## 

## 

## Engineering Review Manager

## Architecture & Design Document

## 

## 29.11.2017

## Version 1.0.0

## 

## 

## 

## 

## 

## **Table of Contents**

[**1. Introduction**](#_hezsepuj00v7) **3**

[1.1 Purpose](#_2w0qiukmd4sy) 3

[1.2 Overview of the system](#_y71v95k2syna) 4

[**2. System Architecture**](#_nx95spmowhw) **4**

[2.1 Architecture](#_wiwedvmulr0z) 4

[**3. Database Consideration**](#_xnwsuxtbfiiv) **5**

[3.1 ER Diagram](#_sv3p2cix2w6q) 5

[3.2 Design Decisions](#_jv8z04v0obkz) 6

[**4. Ballerina Service Endpoints**](#_9wybfdxr76h5) **7**

[4.1 Endpoints summary](#_ducytxbska0x) 7

[4.2 Endpoint response format](#_ctjy5xwun63e) 10

[**5. Config file format**](#_96ffhqrbglyw) **11**

[**6. Web application**](#_m9ojmcs4hm7b) **12**

[6.1 Libraries used](#_8iuf6yomm726) 12

[6.2 Process](#_i4oeg8recu5x) 12

[**7. SSO Enabling**](#_aw0hq28dc0vf) **15**

[**8. Deployment**](#_hrfocly65b8j) **16**

[**9. References**](#_b4e0woeewewv) **16**

## 

## 1. Introduction

### 1.1 Purpose

In order to examine and streamline the process of software development, WSO2 engineers go through many reviews such as User Story Review, Architecture/ Design Review, Code Review, and UX/UI Review simultaneously for tasks that have been completed and that are still under development. Currently the review notes are given in emails as replies to the review invitation by the participants of the review. When some interested parties want to view details of a particular review or someone wants to track the reviews that have been done for a specific product or a component, they need to query the emails manually which is the major drawback in this method of review notes submission. Also some of the required information might not be in the submitted review notes due to the use of poor templates. To overcome this difficulties, an application has to be built to submit and store the details properly for future requirements.

### 1.2 Overview of the system

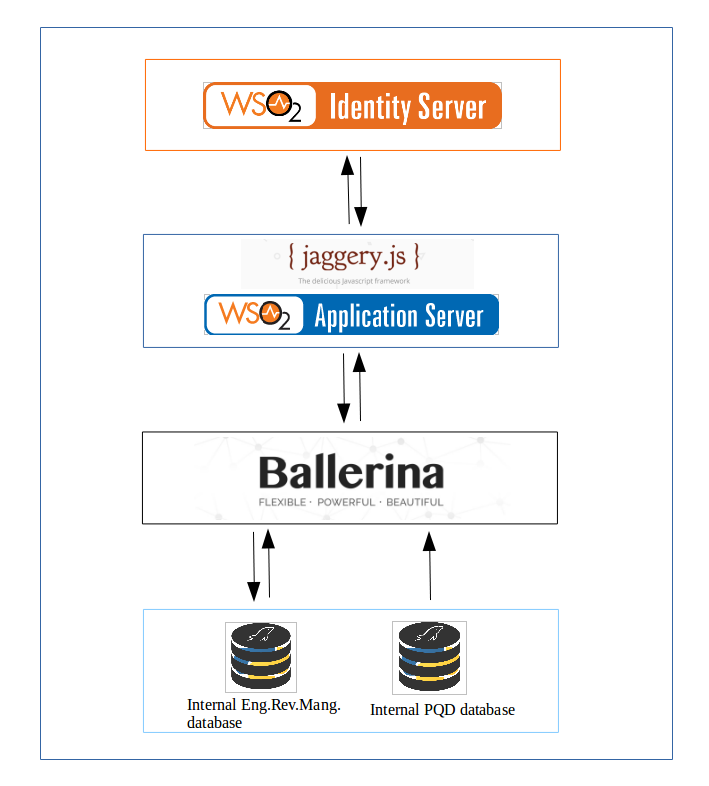
Engineering Review Manager application is designed and implemented to achieve the above purpose. This internal application can be used to submit a review with all the required information by one who participates in a review. A review that is submitted through this application is stored in a database and an email summarizing the review is sent to the relevant parties. These parties include the reporter(person who submits the review), contributors, participants, Eng.team and the relevant teams(optional).

