Project Specification – Version 2.0 (Final Version)

Release Date: Jun 26, 2020

Title:

DALServerlessLMS - A Multi-Cloud based Serverless Learning Management System

Objective:

The primary objective of this project is to build **cloud plumbing system**, where an application will be designed using serverless technologies to process data (more specifically large-scale data). This is like building a water plumbing system, which is built once with plenty of interconnected pieces, and used many times. It does not require any specialist for the operation. In this project, you will be building a "cloud data plumbing system", which could be used by many clients to process their data. You will use different backend services, and simple front-end application to build the system.

Explanation:

This project is introduced in the Serverless Data Processing Course (CSCI 5410) to fulfill the course requirement. This is a group project (weightage 40%), and each group is required to perform specific tasks within a given time frame. There are project constraints, and scope, which must be followed by each team. The project will follow an agile model, where each team should welcome change in the requirements. However, considering the time and resource restriction, requirement changes at the last phase of the development will not be addressed.

Hypothetical Scenario:

DALSoft5410 is building a serverless Learning Management System (LMS) using multi-cloud deployment model, and backend-as-a-service (BaaS). The LMS - "DALServerlessLMS", should provide customization feature, and additional services for instructors, and limited services to students. The LMS should provide an online virtual assistance, which can quickly answer the queries of users, and in addition, it should provide a chat functionality between the registered users.

DALSoft5410 has selected serverless application to minimize the development and project running cost. The company has identified two cloud platforms - AWS, and GCP to build, test, and deploy their application. They have decided to follow the official documentations of AWS and GCP to build the different pieces.

If they select server-oriented architecture, then they need to manage and configure the backend service, which they cannot do due to their resource limitations. Therefore, serverless is the only solution they found at this point. They have obtained two types of accounts from AWS, and one account from GCP, which they can use for building, testing, and deploying their application.

Since they are going to follow agile method, they can build, test, and change each components of the project whenever there is a change in the requirements. One important aspect of this project is building a serverless portable application, which can be used by multiple clients without making changes to the system. E.g. University "X" uses the application to perform file management, searching, chat, and analysis. Another university "Y" can use the same application by creating their accounts in the system and start using the system without making any changes. The setup for university "X", and "Y" should not take more than few minutes. In addition, the application should provide certain options to perform user level configuration change. The main module of the application includes – user management, file management, data processing, analytics, instant messaging, virtual assistance etc.

Important Dates:

V2.0 Date Given: Jun 26, 2020

Final Demonstration: Week of July 13, July 20, and July 27 - For your team's time slot, a survey will be conducted on the 1st week of July.

Report Submission Deadline:

- Design Document - ~~June 26, 2020 at 11:59 pm~~ Jul 8, 2020 at 11:59 pm
- Final Report - July 31, 2020 at 11:59 pm

Note: For late submission of any report, there will be a 10% deduction per day. (exception – Student Declaration of Absence for medical reasons).

DALServerlessLMS Application Overview:

In this application, you are required to build a simple front-end, which will interact with Backend-as-a-Service (BaaS). Some specific requirements are identified, which should be considered for design and implementation. ****You must use Severless architecture**

Module Name	Functional Requirements	Technical Specification	Expectations	Use Case/ Extras/ Test Cases
User Management Module	The system should accept registration request from multiple people and should validate the registration process. (e.g. email validation, password validation, checking required fields etc.) Once the registration details are accepted some information are stored in AWS RDS, and some are stored in GCP Firestore.	User must be able to enter details through a browser. The details are validated and processed at GCP using cloud function, and other third-party services. Try to avoid "re-inventing wheel". If a service is there from GCP or standard third-party then you can use it. This is a multi-cloud environment, therefore, once registration details are processed, except question answer, all the UserIDs/Passwords, and other details should be kept in a database in AWS using database-as-a-service (RDS). **Please do not use EC2, ECS, VMs etc.** The UserID, and security question and answer provided by the users are stored in GCP firestore	Design and orchestration of GCP cloud functions, AWS Lambdas, and creation of APIs for multi-cloud interaction. Your system should be able to register and process 2 to 3 users simultaneously from the web interface. i.e. multiple instances of cloud functions should work for registration (each instance for registering one applicant) Details such as institute name could be added to create the encapsulation of organization. E.g. "Saurabh" related to "DAL", and "Saurabh" related to "SMU" could be 2 different users. **In addition, users of one organization should see each other (if online)**	While working on the project design document, you should consider use case(s), and in addition, compose the test cases. To produce a high-quality design document, screenshots of prototypes (if any) of your registration form, data model could also be added along with the serverless architecture During implementation, you should perform unit testing of this module

