**Feedback Management System**

**FSE - Practice Case study**

**1.0.0**

****

Table of Contents

[1 Important Instructions 2](#_Toc28342609)

[2 Skill list: 3](#_Toc28342610)

[3 Business-Requirement: 4](#_Toc28342611)

[3.1 Problem Statement: 4](#_Toc28342612)

[3.2 Design and architecture 5](#_Toc28342613)

[3.3 Requirements: 6](#_Toc28342614)

[3.3.1 Requirements Flow: 6](#_Toc28342615)

[3.3.2 Wireframes: 9](#_Toc28342616)

[4 Development Environment – Select any one 23](#_Toc28342617)

[4.1 Java Stack 23](#_Toc28342618)

[4.1.1 Development Checklist 23](#_Toc28342619)

[4.1.2 Micro Layers and Development Instructions: 23](#_Toc28342620)

[4.1.3 Case Study Deliverables 28](#_Toc28342621)

# Important Instructions

1. Follow the design specifications mentioned in the practice case study. You are free to improvise certain specifications mentioned in the case study.
2. Please make sure that your code does not have any compilation errors while submitting your case study solution.
3. The final practice case study solution should be deployed in Docker on http server.
4. The UI/UX for the app should be multilingual.
5. UI /UX should be responsive across multiple devices.
6. It should be the **progressive web app** such that certain parts of the application are accessible in absence of connectivity. **It can be tested only after application is built and deployed locally at** [**http://localhost:portnumber**](http://localhost:portnumber) **OR** [**http://ipaddress:portnumber**](http://ipaddress:portnumber)**.**
7. For UI/UX, animate components where required using **via CSS or React-Animation.**
8. **Running the Docker image should run your project** at <http://localhost:portnumber> or <http://ipaddress:portnumber>.
9. The final practice case study solution **should show the maximum code coverage of 90% plus.**

# Skill list:

This practice case study will help you to be proficient and pursue the knowledge of the skills required for a full stack developer and have confidence attend the Full Stack Engineer program assessment (i.e. final Skill Base Assessment) to get yourself certified as Full Stack Engineer.

**Your final Skill Base Assessment (SBA) will comprise of the below skillsets/ tools in the respective tracks.**

This practice case study will comprise of subset skills/tools used in your final SBA certification. Please refer to [4. Development Environment](#_Development_Environment_–) section for more details.

|  |  |
| --- | --- |
| **Modules** | **Java Track** |
| **UI/UX** | HTML 5, CSS and Bootstrap, JavaScript, jQuery and ReactJS |
| **Middleware** | Core Java, Spring Core, Spring WebFlux (Reactive Programming) |
| **Database** | Spring Data Reactive, MySQL (using R2DBC) |
| **Integration** | REST, Micro Services, Spring Boot 2.x |
| **Agile/DevOps** | DevOps Concepts, Junit, SpringTest, Jasmine, Git, GitLab, Maven, Jenkins, GitLab Runner, Docker |
| **Deployment Platforms** | Tomcat |

# Business-Requirement:

## Problem Statement:

1. The proposed Cognizant Outreach – Feedback Management System, as the name suggests is used to manage the Cognizant Outreach event feedbacks from the participants.

This includes automated real-time processing of the feedbacks and respond to participants with an email response for their valuable feedbacks. The system will automatically look for events details and its participant’s details excel in an input path (The input files are attached to this document and the path must be configurable in code). Once the files are made available, the system will start processing them and save the entries in the database. The processed excel files will then be renamed by appending current date-time and moved to processed directory for any future reference. Any mismatch in the file name or the data format inside the file will make the system to drop the files without processing. Appropriate logs should be maintained. These automated system processing can be monitored in the browser UI. This will give wide set of options and information on these processing.

1. Automating this process makes it error free and no manual intervention is required. In addition, this solves the business requirements as any information can be viewed real-time. Admin users can generate a real-time report based on any filter criteria and the same can be emailed to intended people to give a heads up of the event success. As the system is responsive, it can fit to any device and makes it easily accessible and can also be extended to match Cognizant’s new theme with minimal efforts.
2. The system is developed meeting OWASP top 10 security risks and made it more robust and secure with considering common attacks. Henceforth we can guarantee that the system is secure against attacks as of development date.

#### Input Data Format

The input data files for the application will be placed in a share path or uploaded by Admin. Whenever a file is placed in the share path, it should be automatically processed by the application.

* The system will have four sheets as input.
  + OutReach Event Information – will have the details of the ‘Volunteers Registered and Participated’
  + Volunteer\_Enrollment Details\_Not\_Attend – will have the details of ‘Volunteers Registered and did not participate’
  + Volunteer\_Enrollment Details\_Unregistered – will have the details of ‘Registered and unregistered later’
  + Outreach Events Summary – will have the details of Event POC -Column A - Event ID, Column S - POC ID, Column T - POC Name

Please note:

* In the excel OutReach Event Information – Business Unit (Column M) is the associate business unit.
* City/Country is obtained from Outreach Events Summary - Base location (Column C)
* Project level is obtained from Outreach Events Summary - Project (Column G)



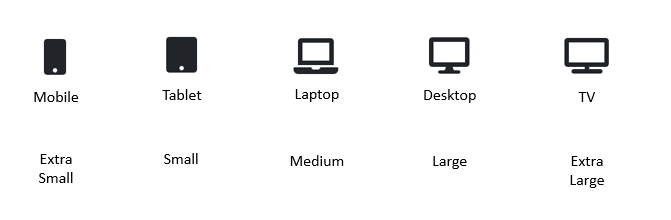


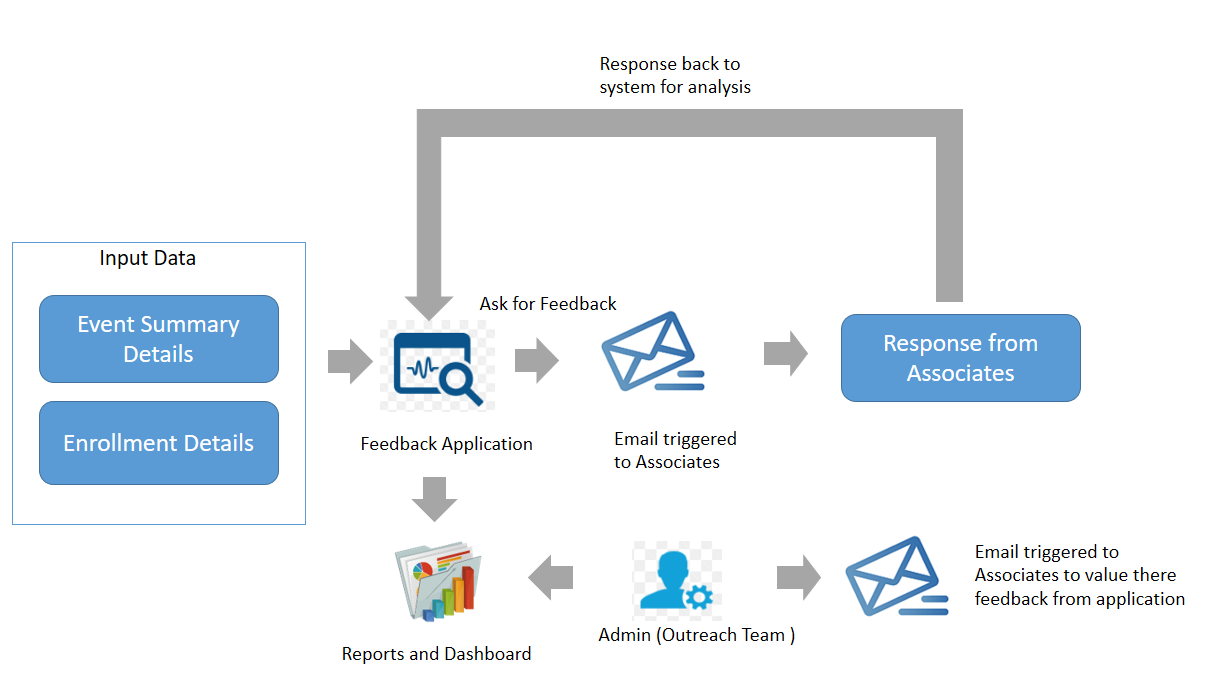




## Design and architecture

The developed system should be responsive and should fit into any screen size as shown below.





## Requirements:

### Requirements Flow:

The requirements are discussed role wise – Admin, PMO, POC.

**Overall Flow:**

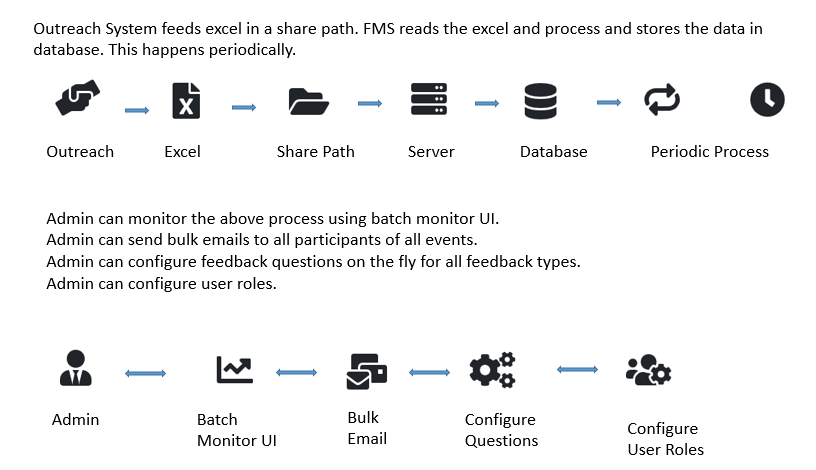
Admin user uses FMS (Feedback Management System) to view the processed events details and trigger emails to participants requesting their valuable feedbacks.

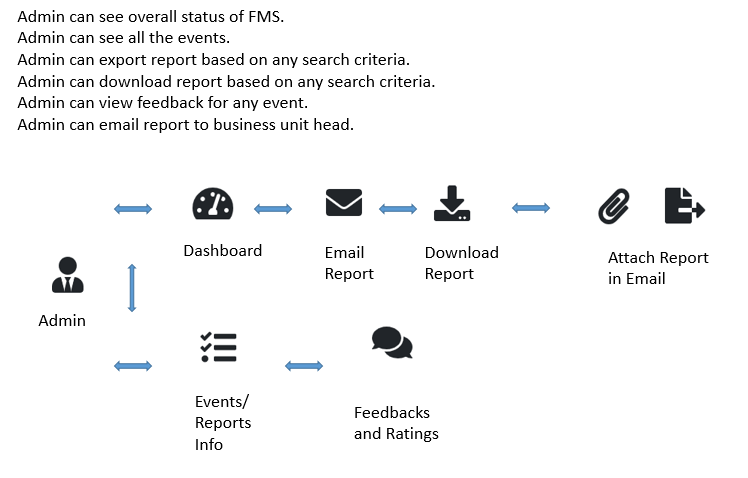
PMO and Admin user can generate report based on the any filter criteria and email the report to business heads describing the events rating and feedbacks.

Roles:

* Admin - will have access to configure the roles, other configurations in the application and all reports/dashboard.
* PMO - will have access to all events reports and dashboard. The PMO details will be provided by Admin to the application.
* Event POC - will have access only to the particular event reports/dashboard for which he/she is the POC.
* Participants can submit their feedback and this is process by FMS in real-time and can be viewed by Admin, PMO and POC users.

#### Admin Requirements:





1. Need to Login with a valid Admin credential.

2. In Dashboard, Admin can see the statistics for all the events.

3. In Events Screen, Admin see all the events. Admin can trigger feedback request email for all the participants of all the events at one go.

