



---

# DALServerlessLMS

FINAL REPORT

---



**Submitted By:**

**Group 13**

Kethan Srinivas Dasari [B00842485]

Jigar Makwana [B00842568]

Devarshi Pandya [B00840003]

Krutin Trivedi [B00843516]

## Contents

1. Introduction.....	6
1.1 The Problem .....	6
1.2 The Solution .....	6
2. DALServerlessLMS modules .....	7
2.1 User Management Module .....	7
2.1.1 Serverless Architecture.....	7
2.1.2 Prototypes .....	8
2.1.3 Use Cases.....	9
2.1.3.1 Use Case: Registration.....	9
2.1.3.1.1 Test Cases .....	9
2.2 Authentication Module.....	10
2.2.1 Serverless Architecture.....	10
2.2.2 Prototypes .....	11
2.2.3 Use Cases.....	14
2.2.3.1 Use Case: Login.....	14
2.2.3.1.1 Test Cases .....	14
2.3 Online Support Module .....	15
2.3.1 Serverless Architecture.....	15
2.3.2 Prototypes .....	16
2.3.3 Use Cases.....	16
2.3.3.1 Use Case 1: Simple query without Database connection .....	16
2.3.3.1.1 Test Cases .....	16
2.3.3.2 Use Case 2: Query requires using Lambda and RDS .....	17
2.3.3.2.1 Test Cases .....	17
2.4 Chat Module.....	18
2.4.1 Serverless Architecture.....	18
2.4.2 Prototypes .....	19
2.4.3 Use Cases.....	19
2.4.3.1.1 Test Cases .....	19
2.5 Data Processing .....	20
2.5.1 Serverless Architecture.....	20

2.5.2 Prototypes .....	20
2.5.3 Use Cases.....	20
2.5.3.1 Use Case 1: Build the word cloud .....	20
2.5.3.1.1 Test Cases .....	21
2.6 Analysis 1 .....	21
2.6.1 Serverless Architecture .....	21
2.6.2 Prototypes .....	22
2.6.3 Use Cases.....	22
2.6.3.1 Use Case 1: Upload all the document.....	22
2.6.3.1.1 Test Cases .....	23
2.7 Analysis 2.....	23
2.7.1 Serverless Architecture .....	23
2.7.2 Prototypes .....	23
2.7.3 Use Cases.....	24
2.7.3.1 Use Case 1: Performing Sentiment Analysis.....	24
2.7.3.1.1 Test Cases .....	24
2.8 Web Application Building and hosting .....	24
3. Overall Serverless Architecture .....	26
4. Workflow .....	27
4.1 User Management Module .....	28
4.2 Authentication Module.....	29
4.3 Online Support Module.....	30
4.4 Chat Module.....	31
4.5 Data Processing .....	32
4.6 Analysis 1 .....	33
4.7 Analysis 2.....	34
5. Testing.....	35
Screenshots of testing invalid data .....	35
6. Repository and Deployment Links .....	36
6.1 GitHub Repository Link: <a href="https://github.com/JigarMakwana/CSCI_5410_Project_Group13">https://github.com/JigarMakwana/CSCI_5410_Project_Group13</a> .....	36
6.2 Deployment Link: .....	36
<a href="https://deployed-branch.d3jgqtjoceto49b.amplifyapp.com/">https://deployed-branch.d3jgqtjoceto49b.amplifyapp.com/</a> .....	36

7. Task Achievement .....	36
8. Challenges faced .....	36
9. Methods of knowledge transfer .....	37
10. Team .....	37
Team Lead:.....	37
Team Members:.....	37
Devarshi Pandya .....	37
Krutin Trivedi .....	37
Jigar Makwana.....	38
Kethan Dasari .....	38
Appendix.....	39
A.1 Minutes of Meeting .....	39
A.2 MS Teams logs.....	41
References.....	44

