages, in any format
2. Html templates
3. Screen shots of sample Html templates

### 3.2. **UX – Angular Components**

In this Phase you will develop, UX of the SPA application using Angular and Typescript. You will use Angular, Javascript and Jquery where required.

#### This Phase includes development of

1. Angular components using Templates(Templates are created in Phase1)
2. Angular Routing
3. Angular Services, to invoke REST End points.

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 is used to interact with the REST end points, using Observable or Promises.

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. 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.

###### Deliverables of this Phase

1. UI screens ready to be integrated with backend REST API endpoints.
2. Demo-able screens with mock data.

## Mid-Tier Java Components

### Architecture Diagram

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



### Database Constructs, Model Classes and Entity classes

This Phase comprises identifying and developing following components

1. Database Constructs like Tables, Views, Integrity Constraints, Indexing, Sequences.
2. Entity Classes and
3. Model Classes

which are required to be used in the next Phases.

Entity classes are the ones which are mapped with underlying Database tables, Entity classes also includes corresponding mappings between them, such as

* One to One Mapping
* One to Many Mapping
* Many to One Mapping
* Many to Many Mapping

### Development of individual Microservices

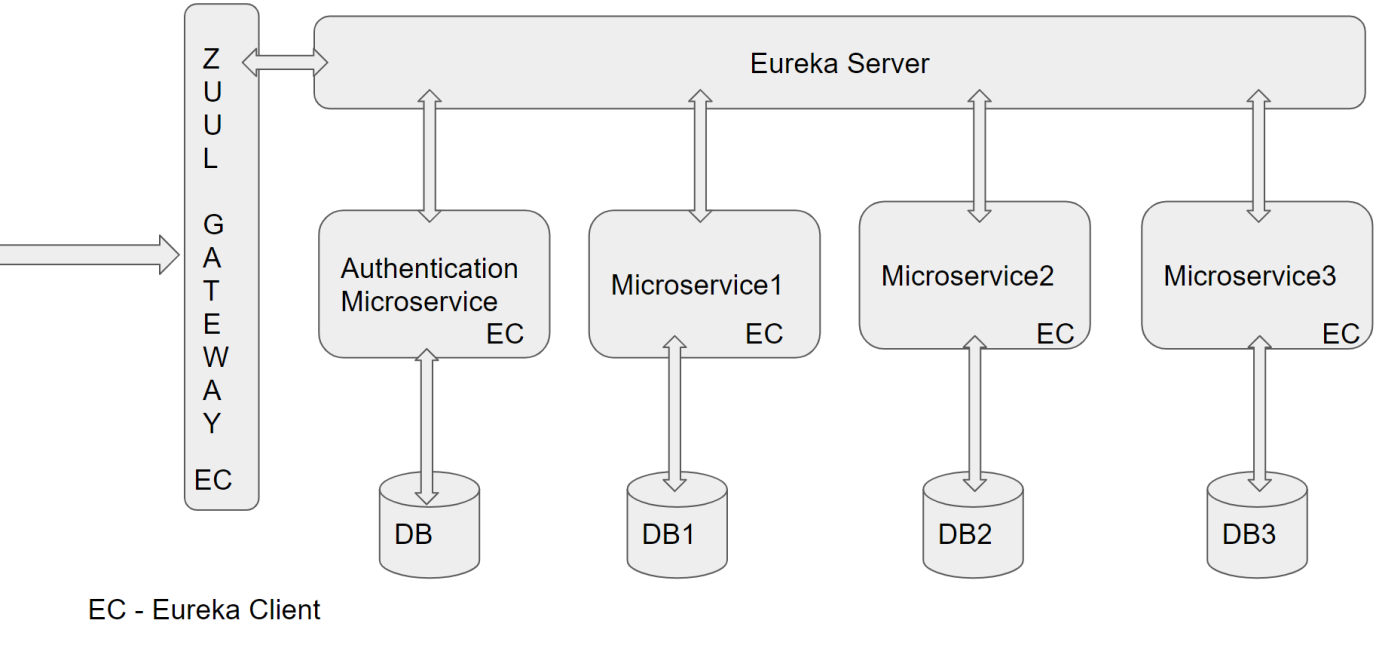
This specific phase is to design/develop individual Microservices. Analyze the requirement and divide back end functionality into multiple Microservices. Based on the Website administrator requirements

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

1. REST Controllers
2. Services
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.
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 wherever custom functionality required
9. Feign Client can be used to invoke one Microservice, from another Microservice
10. Use Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice
11. Unit Testing code can be developed using JUnit, Mockito, and perform Unit Testing

### 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. as shown in architecture Diagram.



Zuul Gateway(create a Zuul based Project using Spring Initilaizer or STS IDE), add required annotation. Authentication and JWT Token validation can be performed in Zuul’s Pre Filter.

Add below details to yml or property file

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
* 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.

## DevOps Activity

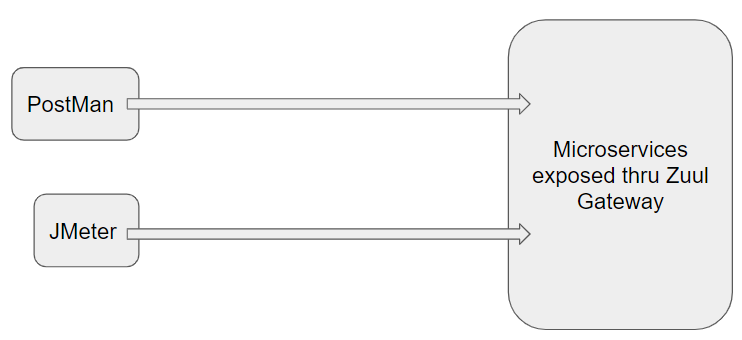
This phase includes performing below Activities

**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. To Setup Docker Client on your VM please refer

**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.

**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

### Perform CI/CD

#### Assessment Deliverables

1. Jenkinsfile, Jenkins screenshots
2. Dockerfile and Docker commands used and Screen shots of usage of Docker

# Important Instructions

Adhere to the design specifications mentioned in the case study.

1. Please make sure that your code does not have any compilation errors while submitting your case study solution.
2. The final solution should be a zipped code having solution. Solution code will be used to perform Static code evaluation.
3. Implement the code using best design standards/family Design Patterns.
4. Use Internationalization for all the labels and messages in Rest API Development.
5. 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.
6. 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.
7. Write web service which takes input and return required details from database.
8. Use JSON format to transfer the results.