The technologies used to implement the client side are jaggery.js[[1]](http://jaggeryjs.org/), JavaScript, HTML and CSS while those used to implement the server side is Ballerina v0.91[[2]](https://ballerinalang.org/). The jaggery application can be accessed via apps.wso2.com and it is hosted in WSO2 application server[[3]](https://wso2.com/products/application-server/). The login access will be provided by WSO2 Identity Server[[4]](https://wso2.com/identity-and-access-management).

Once the review details are submitted through the jaggery application, the form data will sent to the ballerina application which is hosted on a server. The ballerina service will retrieve the payload and record the data in the mysql database. Also an email will generated using the gmail connector of ballerina (available in ballerina 0.91) and will be sent to the relevant parties which are mentioned above.

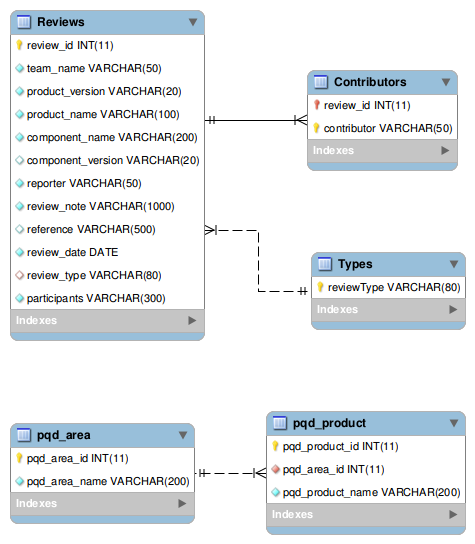
## 2. System Architecture

### 2.1 Architecture



## 3. Database Consideration

### 3.1 ER Diagram



### 3.2 Design Decisions

This application employs two separate databases of which one belongs to the WSO2 Product Quality Dashboard in order to hire WSO2 team details and WSO2 product details and the other is used solely to this application.

1. Pqd\_area and pqd\_product tables are taken from the product\_quality\_dashboard database because these tables are up to date with current status of the company. Others are from wso2\_engineering\_reviews\_database.
2. Types table is there to store the predefined review types at WSO2. Those are used to fill the type dropdown on the jaggery application.
3. Contributors table stores each contributor’s email address with the foreign key review\_id. This information kept in a separate table in order to avoid the redundancy.
4. Component details related to a review are stored in the Reviews table since this detail might not be recorded anywhere.

## 4. Ballerina Service Endpoints

### 4.1 Endpoints summary

Basepath - [https://](about:blank)domainaddress:port/internal/review-manager/v1.0

|  |  |  |
| --- | --- | --- |
| **Service Endpoint** | **Data Service Name** | **Description** |
| POST  /reviews | recordReview | Post the review details as a json payload in the following format.  { "Reporter": Reporter,  "Contributor":Contributor,  "Participants" :Participants,  "GroupEmails":GroupEmails,  "ReviewType":ReviewType,  "ReviewDate":ReviewDate,  "TeamId": TeamId,  "TeamName": TeamName,  "ProductName": ProductName,  "ProductVersion": ProductVersion,  "ComponentName": ComponentName,  "ComponentVersion": ComponentVersion,  "ReviewNotes": ReviewNotes,  "Reference": Reference  } |
| GET  /types | getAllTypes | Get the review types details stored in the database to populate the review types in jaggery app. |
| GET  /teams | getAllTeams | Get the WSO2 team names from database to populate the team names in jaggery app. |
| GET  /products/{teamId} | getProductsByTeam | Get the WSO2 product names under a team id from database to populate the product names under a selected team in jaggery app. |

Following endpoints are designed to get the review details, but have not been used in this application. (might be useful with the requirement of view or update a submitted review details through the app.)