## Table of Figures

<i>Figure 1 High-level Architecture of the DALServerlessLMS .....</i>	<i>6</i>
<i>Figure 2 Overall architecture of the User Management System .....</i>	<i>8</i>
<i>Figure 3 Sign Up Page of DALServerlessLMS.....</i>	<i>9</i>
<i>Figure 4: Overall architecture of the Authentication System .....</i>	<i>11</i>
<i>Figure 5 Login Page of DALServerlessLMS .....</i>	<i>12</i>
<i>Figure 6 Dashboard of the DalServerlessLMS.....</i>	<i>13</i>
<i>Figure 7: Online Support module architecture .....</i>	<i>15</i>
<i>Figure 8: Online Support module architecture .....</i>	<i>16</i>
<i>Figure 9: Chat module architecture .....</i>	<i>18</i>
<i>Figure 10: Chat module prototype .....</i>	<i>19</i>
<i>Figure 11: Data Processing architecture .....</i>	<i>20</i>
<i>Figure 12: Data Processing prototype .....</i>	<i>20</i>
<i>Figure 13 Serverless Architecture of Analysis 1.....</i>	<i>21</i>
<i>Figure 14 Prototype for uploading documents.....</i>	<i>22</i>
<i>Figure 15 Serverless Architecture for Analysis 2 .....</i>	<i>23</i>
<i>Figure 16: Deployment Architecture .....</i>	<i>24</i>
<i>Figure 17: Deployment on AWS Amplify.....</i>	<i>25</i>
<i>Figure 18 Overall Architecture .....</i>	<i>26</i>
<i>Figure 19: Legend.....</i>	<i>27</i>
<i>Figure 20 User Management Workflow .....</i>	<i>28</i>
<i>Figure 21 Workflow of User Management Module .....</i>	<i>28</i>
<i>Figure 22 Authentication Workflow .....</i>	<i>29</i>

<i>Figure 23 Workflow of Online Support Module .....</i>	<i>30</i>
<i>Figure 24 Workflow of Chat Module.....</i>	<i>31</i>
<i>Figure 25 Workflow of Data Processing .....</i>	<i>32</i>
<i>Figure 26 Workflow of Analysis 1 .....</i>	<i>33</i>
<i>Figure 27 Workflow of Analysis 2 .....</i>	<i>34</i>
<i>Figure 28 Password validation .....</i>	<i>35</i>
<i>Figure 29 Email Validation.....</i>	<i>35</i>
<i>Figure 30 Authentication failure .....</i>	<i>36</i>

# 1. Introduction

---

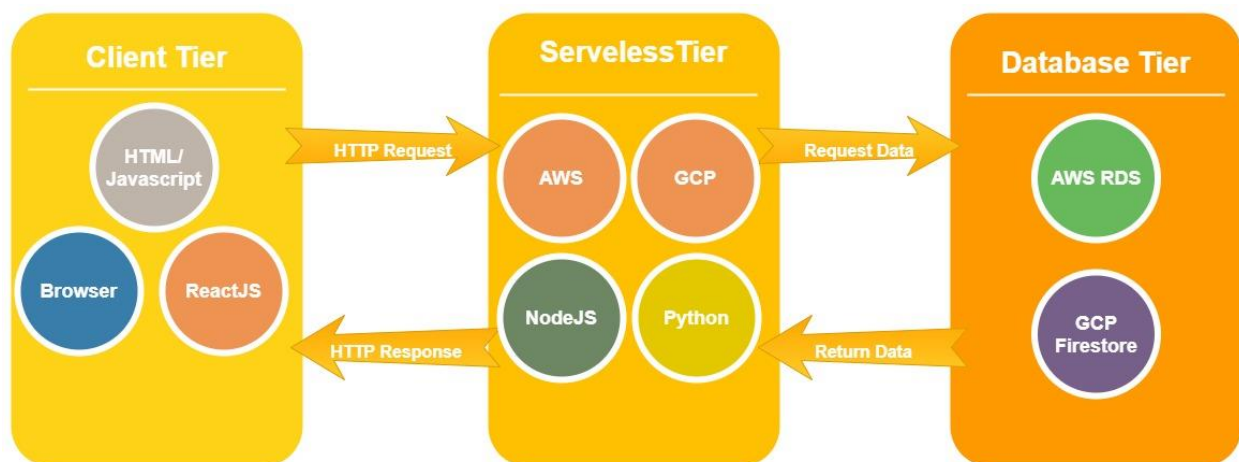
## 1.1 The Problem

As stipulated in Project Specification\_V2.0 [1], the project requirement is to build a serverless Learning Management System (LMS) called DALServerlessLMS which is a cloud plumbing system. The system uses various cloud technologies to process the data. The system is comprised of 8 interconnected modules and each module has constraints on using the different cloud technologies and services while developing. On a high level, the system is consisting of frontend, serverless backend, and the cloud database.