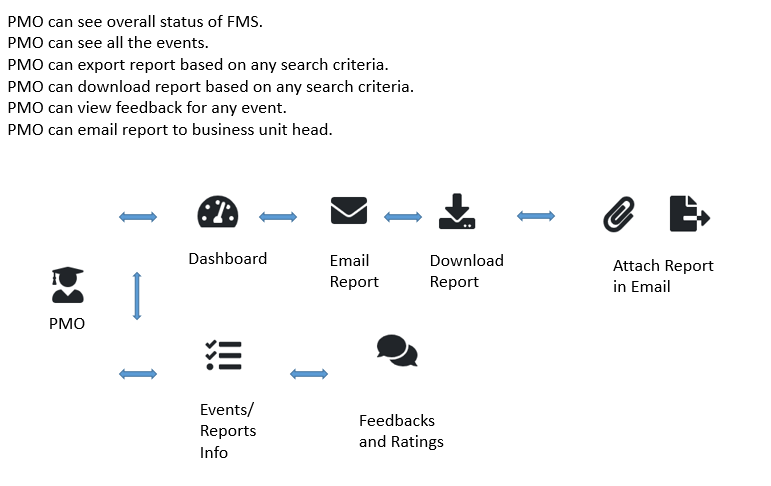
4. In Events Detail Screen, Admin can trigger feedback request email for all the participants of the particular event and view the details of the event.

5. In Reports Screen, Admin can see the reports for events based on any filter criteria and email the filtered report to BU heads.

6. In Role Screen, Admin can add/remove PMO roles for the active cognizant employee.

7. In Questions Screen, Admin can add/edit feedback questions in real-time that needs to be displayed to the user for sharing their feedback.

#### PMO Requirements:



This flow is same as Admin except that PMO has no access to configuring roles, questions and send feedback request emails.

1. Need to Login with a valid PMO credential.

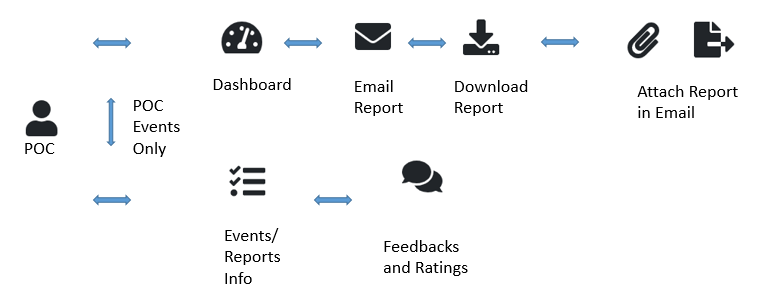
2. In Dashboard, PMO can see the statistics for all the events.

3. In Events Screen, PMO see all the events.

4. In Events Detail Screen, PMO can view the details of the event.

5. In Reports Screen, PMO can see the reports for events based on any filter criteria and email the filtered report to BU heads.

#### POC Requirements:



This flow is same as PMO except that POC can see only the details of the events he/she is mapped to.

1. Need to Login with a valid POC credential.

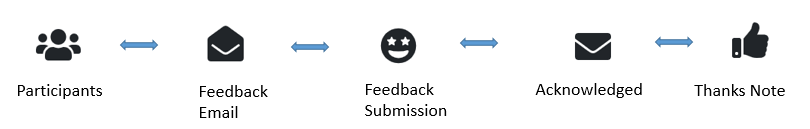
2. In Dashboard, POC can see the statistics for the events he is mapped to.

3. In Events Screen, POC see his events.

4. In Events Detail Screen, POC can view the details of the event.

5. In Reports Screen, POC can see the reports for events based on any filter criteria and email the filtered report to BU heads.

**Participant Flow:**



1. Participant will receive email requesting to share the feedback.

2. Participant need to click on the link in the email to share the feedback.

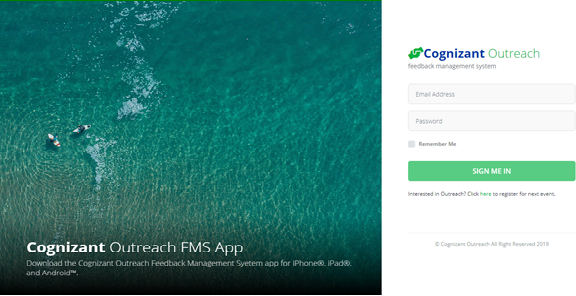
3. Based on the Participant’s status (participated/not participated/unregistered), Participant will be redirected to respective feedback screen to share the feedback.

4. Upon receiving the feedback, email response will be triggered to value their response and user will be redirected to thanks screen.

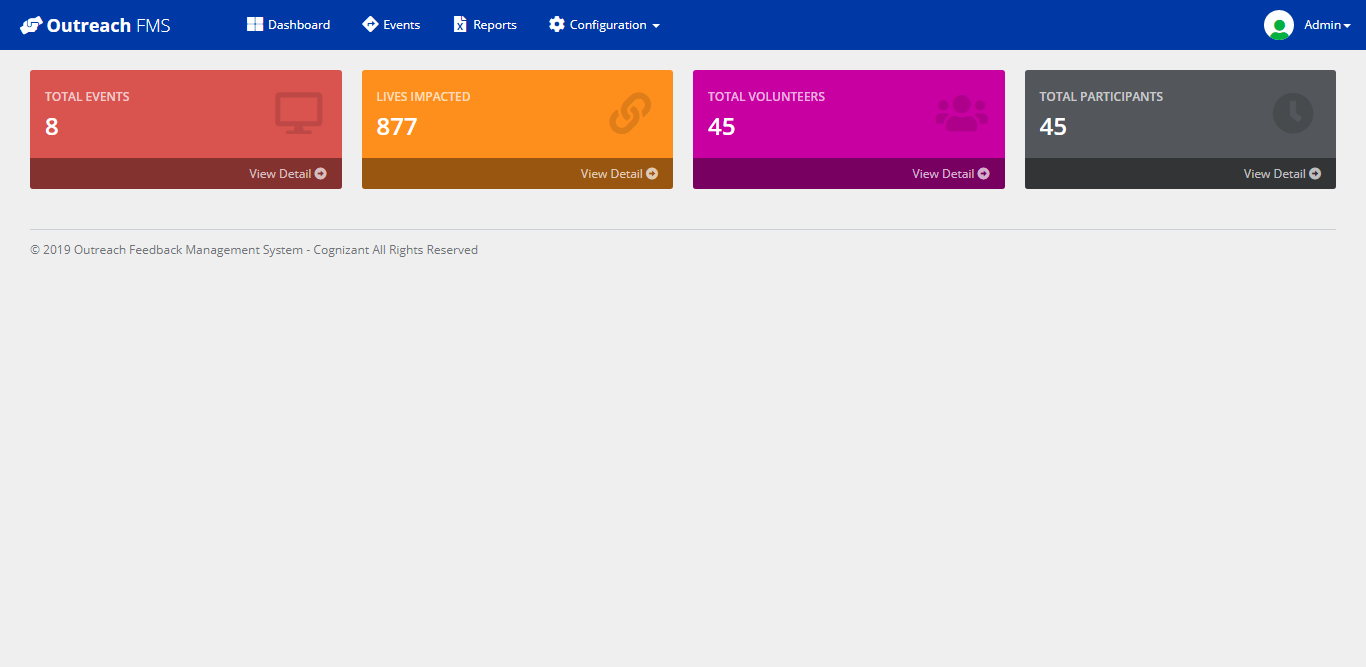
### Wireframes:

#### Admin Wireframe:

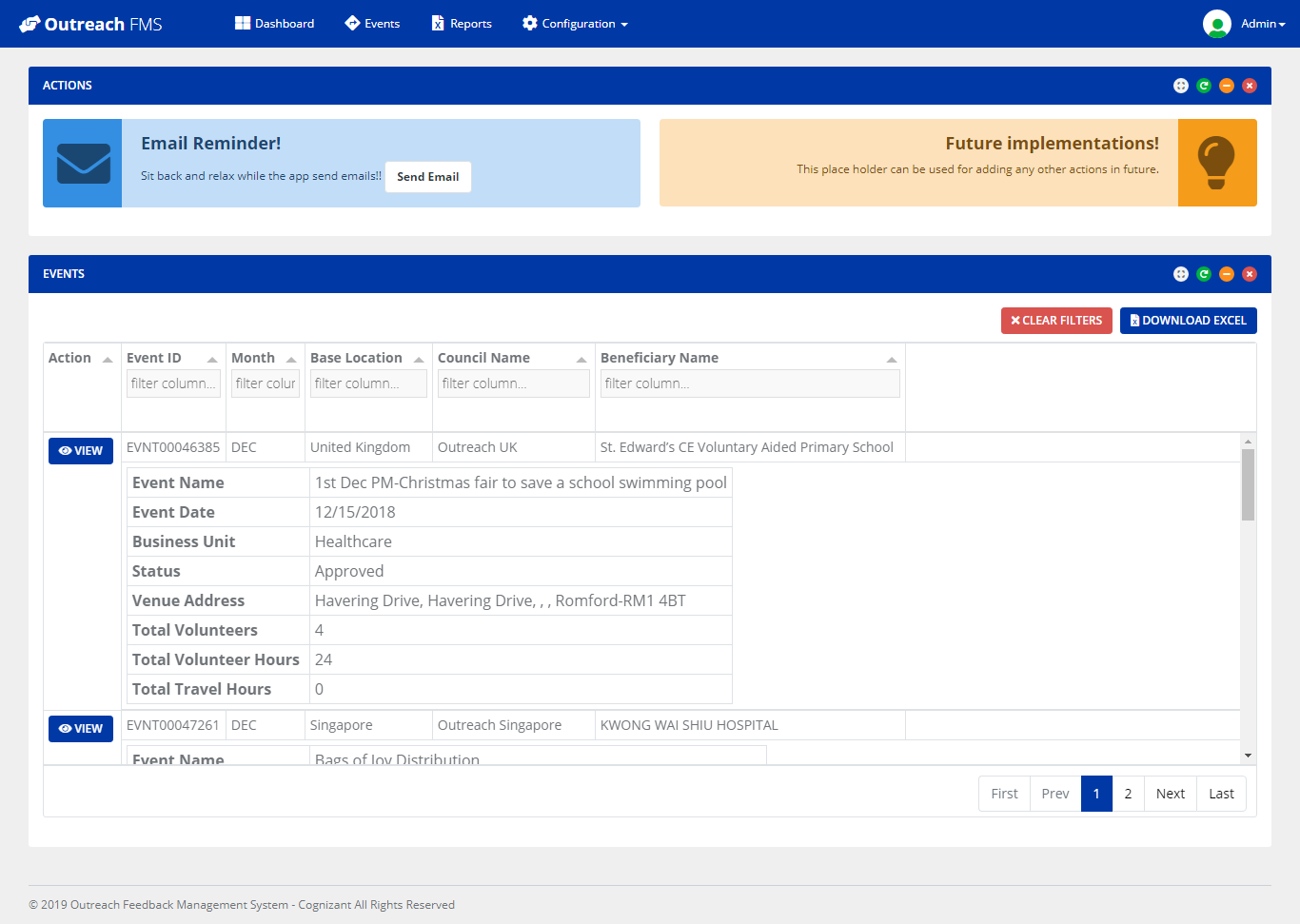
1. Login



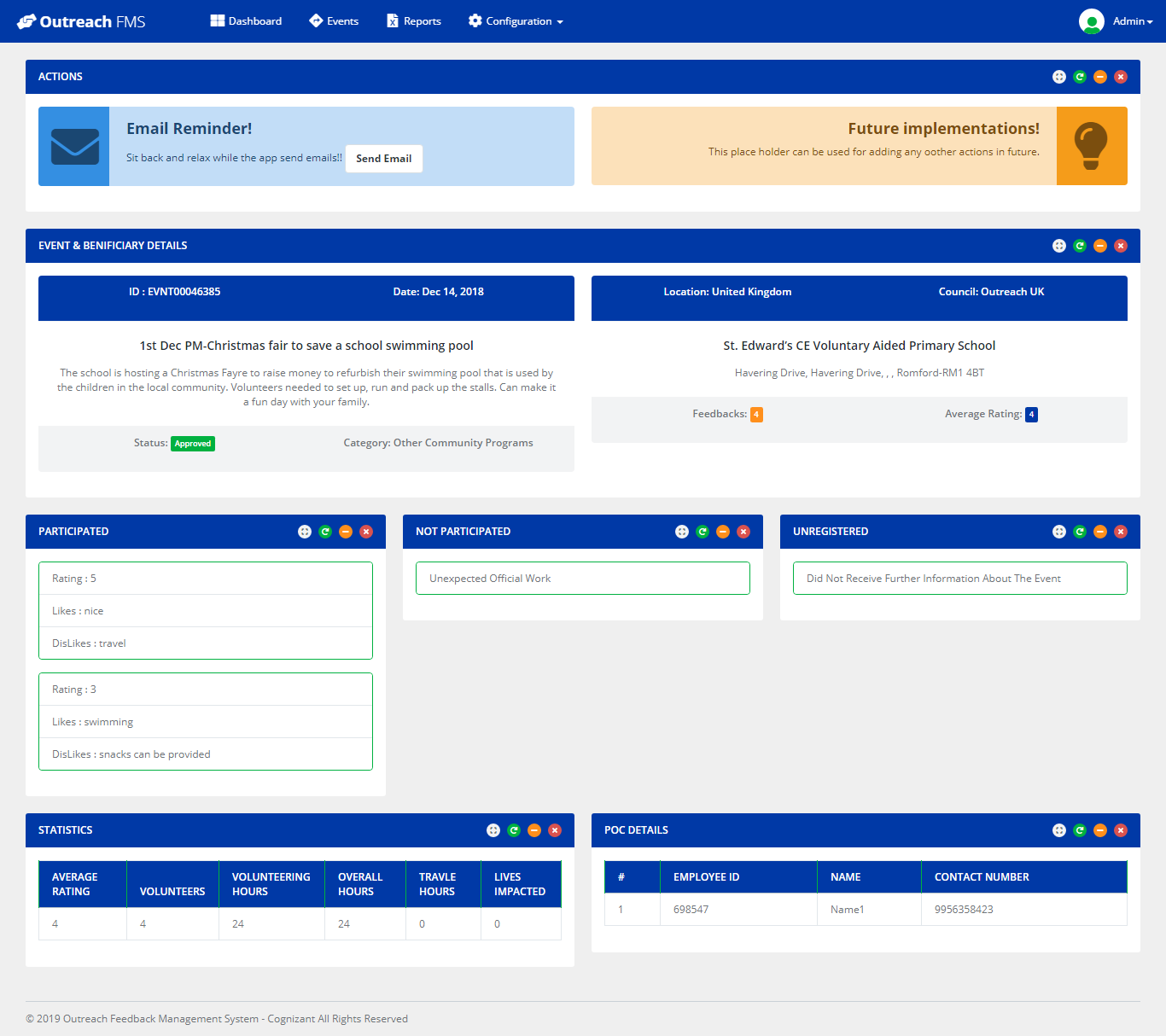
1. Dashboard



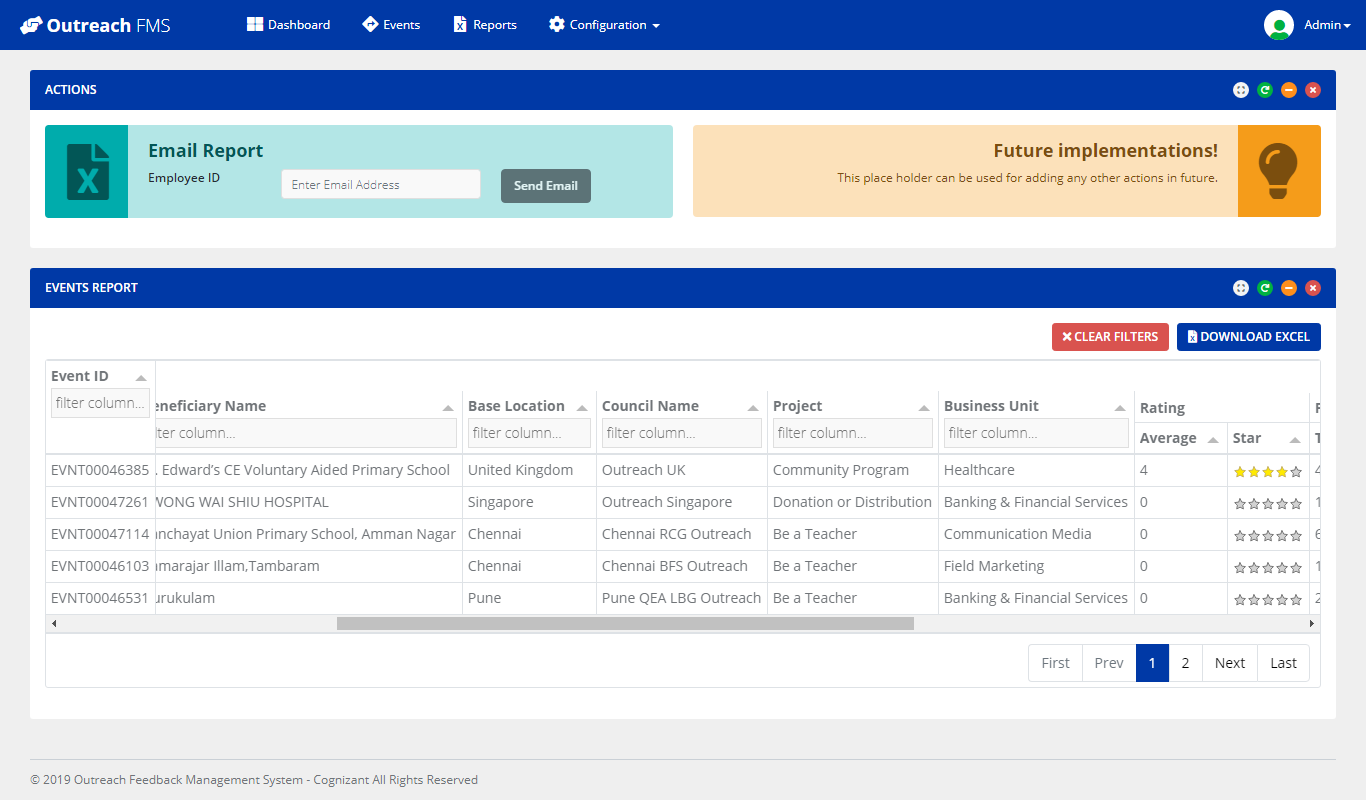
1. Events List



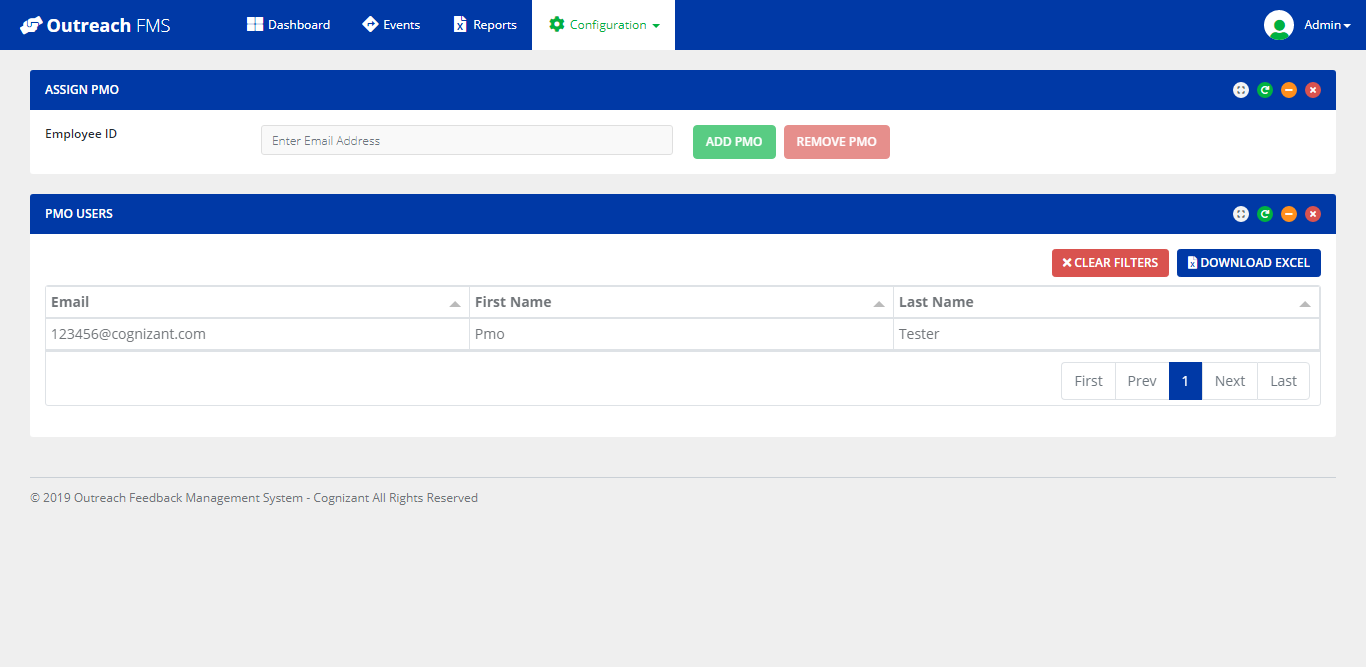
1. Event Detail