|  |  |  |
| --- | --- | --- |
| **Service Endpoint** | **Data Service Name** | **Description** |
| GET  /reviews?dateFrom=2017-09-01&dateTo=2017-10-30 | getAllReviews | Get all review details in the database.  (Optionally a date range can be given for further filtering of the details.) |
| GET  /reviews/{reviewID} | getReviewById | Get details of the review having the given review id in path parameter. |
| GET  /reviews/types/{typeName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByType | Get details of all reviews which belong to a given type in the path parameter.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/teams/{teamName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByTeam | Get details of all reviews which belong to a given team name in the path parameter.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/teams/{teamName}/{typeName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByTeamAndType | Get details of all reviews which belong to a given type in the path parameter.  Optionally a date range and a contributor name can be given for further filtering of the details. |
| GET  /reviews/products/{productName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByProduct | Get details of all reviews belong to a given product name in the path parameter.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/products/{productName}/{typeName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByProductAndType | Get details of all reviews belong to a given product name and a type in the path parameters.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/components/{componentName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByComponent | Get details of all reviews belong to a given component name in the path parameter.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/components/{componentName}/{typeName}?dateFrom=2017-09-01&dateTo=2017-10-30&contributor=sajinie | getReviewsByComponentAndType | Get details of all reviews belong to a given component name and a type in the path parameters.  (Optionally a date range and a contributor name can be given for further filtering of the details.) |
| GET  /reviews/contributors/{contributor}?dateFrom=2017-09-01&dateTo=2017-10-30 | getReviewsByContributor | Get details of all reviews belong to a given contributor in the path parameters.  (Optionally a date range can be given for further filtering of the details.) |
| GET  /reviews/contributors/{contributor}/{typeName}?dateFrom=2017-09-01&dateTo=2017-10-30 | getReviewsByContributorAndType | Get details of all reviews belong to a given contributor and a type in the path parameters.  (Optionally a date range can be given for further filtering of the details.) |

### 4.2 Endpoint response format

All the above endpoints provide a json response in a same format in the response body. Each response json has an error field indicating whether there was any error produced in processing/retrieving the data.

When you are posting the form data to endpoint, it will reply with a json element called ‘message’ indicating a summary. If any error generates it will mention it in the message element.

Eg :

{

“error” : {

“Status” : true,

“msg” : “Sorry! Unable to save the review”

}

}

When a GET request for reviews is successfully executed, the data will come under another field called “reviews” as below. (likewise team, product and type details will come under “teams”, “products” and “types” fields respectively)

Eg :

{

“error” : {

“Status” : false,

“msg” : “”

},

“reviews” : [ ]

}

## 5. Config file format

In the Ballerina service side, a config.json file has to be placed in the same folder where balx file is hosted in order for the program to run. The config file includes the configuration details of two databases and gmail tokes.

Following is the format of the config file.

{ "sql" :{

"jdbcURL":"",

"username": "",

"password": "",

"PQD\_jdbcURL":"",

"PQD\_username": "",

"PQD\_password": ""

},

"gmail":{

"recipientEmail":"",

"userId":"",

"accessToken":"",

"refreshToken":"",

"clientId":"",

"clientSecret":""

}

}

Also in the jaggery application, constants.jag file is required to be placed in the root level and provide following parameters.

<%

var BALLERINA\_SERVICE\_BASE\_PATH = "";

var JWT\_HEADER\_NAME = "";

var CERTIFICATE\_PATH = "";

var SSO\_LOGIN\_URL = "https://apps.wso2.com";