## 1.2 The Solution

We have solved the problem by following the standard Software Development Life Cycle where the first phase was Requirement Analysis where we studied the requirement of the problem and feasibility of the project. After the requirement analysis and feasibility analysis, we thoroughly researched the various cloud technologies to build a multi-cloud serverless system that uses backend-as-a-service (BaaS) from different cloud platforms. As the conclusion of the research, in the design phase, we made a design document for the DALServerlessLMS. In the Implementation phase of the project, we developed the individual modules and in the end, we integrated all the modules. In the Testing phase, the system is thoroughly tested where we found some bugs. We fixed those bugs and again tested the system until the system performed as expected. In the Deployment phase, we deployed the DALServerlessLMS using AWS Amplify.

Figure 1 shows the high-level architecture of the DALServerlessLMS application. We followed the 3-Tier Architecture to build the system. The first tier is consists of frontend which was mainly developed using HTML and ReactJS [2] library. The second tier is Serverless backend where we used various cloud technologies from AWS [3], and GCP [4] to build, test, and deploy the application. The business logic of the application has been mostly written in Python and NodeJS. The third tier of the system, the database tier, comprised of AWS RDS and GCP Firestore.



*Figure 1 High-level Architecture of the DALServerlessLMS*

To develop the backend system various services of AWS and GCP have been utilized and connected to build the different modules of the system. The official documentation of AWS and GCP is followed to build the different pieces. The main features of the application include – User Management, File Management, Data Processing, Analytics, Instant Messaging, and Virtual Assistance, For example, the Online Virtual Assistance quickly answers the queries of students and instructors. The Chat functionality allows the registered and online users to chat with other online users from the same institute.

We have followed the Agile methodology for developing the DALServerlessLMS. We conducted scrum meetings using MS Teams. We also recorded the Minutes of Meetings for the meetings where the potential action plans were made and decisions were taken.

## 2. DALServerlessLMS modules

---

### 2.1 User Management Module

#### 2.1.1 Serverless Architecture

Figure 2 demonstrates the serverless architecture of the User Management module of the DALServerlessLMS application. The User Management leverages the serverless technologies: Google Cloud Function, Google Firestore, AWS API Gateway, AWS Lambda, AWS RDS, and AWS Cognito.

As shown in Figure 3, once the user enters the registration data and clicks the Sign-Up button, an HTTP API call has been made to the GCP Cloud Function using Axios from ReactJS frontend. The Cloud Function validates the registration data such as email addresses. If the validation is successful; the Amazon Lambda Function is called using Amazon API Gateway. This Lambda function stores the user's Username, Email, Password, and Institute Name in Amazon RDS MySQL. Further, this Lambda function, adds the user's Username, Email and Password in Amazon Cognito User Pool. Once these two operations are done at AWS side Lambda sends the success response to Google Cloud Function. Then, Google Cloud Function adds the user the Email, Security Question, and Security Answer in Google Cloud Firestore and sends the success response to the frontend. If the validation of the data is not successful appropriate error message is shown to the user at the frontend. Some frontend data validation is also performed while registration.