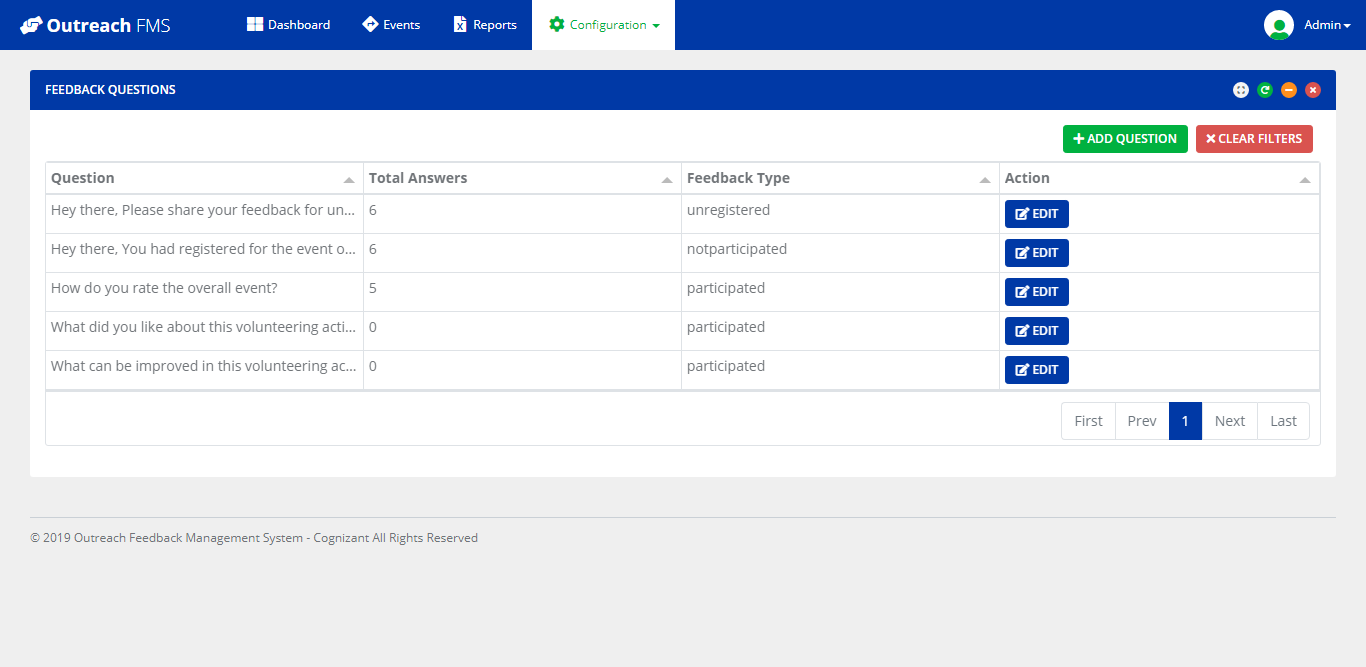
1. Reports



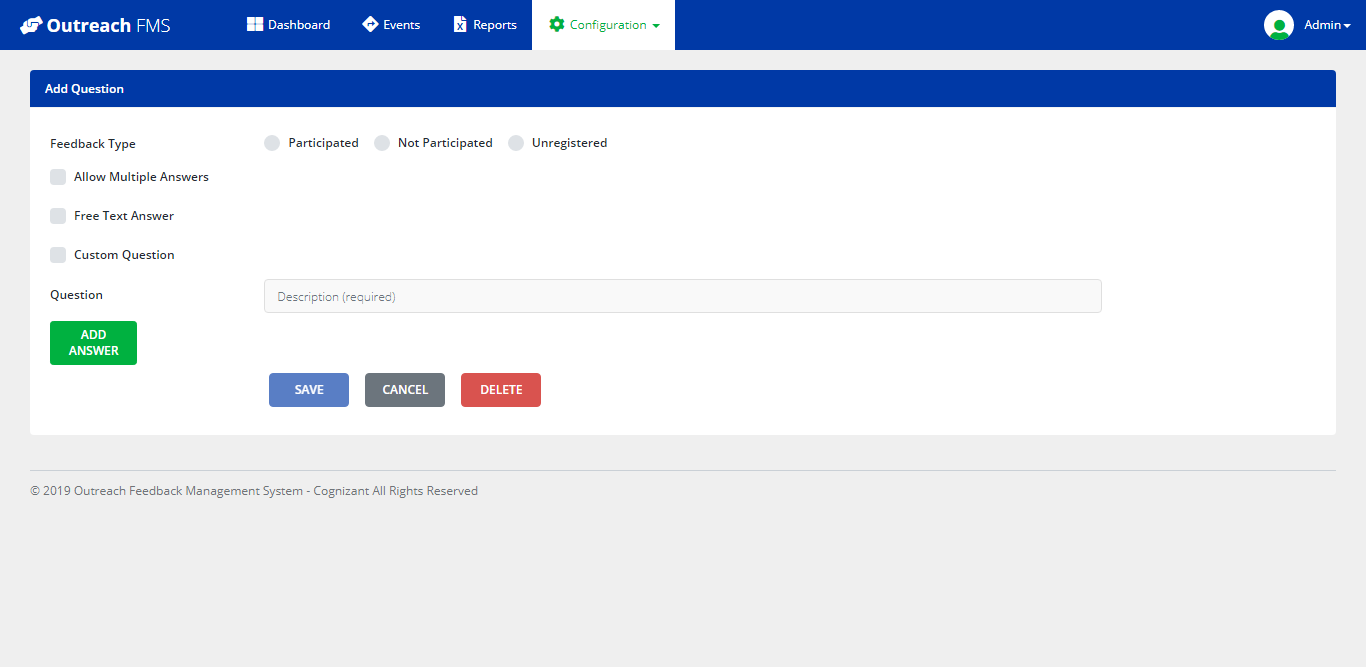
1. Roles

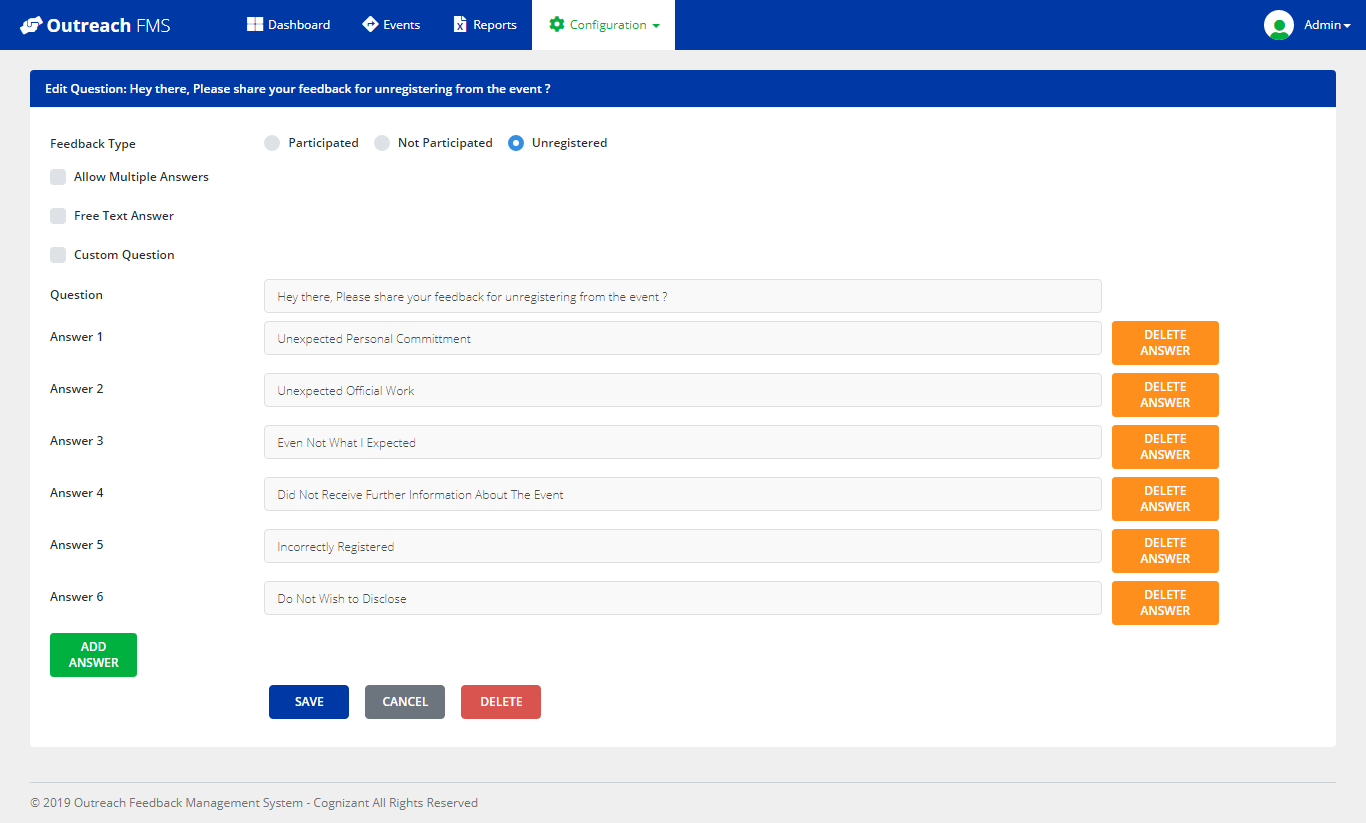


1. Feedback Questions



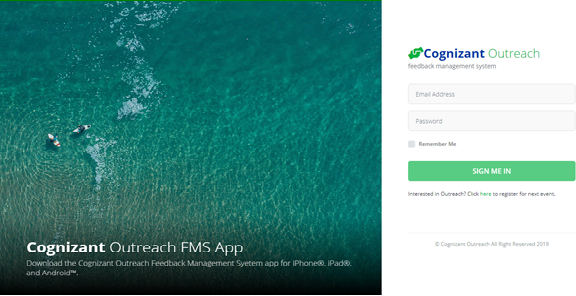
1. Feedback Questions Add/Edit