%>

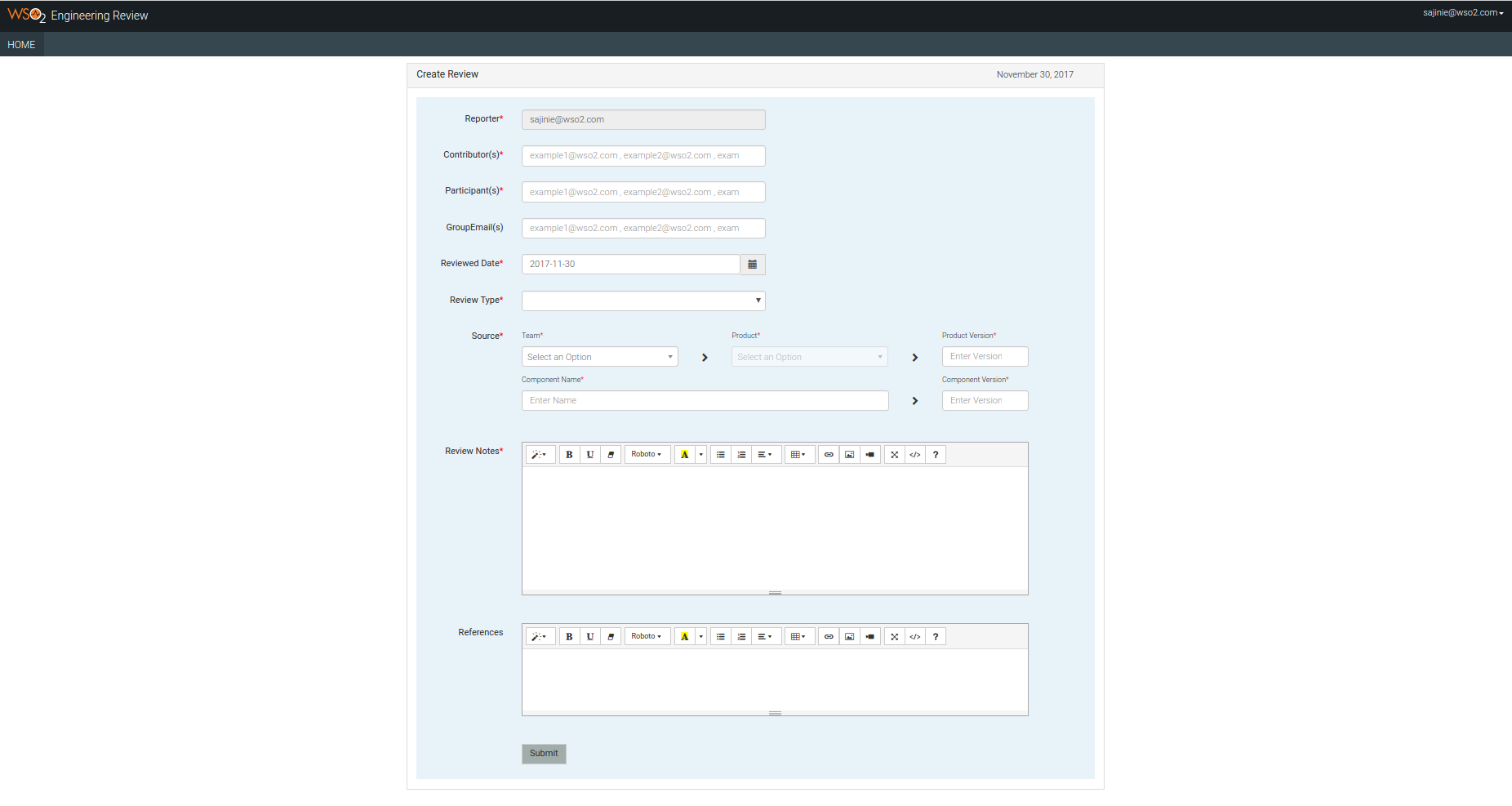
## 6. Web application

### 6.1 Libraries used

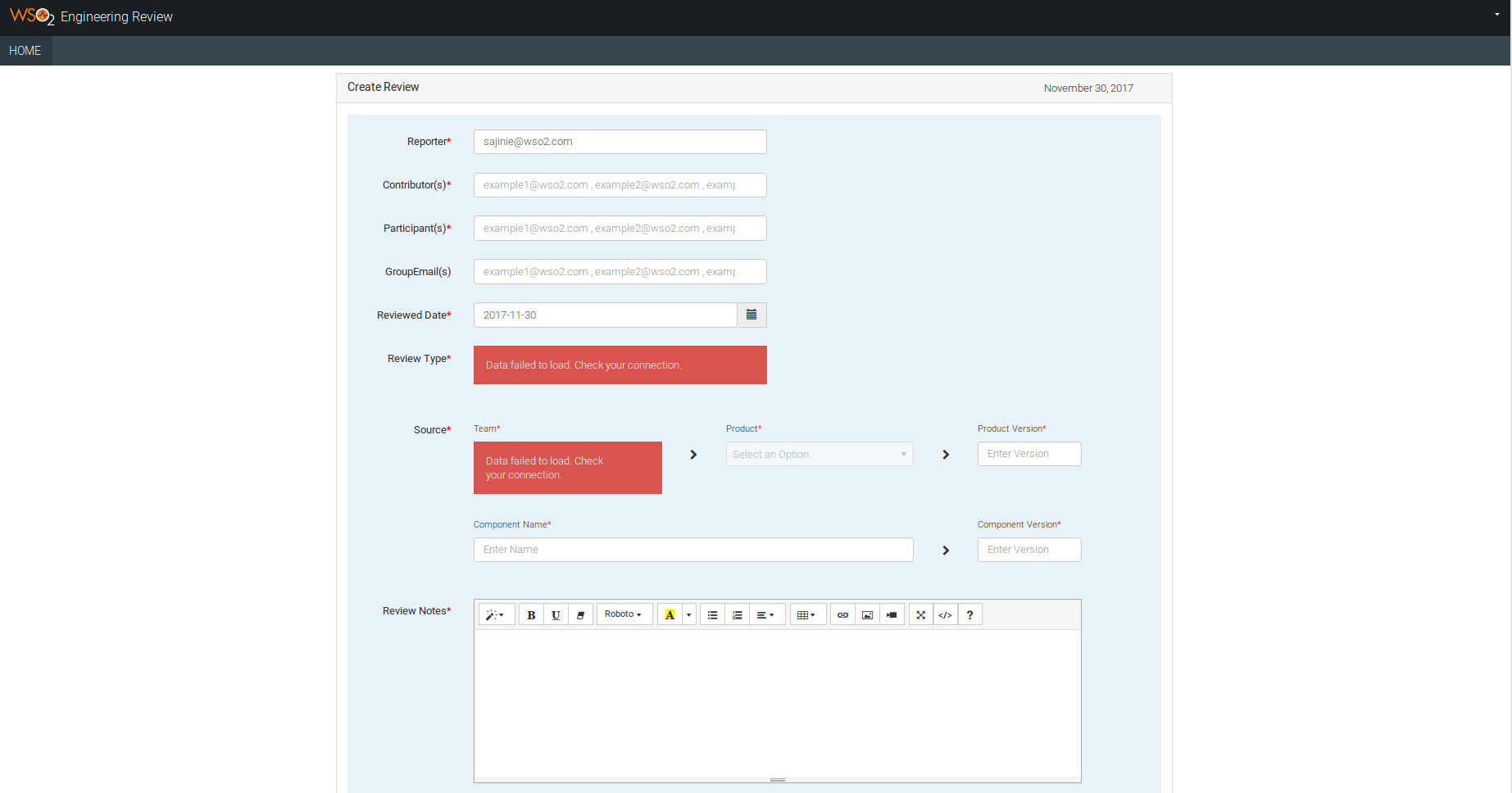
|  |  |  |
| --- | --- | --- |
| Library | Version | Description |
| WSO2 Theme[[5]](http://wso2-dev-ux.github.io/theme-wso2) | v2.0.0 | This is used to recreate the WSO2 theming to application.  Note: WSO2 theme itself uses dependencies on its own such as bootstrap 3.3.7, tinymce 4.2.3 etc., |
| Summernote[[6]](http://www.daterangepicker.com/) | v0.8.8 | This library is used to enable the WYSIWYG editor in the review form. |
| Daterangepicker[[7]](http://www.daterangepicker.com/) | v2 | This library is used to select the date range for engagement period. |