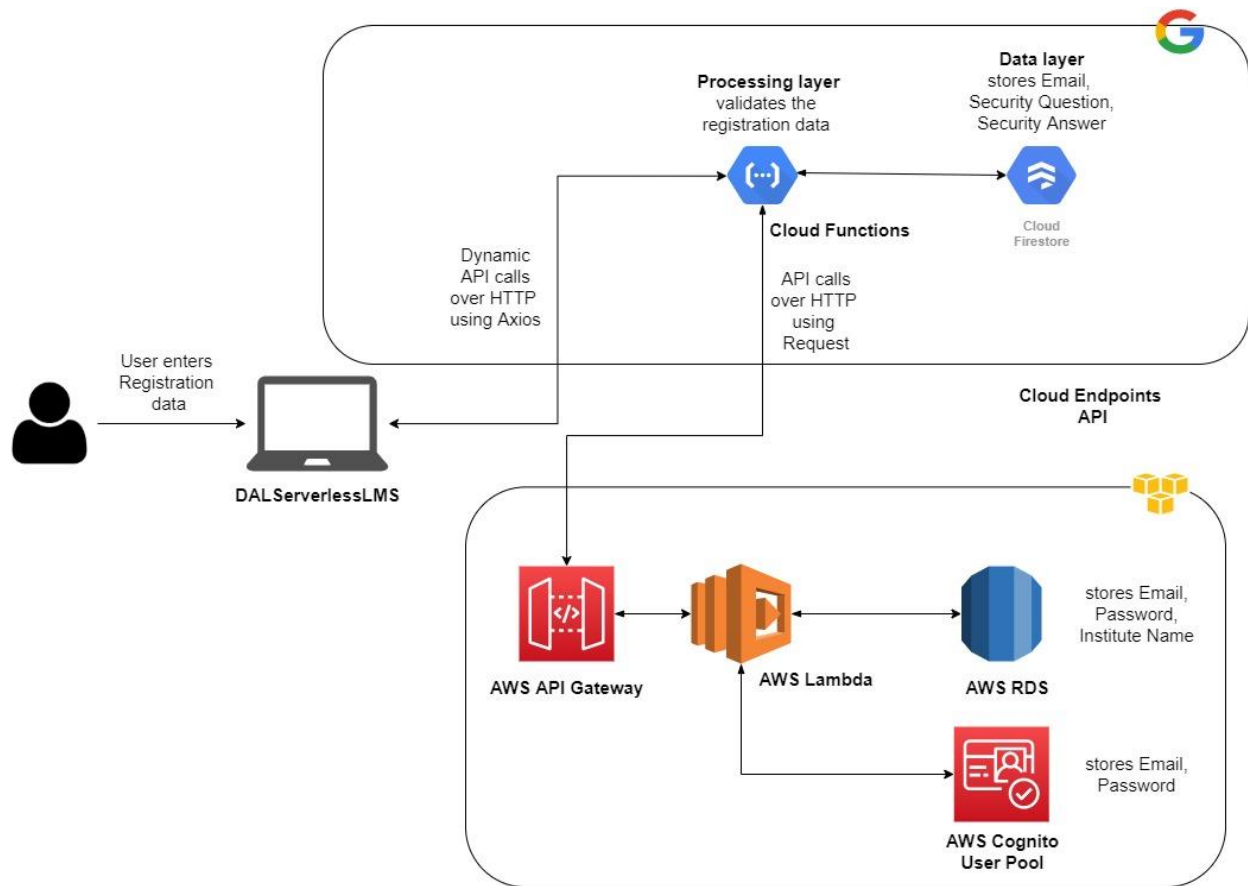



Figure 2 Overall architecture of the User Management System

### 2.1.2 Prototypes

Figure 3 demonstrates the high-fidelity prototypes of the “Sign Up” page of the DALServerlessLMS application.




 DALServerlessLMS

[Log in](#) [Sign up](#)

# stacklearner

STACK UP YOUR  
CODING SKILLS



## Get Started

**Username**

**Email**

**Password**

**Institute Name**





**Security Question**

**Security Answer**

[Sign Up](#)

By signing up, you agree to DALServerlessLMS's [Terms of Service & Privacy Policy](#).

Or, Sign Up with

[Chat](#)

*Figure 3 Sign Up Page of DALServerlessLMS*

### 2.1.3 Use Cases

#### 2.1.3.1 Use Case: Registration

Figure 3 shows the landing page of 'DALServerlessLMS' which has a section for the Sign-Up. For Sign Up, the user provides a registered Username, Email, Password, Institute Name, Security Question, and Security Answer

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** A new user
- **Goal:** To create an account so you can use this application in the future
- **Task:** Sign up as a user
- **Defining our user persona:** A new user who is interested in signing up with the application.
- **Defining our application:** The application would allow users to create a new account.

#### 2.1.3.1.1 Test Cases

1. User visits 'DALServerlessLMS' landing page [user action]

2. The user enters their preferred username, email id, password, Institution name, security question, and security answer [user action]
3. User clicks on 'Sign Up for free' button [user action]
  - 3.1 System displays a message letting the user know their email is not valid [system action]
  - 3.2.1 User enters a valid email address to Sign up [user action]
  - 3.2.2 User enters a new password [user action]
  - 3.2.3 User clicks on 'Sign Up for free' button [user action]
  - 3.2 System displays a message letting the user know their email has already been used [system action]
  - 3.3.1 User enters a new email address to Sign up [user action]
  - 3.3.2 User enters a new password [user action]
  - 3.3.3 User clicks on 'Sign Up for free' button [user action]
  - 3.3 System displays a message letting the user know their preferred password does not meet the minimum 6-character length [system action]
  - 3.4.1 User enters a new password [user action]
  - 3.4.3 User clicks on 'Sign Up for free' button [user action]
4. The system validates the registration information [system action]
5. The system displays a registration confirmation message to the user [system action]

## 2.2 Authentication Module

### 2.2.1 Serverless Architecture

Figure 4 demonstrates the serverless architecture of the Authentication module of the DALServerlessLMS application. The Authentication System leverages the serverless technologies: AWS API Gateway, AWS Lambda, AWS RDS, and AWS Cognito, Google Cloud Function, and Google Firestore.