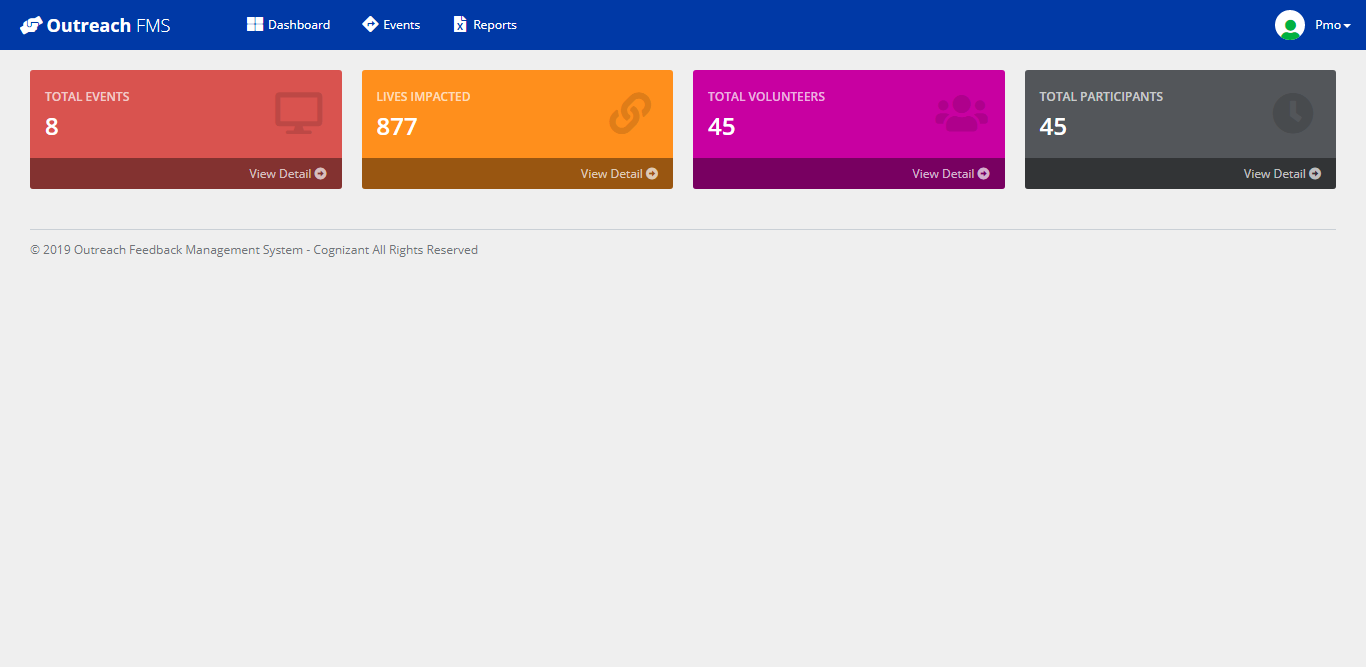


#### PMO Wireframe:

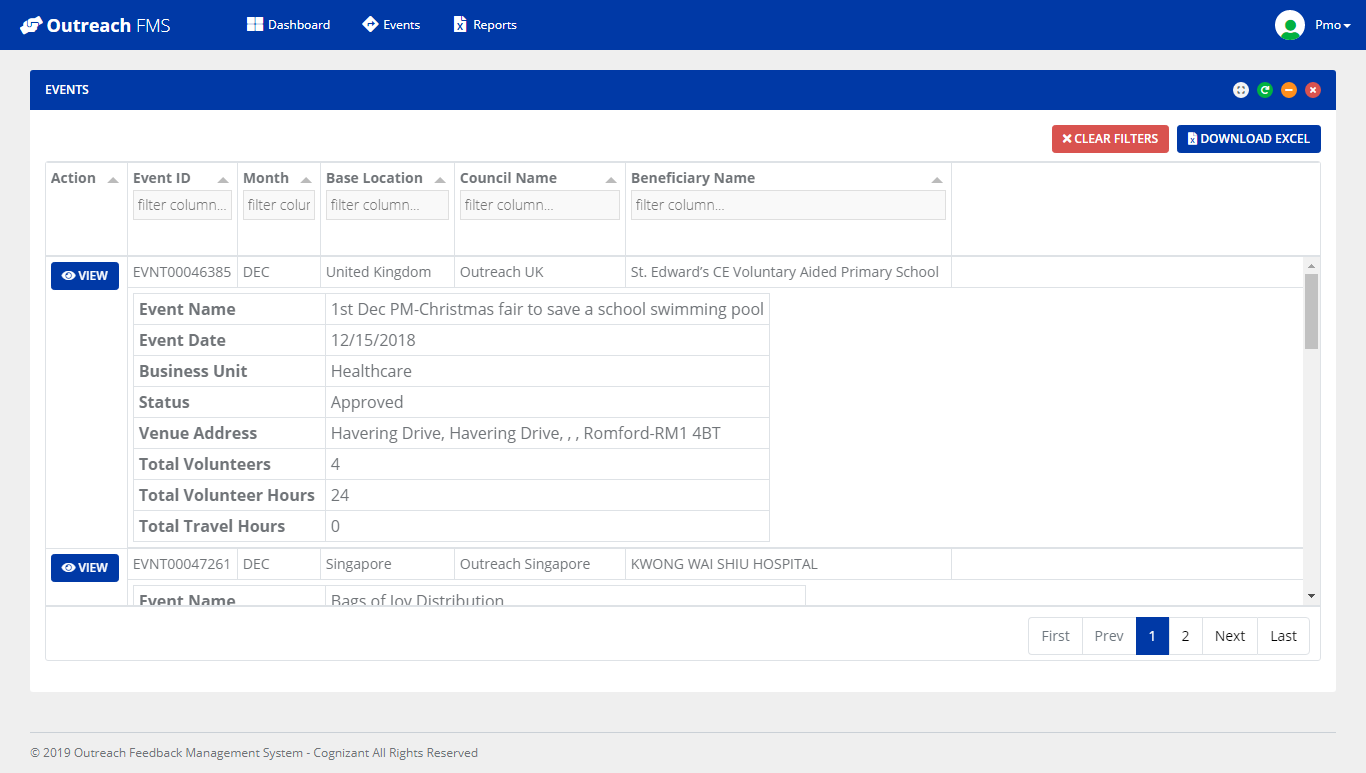
1. Login



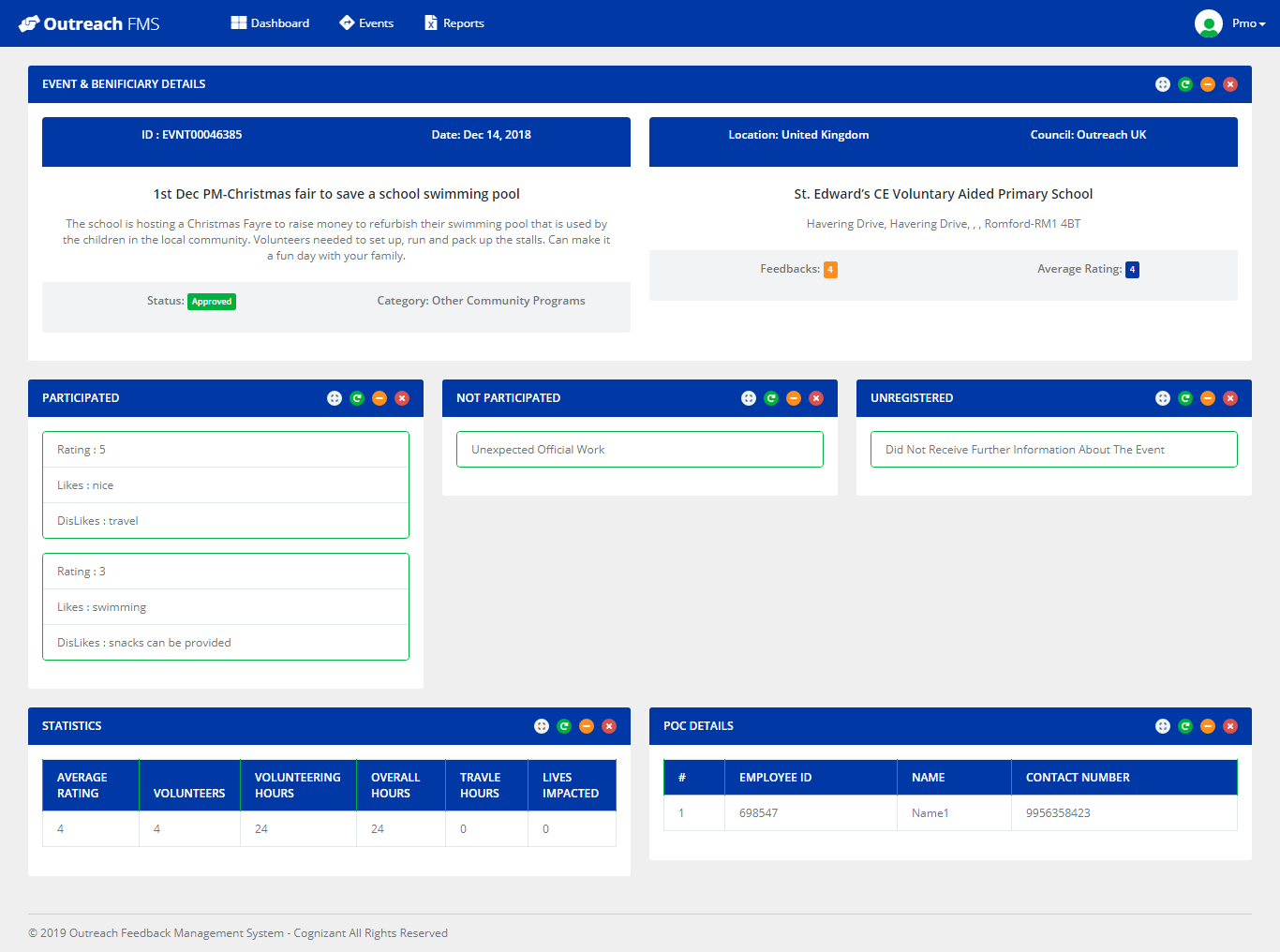
2. Dashboard



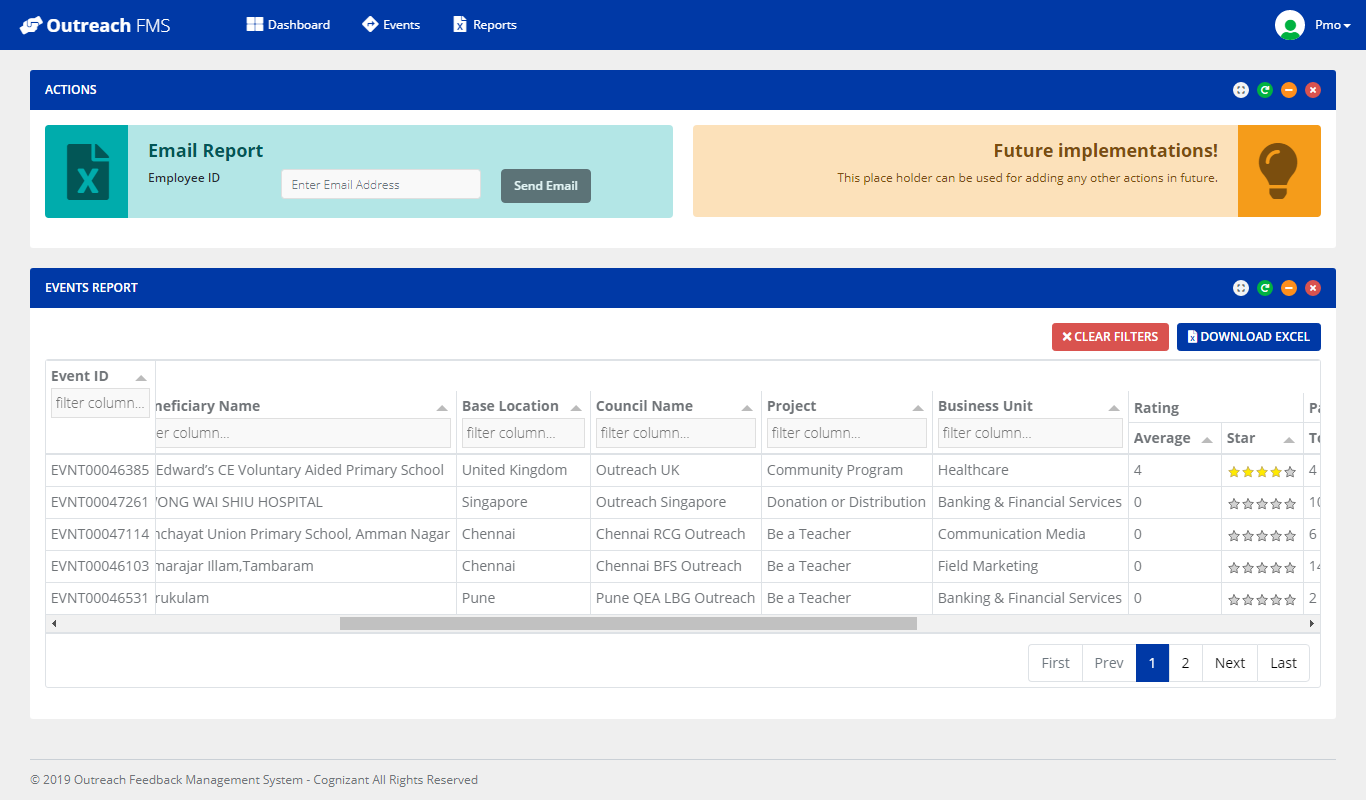
3. Events