### 6.2 Process

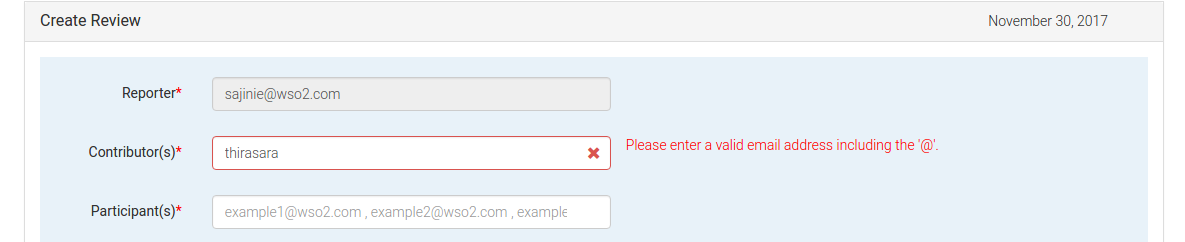
1. This is the main page that is used to submit a review.

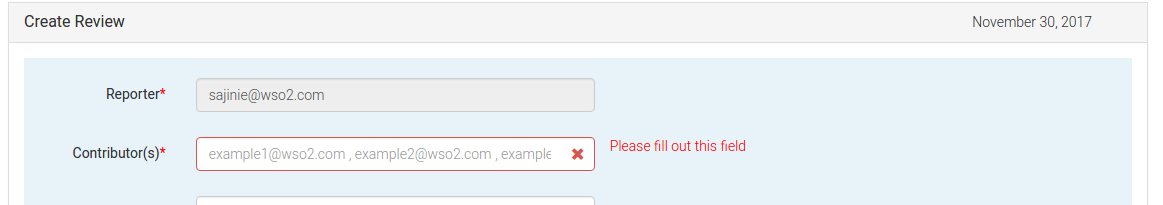


1. When some error occur due to poor connection to the backend or if the backend services are not working properly, these error messages will get displayed in places where the data is rendered.



1. The form is validated to have these validation messages.





1. If the form get submitted successfully, the home page will be directed to another page displaying this success message.



1. If any error is occurred while in the process of saving the review and generating the email, this message will get displayed.



## 7. SSO Enabling

In order to deploy the application in apps.wso2.com, the application needs to validate the jwt token that is sent in the header when the cloud is redirected to the application. The basic need in this application is to retrieve the jwt token from the header “x-jwt-assertion” and get the claim out of it using jwt parse package. It relatively simple to implement this mechanism in jaggery.

var jwtClientModule = require('../js/jwt-client.js');

var jwtClient = new jwtClientModule.JWTClient(request, JWT\_HEADER\_NAME, CERTIFICATE\_PATH);

jwtClient.init();

log.debug("JWT = " + jwtClient.jwt);

var emailAddress;

log.debug("Boolean : " + jwtClient.isJWTPresent());

log.debug("Certificate path : " + jwtClient.certificatePath);

if(jwtClient.isJWTPresent()){

try {

jwtClient.parse();

log.info("Verified the signature of the JWT.");

var is\_valid\_session = true;

var claims = jwtClient.getClaims();

emailAddress = claims.get("http://wso2.org/claims/emailaddress");

} catch(e){

log.error("Authentication failure. Something went wrong " + e);

}

} else{

log.info("User is not authenticated. Redirecting to apps.wso2.com...");

response.sendRedirect(SSO\_LOGIN\_URL);

}

## 8. Deployment

1. Host the ballerina service in a server with a valid port number.
2. When deploying the ballerina services create config.json file with the necessary fields mentioned in this document.
3. The SQL schema for application has to be put into a SQL server and the details of the server should be included in the config.json.
4. Deploy the jaggery application in the application server.
5. In jag/constants.jag file, point the baseURL variable to the server that hosts ballerina.
6. If needed, include the certificate of the public ip that the ballerina services are hosted, in the client-trust keystore file of application server.

## 9. References

[1] - <http://jaggeryjs.org/>

[2] - <https://ballerinalang.org/>

[3] - <https://wso2.com/products/application-server/>

[4] - <https://wso2.com/identity-and-access-management/>

[5] - [http://wso2-dev-ux.github.io/theme-wso2](http://wso2-dev-ux.github.io/theme-wso2/)/

[6] - <https://summernote.org/>

[7] - <http://www.daterangepicker.com/>