The Authentication System implements two-factor authentication. The 1<sup>st</sup>-factor authentication is done at the AWS side while the 2<sup>nd</sup>-factor authentication is done at GCP. The 1<sup>st</sup>-factor authentication includes user email address and password verification while the 2<sup>nd</sup>-factor authentication includes verification of security question and security answer.

When the user enters the login data and clicks the login button (Figure 5), the Amazon Lambda Function is called using Amazon API Gateway from the frontend using Axios. The Lambda Function calls a sub-function that performs 1<sup>st</sup>-factor authentication by verifying the user's email address and password using AWS Cognito. The Lambda Function further calls a Google Cloud

Function which performs the 2<sup>nd</sup>-factor authentication. The Google Cloud Function retrieves the user's data from Google Firestore and verifies with the user input. The Google Cloud Function sends success or failure response to the Amazon Lambda Function. Further, Lambda Function sends the response to the frontend. If both the authentications are successful then the user is redirected to the dashboard of the DALServerlessLMS. If one of the authentications is failed the appropriate error message is shown to the user on frontend and the user is not redirected to the dashboard of the DALServerlessLMS.

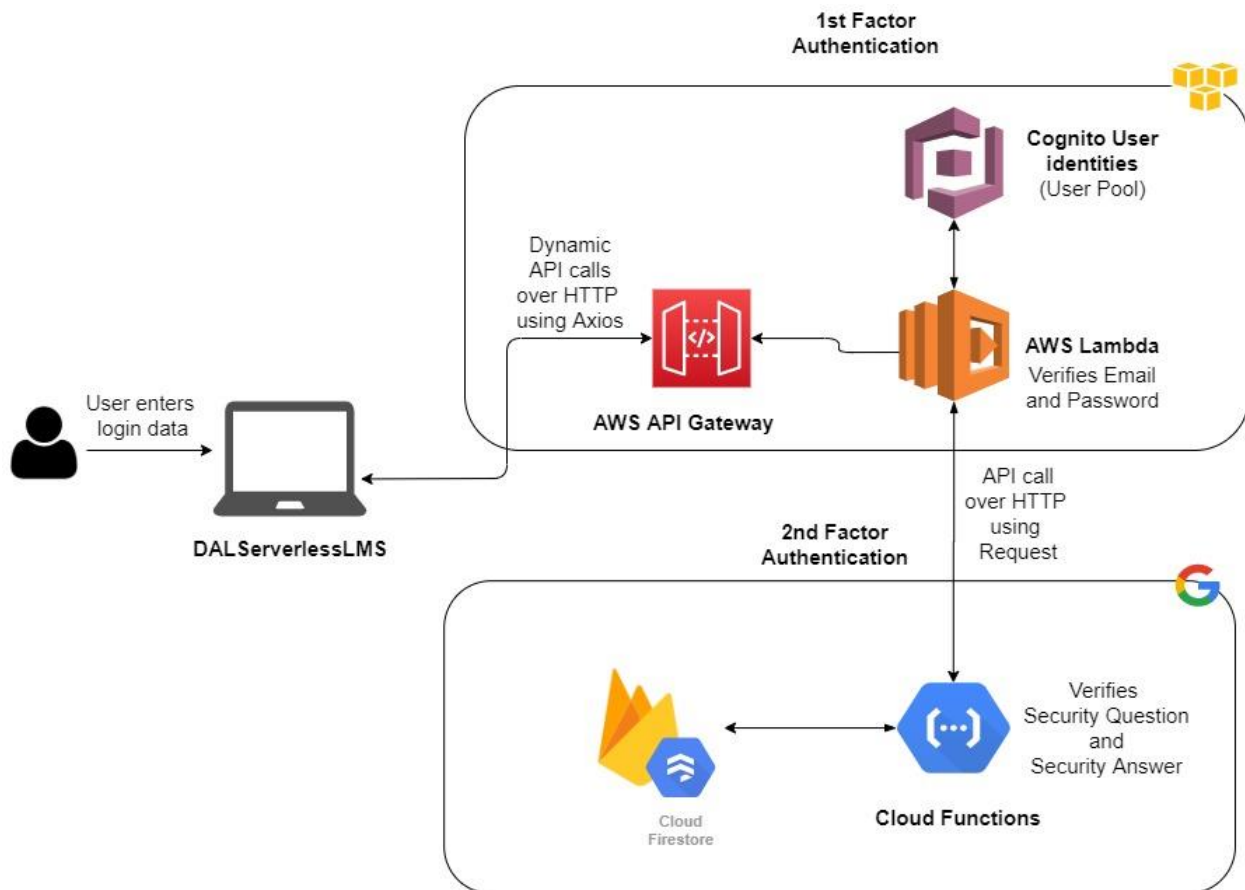



Figure 4: Overall architecture of the Authentication System

### 2.2.2 Prototypes

Figure 5 shows the high-fidelity prototypes of the “Login” page of the DALServerlessLMS application.

 DALServerlessLMS

[Log in](#) [Sign up](#)

### Log In to DALServerlessLMS

Email

Password





Security Question

Security Answer

[Forgot password?](#)

[Log in](#)