4. Event Details

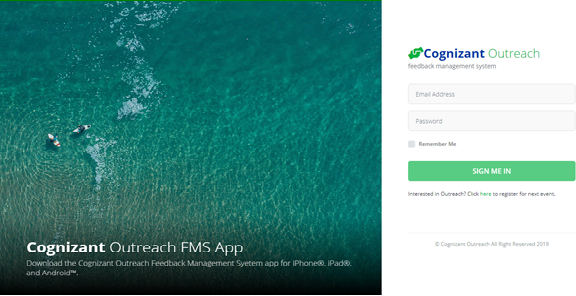


5. Report

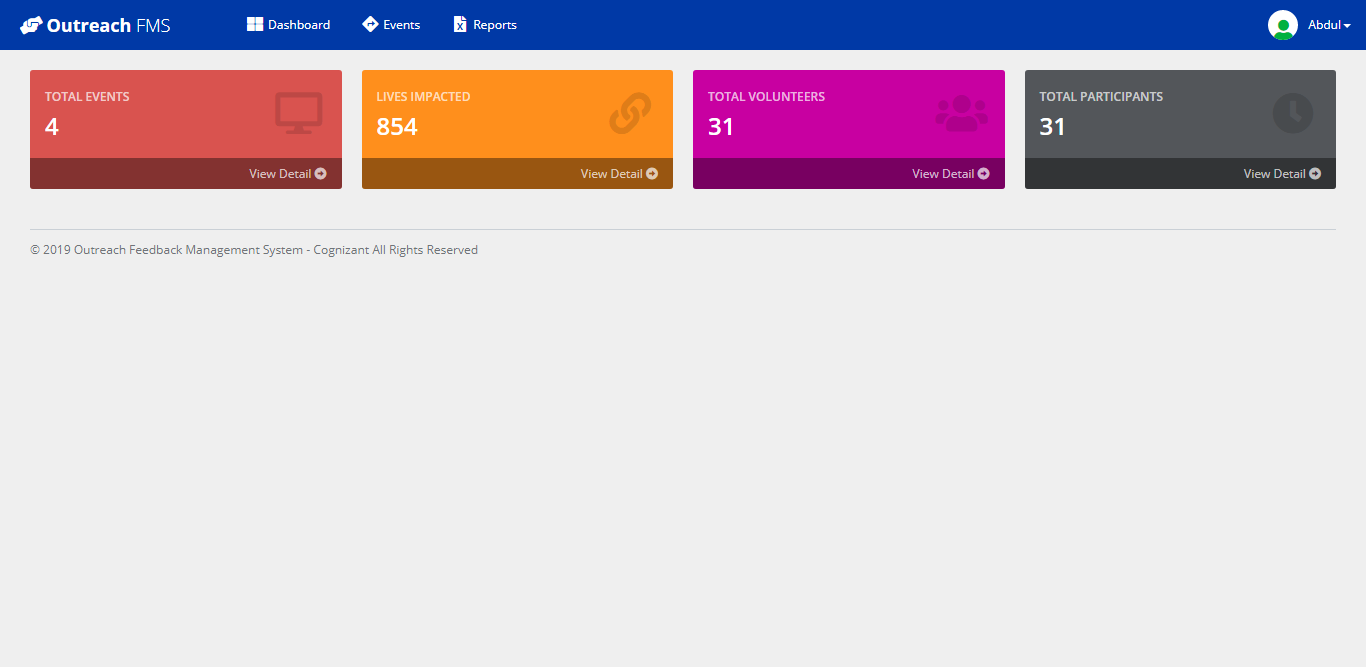


#### POC Wireframe:

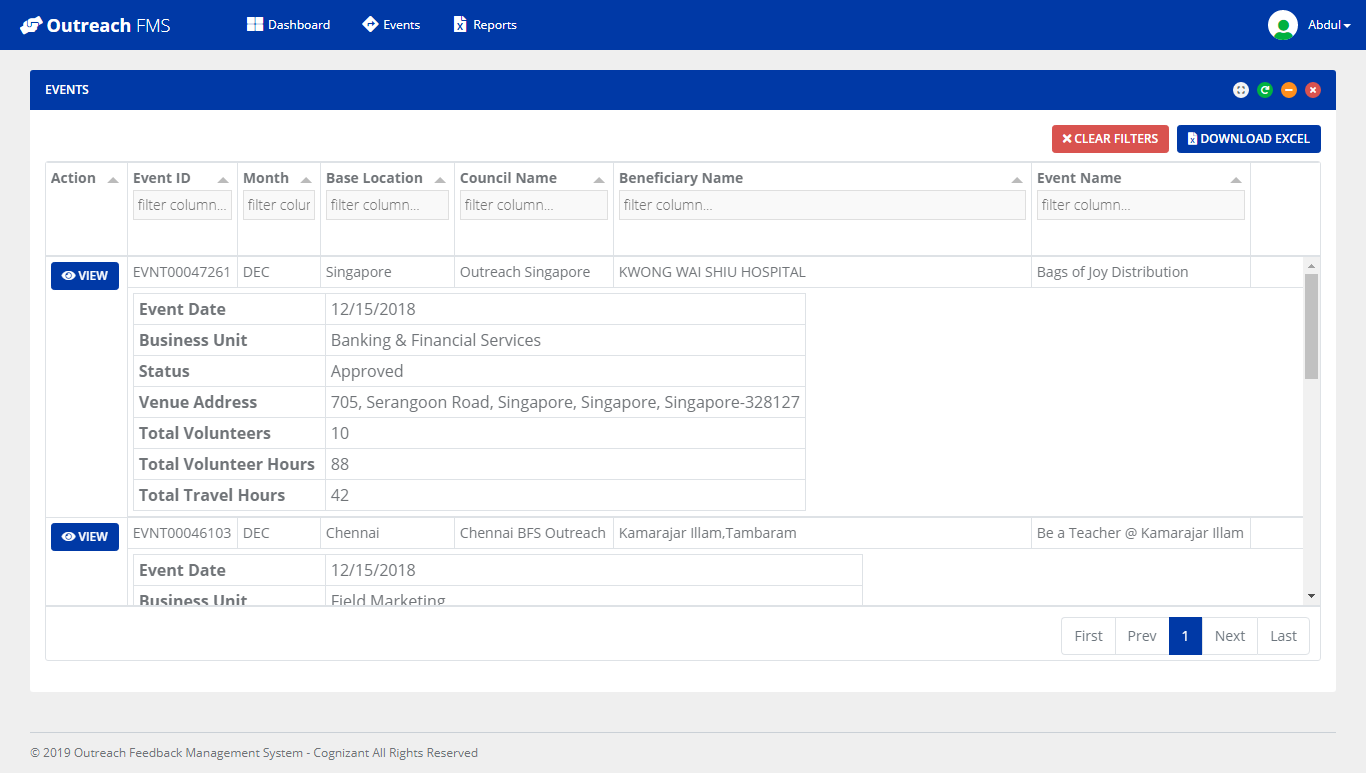
1. Login



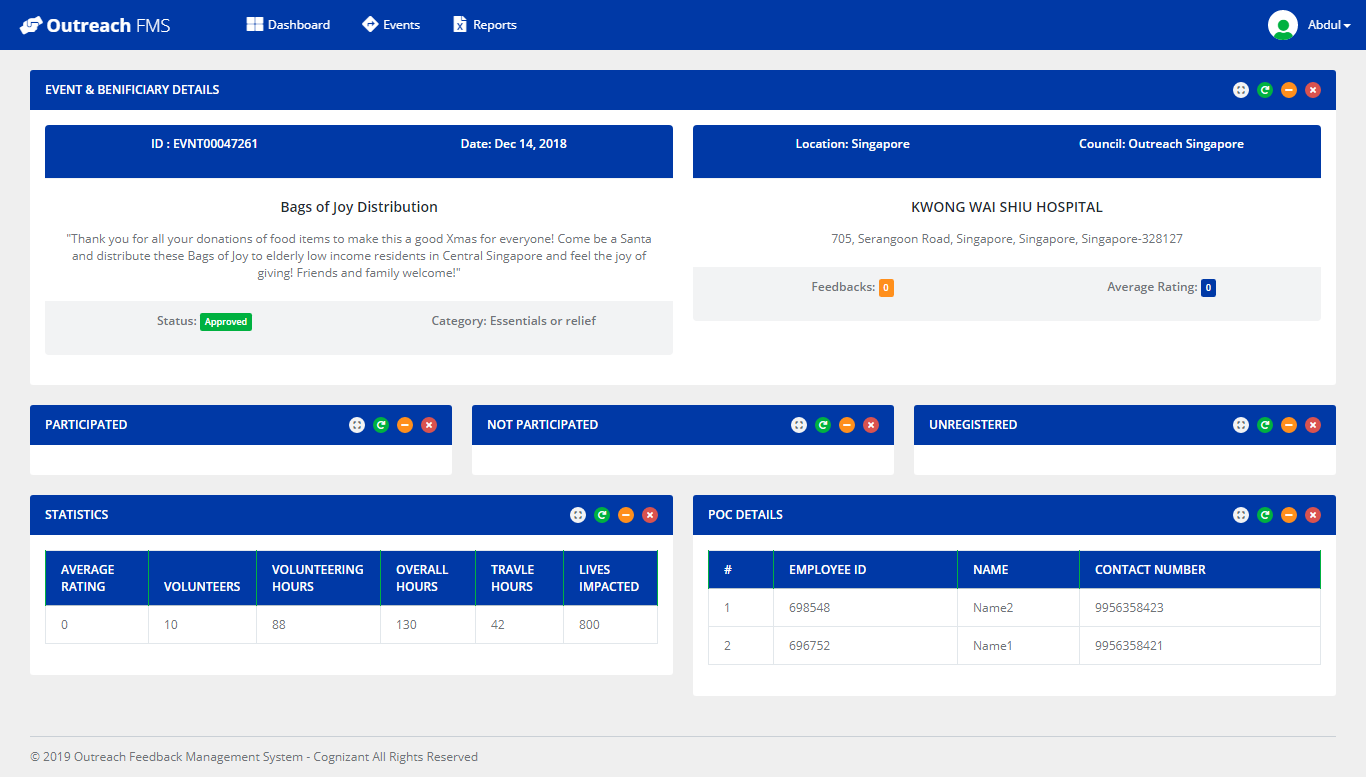
2. Dashboard