Module Name	**Functional Requirements**	**Restriction/ Flexibility**	**Expectations**	**Extras/ Test/ Use Case**
Authentication Module	Your system should use multi-factor authentication (e.g. UserID/Password and Answering the Security question) using GCP and AWS. This module is like registration module and requires multi-cloud model. Once a user tries to login through the front-end login box/form, 1st factor authentication check is done at AWS, 2nd factor authentication check is done at GCP	A lambda function works with service like cognito to perform 1st factor authentication validation Once it is done, an API is called, which works with GCP cloud function, and firestore to perform 2nd factor (security question) authentication validation **Your group can reverse the operation of GCP and AWS. E.g. using GCP Function and Firebase for 1st factor validation Using Lambda, and RDS for 2nd factor authentication validation	You should explore GCP and AWS functions (Lambda, and Cloud functions), create APIs, work with end points, and avoid server-oriented-architectures	While working on the project design document, you should consider use case(s), and in addition, compose the test cases. During implementation, you should perform unit testing of this module
Module Name	**Functional Requirements**	**Restriction/ Technical Requirements**	**Expectations**	**Extras/ Test/ Use Case**
Online Support Module	Bots should respond to queries e.g. A user who has logged in can perform some operations in the system. A bot can help the user in navigation and searching. "Saurabh" has logged in and asked chatbot "VirtualHelp" **Example Use Case 1:** Saurabh - "hey, I need some help?" – VirtualHelp –"What help?" Saurabh – "I want to know how to upload data" VirtualHelp – "Do you want to process or analyze?"	AWS Lex with Lambda is required for the application. Example 1 does not require any advanced operation or connecting to database. It is simple pre-defined utterances, prompt, slots, fulfillment Example 2 requires using Lambda, and querying the RDS, which has information on Login Status	A good workflow, and well-designed utterances, slots etc. You should explore the service	While working on the project design document, you should consider use case(s), and in addition, compose the test cases. During implementation, you should perform unit testing of this module

	Saurabh – "process" VirtualHelp - "go to top left, you will find a button for uploading and processing" **Example Use Case 2:** Saurabh - "hey, I need some help?" – VirtualHelp –"What help?" Saurabh – "I want to know, who else is online in my organization" VirtualHelp –"Can you confirm your organization name for security" Saurabh – "DAL" VirtualHelp – "Let me query the database for you"			
Module Name	**Functional Requirements**	**Technical Requirements**	**Expectations**	**Extras/ Test/ Use Case**
Chat Module	Group members should be able to communicate like an instant messaging engine. Store the chat messages as JSON structure in GCP Cloud Storage	This module requires you to use GCP pub/sub to build the chat module, and keep each chat session as a JSON file in the GCP Cloud Storage	This step expected because it will be used in Analysis	While working on the project design document, you should consider use case(s), and in addition, compose the test cases. During implementation, you should perform unit testing of this module
Module Name	**Functional Requirements**	**Technical Requirements**	**Expectations**	**Extras/ Test/ Use Case**
Data Processing	A container-based application to extract named entities (uppercase entities) from uploaded files and build a word cloud	Create a docker container for the job and use Cloud Run to run your container. Remove stop words, and then process (find entities with 1st or all letters as uppercase) uploaded files. For testing purpose, you can upload the news	Please do not worry about the accuracy of named entity detection. One single word cloud should be built considering detected named entities from all the files.	While working on the project design document, you should consider composing the test cases. During implementation, you should perform unit testing of this module

		files given for assignment 3. You can use any service (including 3rd party) for generating the word cloud		
Module Name	**Functional Requirements**	**Technical Requirements**	**Expectations**	**Extras/ Test/ Use Case**
Analysis 1 Machine Learning module is renmaned	Clustering of uploaded Files based on similarity in the titles (If you do not have any example files, please let me know)	You need to use SageMaker, and KMeans Algorithm to find if files are related. You need to consider only the title of the files (Reuter News Article may work) for this unsupervised learning Files should be uploaded on S3 bucket for analysis.	Please do not worry about the accuracy of the analysis I am interested in the plumbing	While working on the project design document, you should consider composing the test cases. During implementation, you should perform unit testing of this module
Analysis 2 Newly Added	Sentiment Analysis of user chat messages. Just tagging the sentiment is needed	Ingest the Chat module JSON files from Cloud Storage to S3 bucket. Trigger Lambda when all files are available. Use Amazon Comprehend to tag the chat messages and update the JSON files. Store the tagged JSPN chat messages in another S3 bucket		
Module Name	**Functional Requirements**	**Technical Requirements**	**Expectations**	**Extras/ Test/ Use Case**
Web Application Building and hosting	Building a front-end application using suitable framework, and calling backend services	Just host the light-weight web application on AWS or Localhost	Majority of the operation must be done in a serverless manner.	A prototype or actual design could be added as part of the design document. During implementation, you should perform testing of this module with the testing of the entire system
Module Name	**Functional Requirements**			
Documentation	This project requires extensive and systematic documentation. Every team meeting must be logged with dates, and added as part of design document, and final report.			

Short Term Expectations and Deliverables:

Working on Design Document

a. Build the Serverless Architectures for individual use cases, and modules
b. Build one complete Serverless Architecture highlighting all the modules and workflows.
c. Explain how you are going to implement the modules.
d. Find Challenges
e. How are you performing knowledge transfer?
f. Log and evidence of regular meetings with team members
g. Document your questions, and concerns

Final Deliverables:

1. Documents:
   a. Feasibility Report – 2%
   b. Design Document – 8%
   c. Final Report - 10%

2. Presentation:
   a. Pre-recorded presentation (maximum 30 min) on background study, team contribution, and test case. {You can use zoom to record your Team's presentation and upload the audio/video, and the presentation scripts (if any) to specific Brightspace submission folder}
   b. Pre-recorded presentation (maximum 30 min) on demonstrating the working model, test results, and analysis {You can use zoom to record your Team's presentation and upload the audio/video, and the presentation scripts (if any) to specific Brightspace submission folder}
   c. Synchronous session with Instructor, Tas (maximum 30 min) for Q&A

Communication:

1. Office 365 Teams Channel for Project: Project Discussion Channel
   a. Use @mention to communicate with your group/ TA/ Instructor
2. For email communication
   a. Cc your team members, and TAs

Note: The requirements might change, and your team should welcome any changes in the requirements.