Login with another account

Chat

*Figure 5 Login Page of DALServerlessLMS*

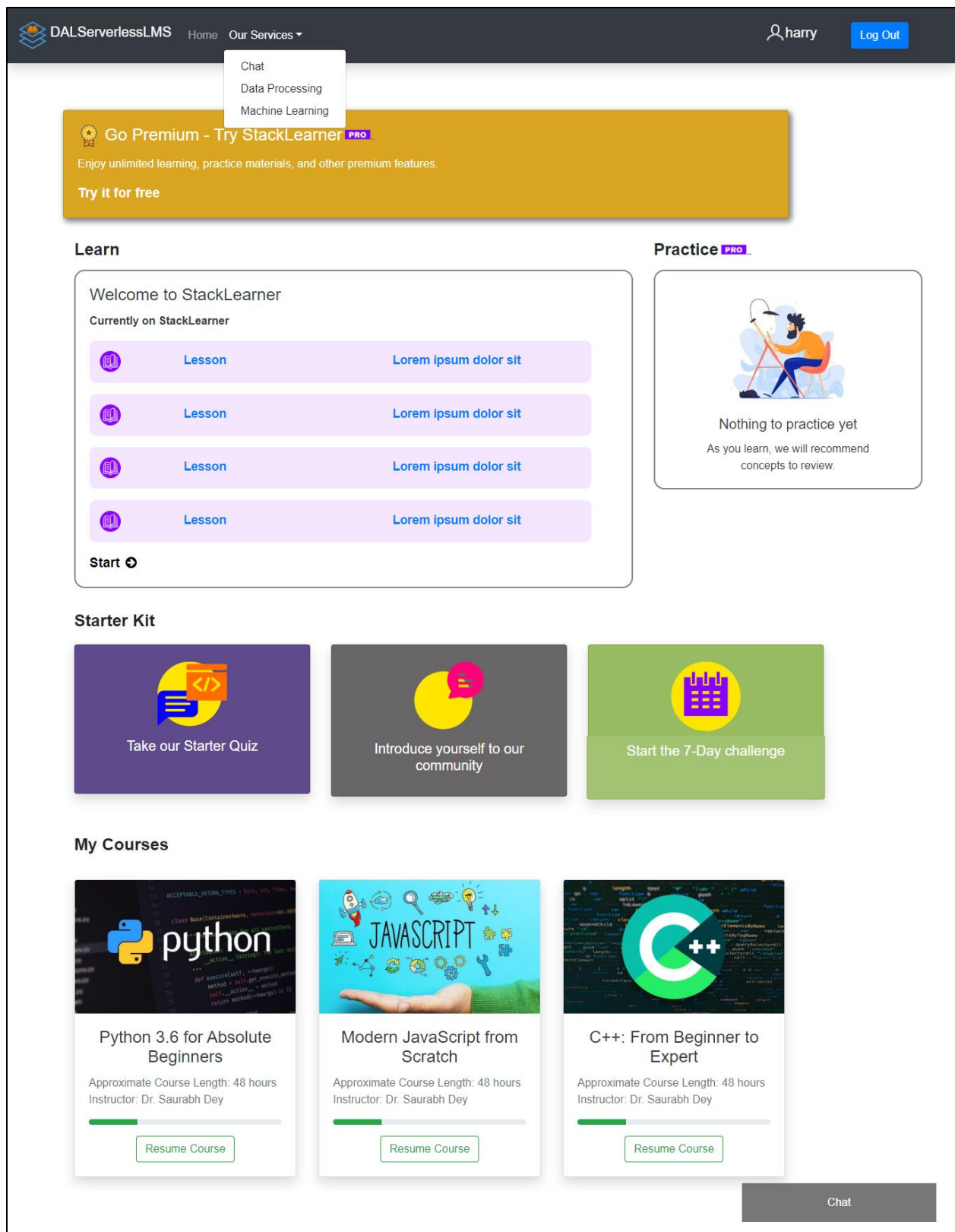


Figure 6 Dashboard of the DalServerlessLMS

### 2.2.3 Use Cases

#### 2.2.3.1 Use Case: Login

Figure 5 shows the login page of the 'DALServerlessLMS'. For login, the user provides a registered Email address, Password, Security Question, and Security Answer. On successful login, as shown in figure 6, the user is redirected to the dashboard page.

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** A registered user
- **Goal:** To login into your account to explore the application features
- **Task:** Login as a user
- **Defining our user persona:** A new user who has just created an account and is excited to explore the features of the application.
- **Defining our application:** The application would allow users to log in to their accounts and give access to all the features of a registered user.

##### 2.2.3.1.1 Test Cases

1. User is on the login page [user action]

2. The user enters their Email address, Password, Security Question, and Security Answer [user action]

3. User clicks on 'Login' button [user action]

3.1 Invalid Email: System displays a message letting the user know their credentials are incorrect [system action]