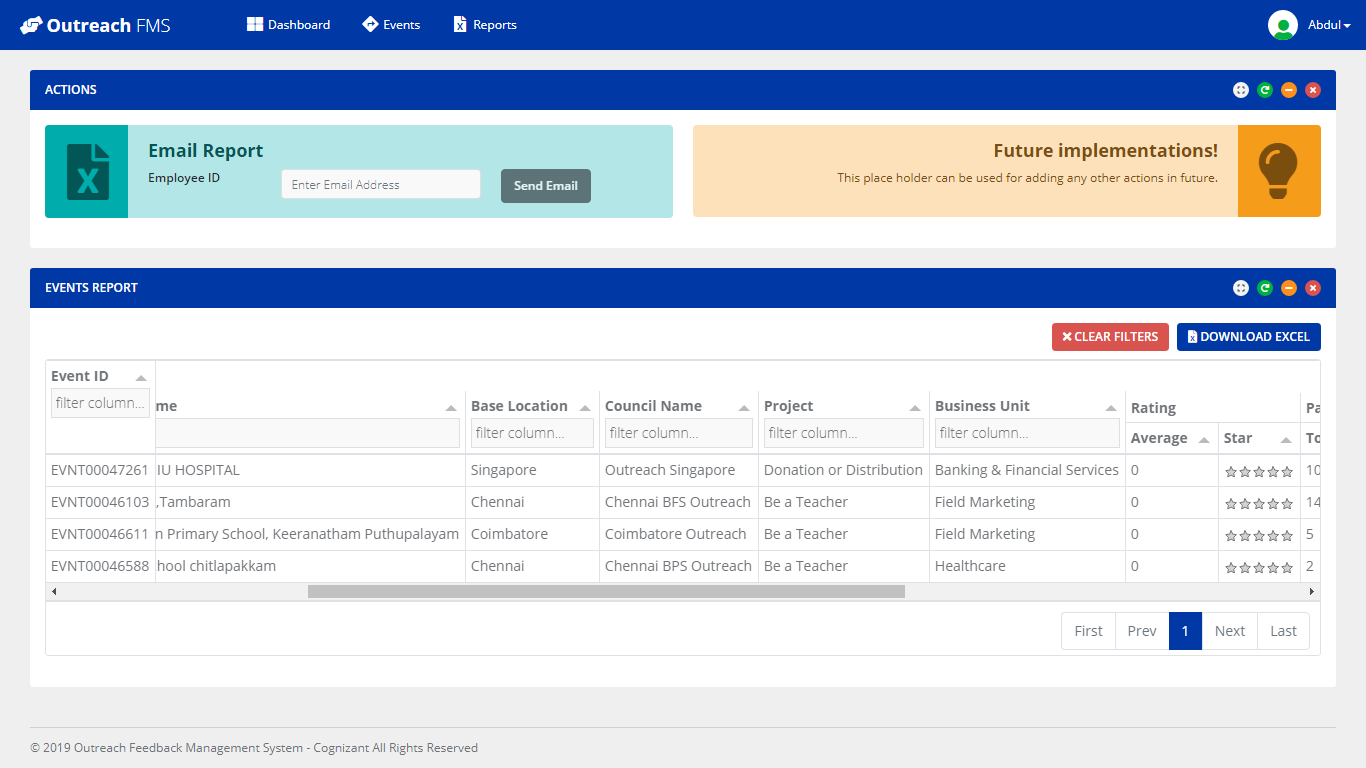
3. Events



4. Event Detail

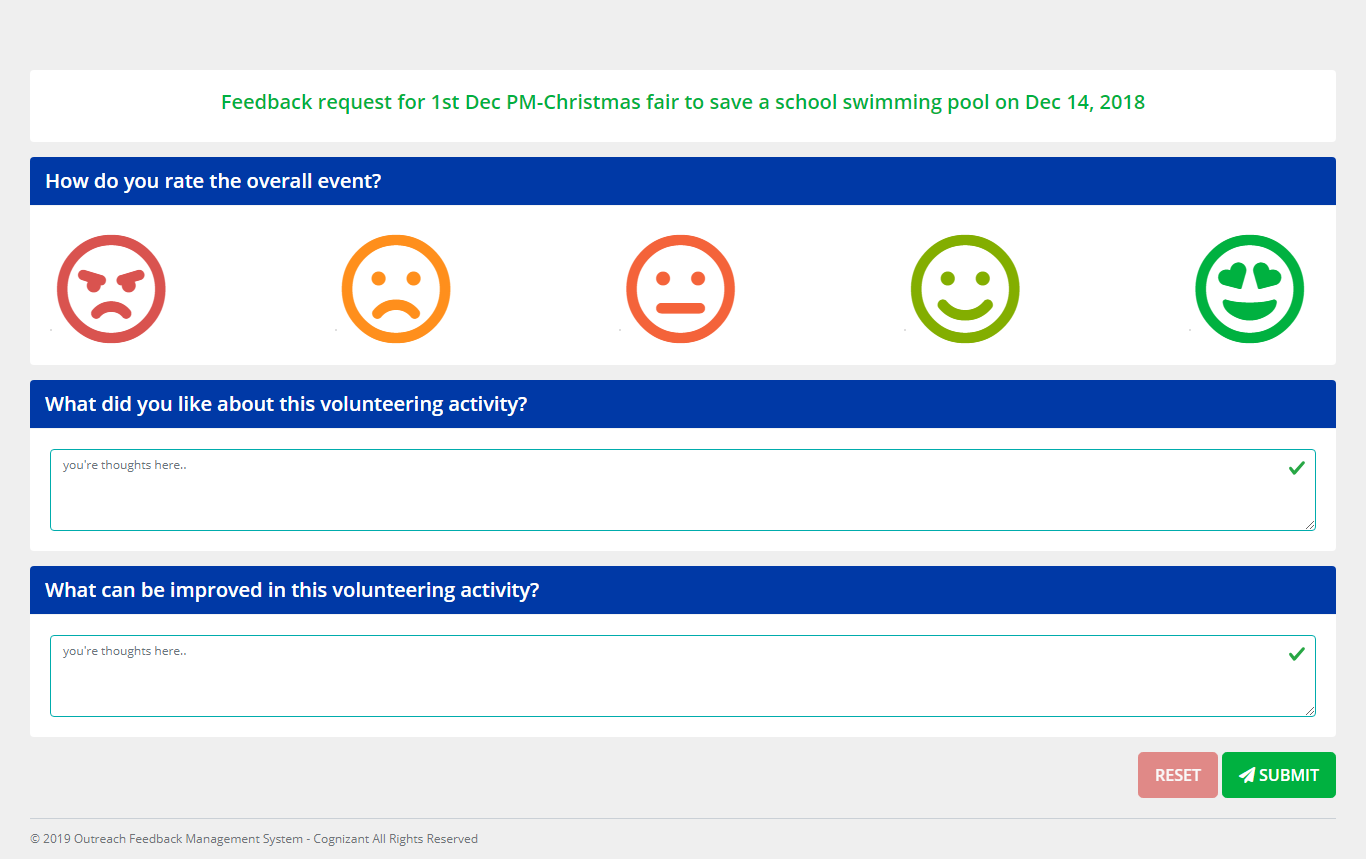


5. Report

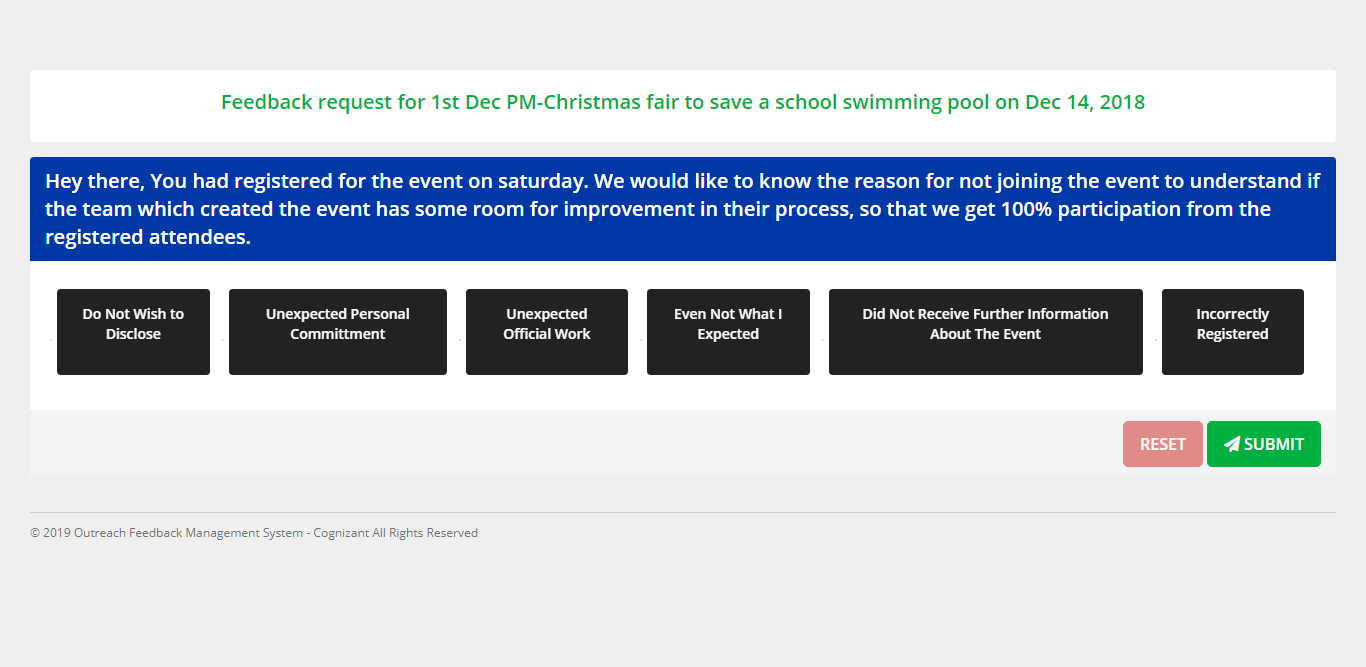


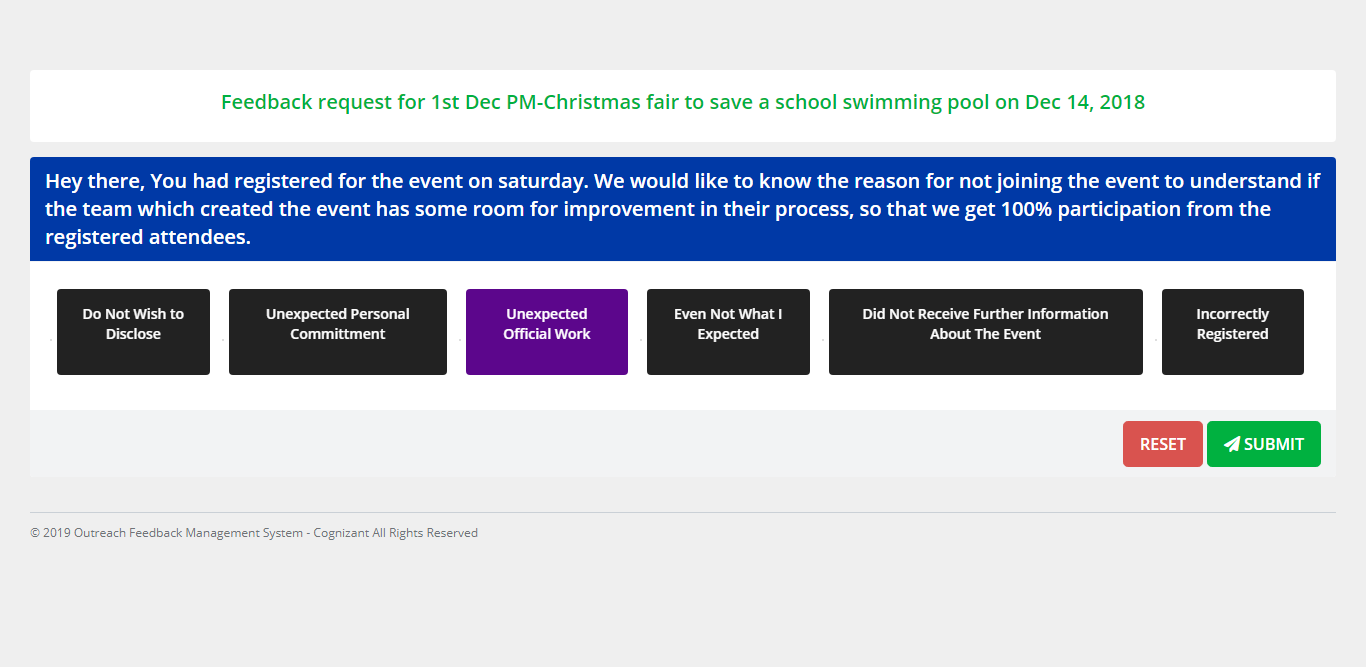
#### Participant Wireframe:

1. Participated Feedback Flow

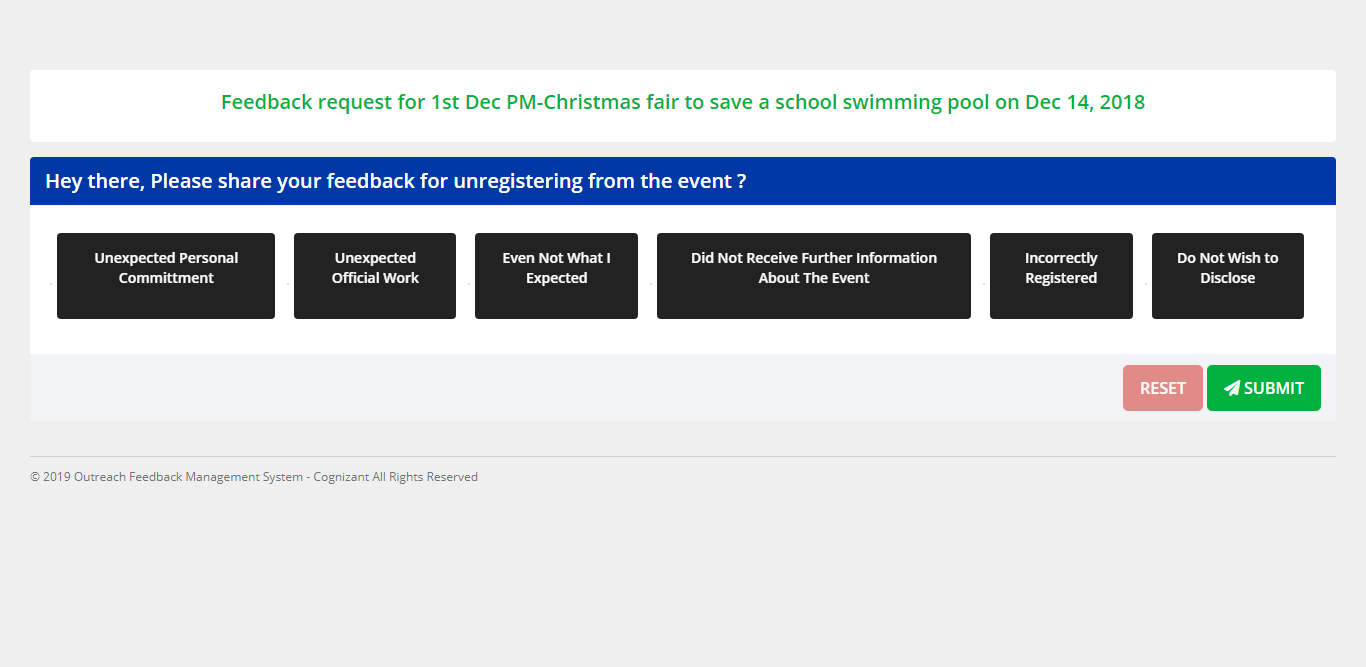


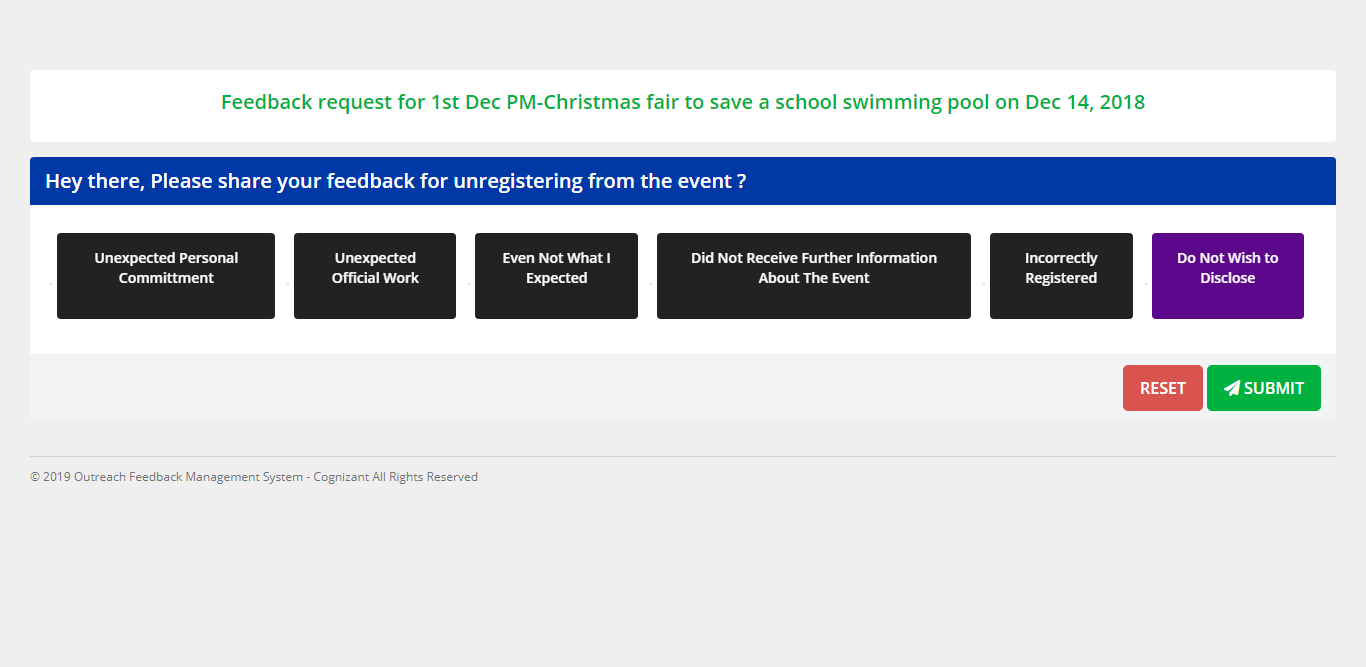
2. Not Participated Feedback



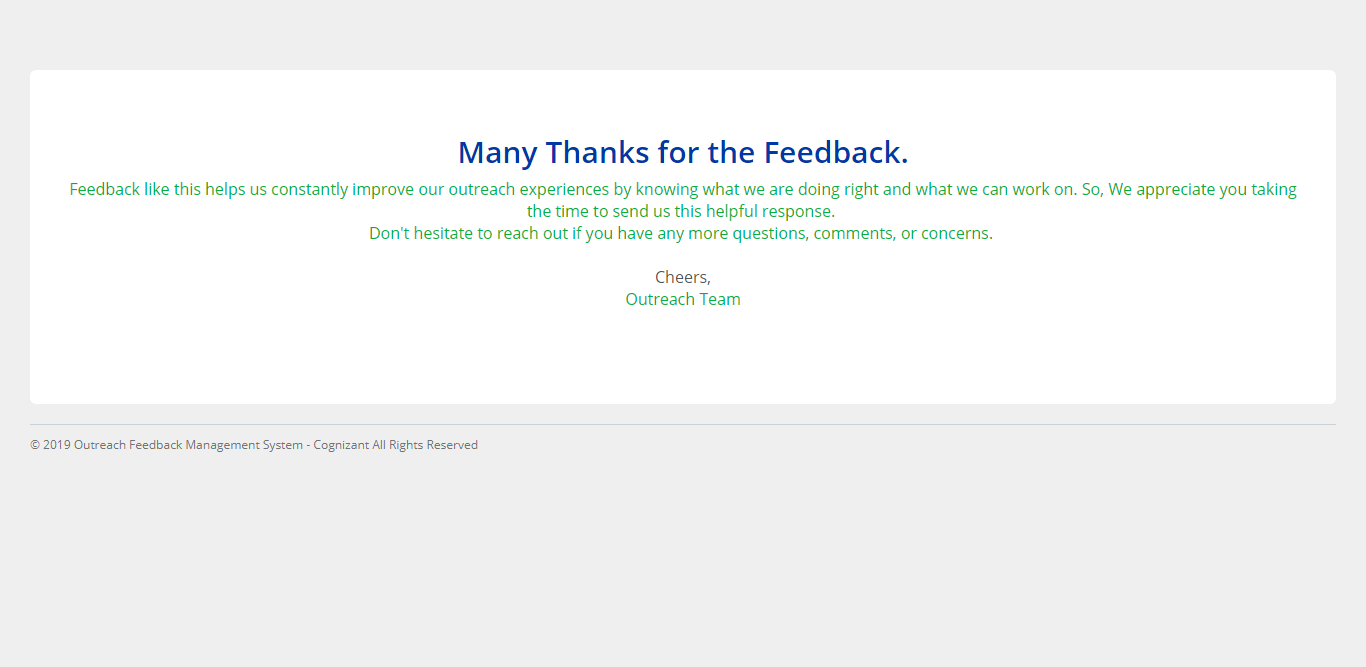


3. Unregistered Feedback





4. Info Screen After submitting feedback



# Development Environment – Java Stack

## Java Stack

### Development Checklist

Please ensure that you follow the below attached checklist while developing the case study. These guidelines will be used to assess your final SBA as well.



### Micro Layers and Development Instructions:

This section will allow you to understand the micro layers of developing the case study along with development instructions and environment.

#### UI/UX Layer

**Development instructions to this layer:**

1.       Front End need to be divided into multiple components to accommodate the pages. (Components has to be divided into Container components and child components)

2.       For Authentication, store JWT token in Local or Session Storage. REST APIs are invoked from the corresponding Services,

3.       As known JWT token is generated on the Server side and received by Client on successful authentication.

4.       Use React-Redux to maintain state for each major components

5.       In UX, you will use react, Javascript and Jquery where required.

6.       UX includes development of

a)       Stateless components (Which does not have any state and just the render the elements in DOM)

b)      Container components with redux & react-redux (where the state is maintained)

c)       Child components (Where the props are passed from container components)

7.       Use react-router-dom feature to display Navigation links on Web Pages. When User clicks a Navigation link, the corresponding component should be displayed.

8.       Use redux & react-redux effectively to handle data mutations and state management

9.       Maintain store for major container components (use Redux store/reducers/effects – Fetch the response data from API and update the store)

10.   Data from store needs to subscribed in container components and has to be shared with child components.**​**

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| HTML5 and CSS3 |
| Bootstrap |
| React JS |
| Jquery/Javascript |
| GIT/Gitlab |
| Jenkins |
| Docker |

#### Middleware Layer

**Development instructions to this layer:**

1. In this micro layer, you will develop Spring WebFlux (Reactive Programming) web application using Java 8 and Spring 5.
2. Analyse the requirement and create Microservices for each functionality.
3. Design the service endpoints of the microservice using Swagger UI.
4. Generate the skeleton code of the project using Swagger Codegen (Annotation-based Programming model).
5. Use both the below programming models while implementing the micro services:
   * + 1. Annotation-based Programming model (using @Controller, @RequestMapping, etc). Use swagger-codegen tool for auto code generation.
       2. Functional programming model using Java 8 Lambda style Routing and Handling (Generate code manually).
6. Use operators for manipulating primitive data types. Use enums to define collection of constants.
7. Running the Docker image should run the Spring MVC project at <http://localhost:portnumber> or <http://ipaddress:portnumber>.
8. Use Spring Validation, Spring Exception, Spring Security Reactive where required.
9. Use SL4J for logging.
10. Use Maven as build automation tool.
11. If you wish to use template library (e.g. thymeleaf) in place of JSP for building front-end; please feel free to do so.
    * + 1. <https://www.thymeleaf.org/>
        2. <https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html>
12. Your Business Logic Plan includes:
    1. Declaring methods you plan to use for data validation, file io, data formatting or any other utility classes/methods.
    2. Documenting methods you plan to use for data validation, file io, data formatting or any other utility classes/methods.
13. Your Business Logic includes:
    1. Data Validation
    2. DB IO
    3. Data Formatting for DB IO
    4. Checking proper input from keyboard
14. Use Spring Validation framework (optional) to validate the inputs
15. DI can be used injecting beans to the Spring Application Context.
16. Use SonarQube for maintaining code quality.
17. On Client-side, use reactive “WebClient” instead of “RestTemplate” for consuming services.
18. Use Project Reactor data streams “Mono” (i.e. 0...1 elements) or “Flux” (i.e. 0...N elements) as the return type of all the reactive APIs/Functions.

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| Java 8 |
| Spring Core and Spring Webflux in Spring 5 |
| GIT/Gitlab |
| Jenkins |
| Docker |
| SonarQube |
| SwaggerUI |

#### Database Layer

**Development instructions to this layer:**

1. In this micro layer, you will develop the application using Java 8 and Spring Boot Framework with ORM (Spring Data) support for a MySQL DB.
2. Use R2DBC (Reactive Relational Database Connectivity) for providing reactive programming support for relational database (MYSQL).
3. Reactive data repositories should implement “ReactiveCurdRepository” interface.
4. You will develop Database layer of the respective application with MYSQL support. You will also write queries applicable to each page discussed in the case study.
5. Implement the database, which is normalized and non-redundant.
6. You are free to change the database schema as per the solution discussed in this case study in order to achieve:
   * + 1. Normalization
       2. Non-redundancy
7. For Spring Data implementation, use entity classes and JPA repositories
8. You can use Database Driver and obtain a database connection
9. Use GET/INSERT/UPDATE methods to implement the business logic
10. Use POJO classes to access the data from database
11. Use DBCursor and BasicDBObject or equivalent classes to access the data from database

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| Spring Data Reactive |
| MySQL |

#### Integration Layer

**Development instructions to this layer:**

1. In this micro layer, you will develop Spring Boot web application using Rest and Microservices.
2. Use REST API for exposing resources
3. Usage of HTTP GET/PUT/POST request method designators for the business methods which is to be exposed. Customize the request and response formats according to the requirement
4. Use appropriate RETURN CODES based on the service outcome
5. Extract query/form/header parameters from the input
6. Build a custom response based on the input
7. Use JSON/JAXB data types in the service request/response
8. Spring-boot-starter-data-rest API of Spring Boot bundle should be used
9. Use Spring Boot Main class - @SpringBootApplication for running the application
   * + 1. Develop individual Microservices for each functionality. Each of the Microservices need to comprise below functionality, which need to be developed.
     1. REST Controllers or Routing Functions
     2. Services or Handlers
     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 Microservices.
     5. Microservices should interact with corresponding DB tables or Databases it owns.
     6. Microservice need to interact with other Microservice
     7. Send Email Notifications to Users in Use Cases wherever appropriate.
     8. Usage of Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice
     9. Unit Testing code should be developed using JUnit /Mockito and perform Unit Testing

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| Spring Boot |
| REST, Microservices |
| GIT |
| Jenkins |
| Docker |

#### Agile/Devops Layer

**Development instructions to this layer:**

1. In this micro layer, you will configure Jenkins and create deployable for Docker. Use the application, which you may have developed in either of the microlayer:
   1. Spring Core and Spring Webflux Microlayer (using Spring 5 reactive programming)
   2. Spring Boot & ORM Microlayer (using Spring 5 reactive programming)

##### Configure Jenkins and Docker for the project

1. Import the project (as discussed above) in Spring Tool Suite and configure it locally to run it as Spring Boot App.
2. You may need to configure MySQL credentials and database name.
3. Execute the project locally and access the app at <http://localhost:portnumber>
4. Once, it is working fine in local development environment; do the following as a part of deploying the app to Docker as container:
   1. Push the app source in internal GIT server.
   2. Configure Jenkins locally to pull the source from internal GIT repository
   3. Jenkins should build the project and create the deployable (jar). It should run the unit tests created in ”Maven, GIT, Junit, Tomcat Micro Layer”
   4. Create a Docker file and docker-compose.yaml such that, when you run the following command:
      1. docker-compose up
      2. It should deploy the attached project at <http://localhost>
   5. Hints:
      1. Docker Compose file should:
         1. Create ngix container
         2. Create mysql container
         3. Configure attached “app” to run in docker containers

##### CI & CD

Make few changes in the project (source code)

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

If it running locally without errors, push the changes to the internal GIT repository.

Jenkins will automatically pull the code updates from internal GIT repo, build, and deploy the project with updated code.

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

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| Junit/Mockito |
| Git/GitLab |
| Jenkins |
| Docker |
| Maven |
| Jasmine |

#### Deployment Platforms

**Development instructions to this layer:**

1. In this micro layer, you will deploy the application in Tomcat server
2. Deploy the application by signing-in as manager in Apache Tomcat

|  |
| --- |
| **Framework(s)/SDK/Libraries** |
| Tomcat |

### Case Study Deliverables

1. Create a descriptive ‘READ ME’ document about the project; include the Steps to execute the project in STS or Eclipse and how to run/deploy the project. This should help the evaluators to set up your project in their environment for the execution. Detail about any database set up if required to evaluators
2. Upload your deliverables as one singe ZIP file into Moodle
3. If the ZIP file size is more than 50 MB, then discard all your JAR files/Node Modules for the upload. Make sure the instructions and process of including it is available in README file
4. ZIP file Naming Convention –> <AssocaiteID>\_<CourseCode>\_<Track>\_<Upload Date DDMMYYYY> Eg., 145865\_ATKTM167\_DOTNET\_26082019
5. Keep the final back up in <https://code.cognizant.com>
6. Please make sure that your code does not have any compilation/console errors while being deployed using Docker.
7. The final solution should consist of:
   1. Docker Image, which can be executed to deploy the project at <http://localhost:portnumber> or <http://ipaddress:portnumber>.
   2. Dump of command “git log”
   3. Jenkins report about number of times it pulled the code from GIT to create a production/minified version of UI solution.
   4. Docker File used to create Docker image from Jenkins.
   5. Working POM.xml for the project.
8. Docker Image to be executed to run the database, pre-populated with sample records.
9. Jenkins/build report and code coverage report from JaCoCo

**Before submitting the case study ensure you,**

1. Set up Jenkins on the localhost/cloud and integrate with GIT repository.
2. Setup Jenkins to create Docker image, which can deploy the final solution.
3. Save the Jenkins/build reports as a part of final deliverable.