3.1.1 User enters a registered email and other valid data [user action]

3.1.2 User clicks on 'Login' button [user action]

3.1.3 User is redirected to the dashboard page [system action]

3.2 Invalid Password: System displays a message letting the user know their credentials are incorrect [system action]

3.1.1 User enters a valid password and other valid data [user action]

3.1.2 User clicks on 'Login' button [user action]

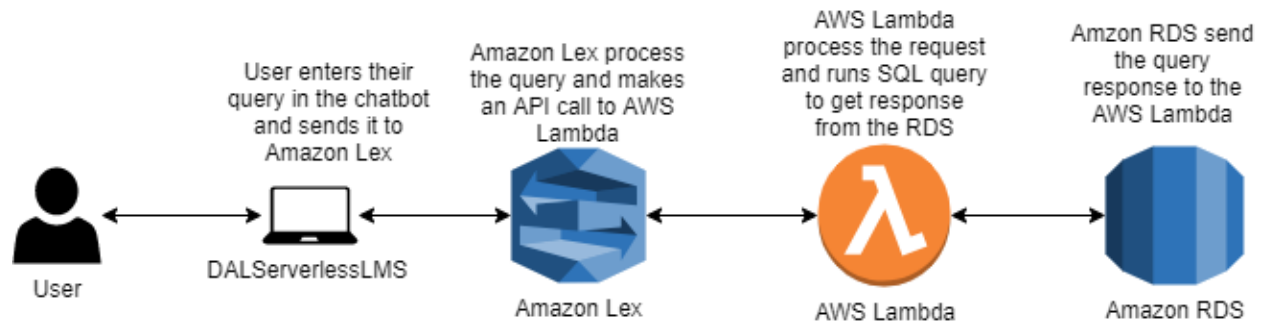
3.1.3 User is redirected to the dashboard page [system action]

3.2 Invalid Security Question: System displays a message letting the user know their credentials are incorrect [system action]

- 3.1.1 User enters a valid security question and other valid data [user action]
- 3.1.2 User clicks on 'Login' button [user action]
- 3.1.3 User is redirected to the dashboard page [system action]
- 3.2 Invalid Security Answer: System displays a message letting the user know their credentials are incorrect [system action]
- 3.1.1 User enters a valid security answer and other valid data [user action]
- 3.1.2 User clicks on 'Login' button [user action]
- 3.1.3 User is redirected to the dashboard page [system action]

## 2.3 Online Support Module

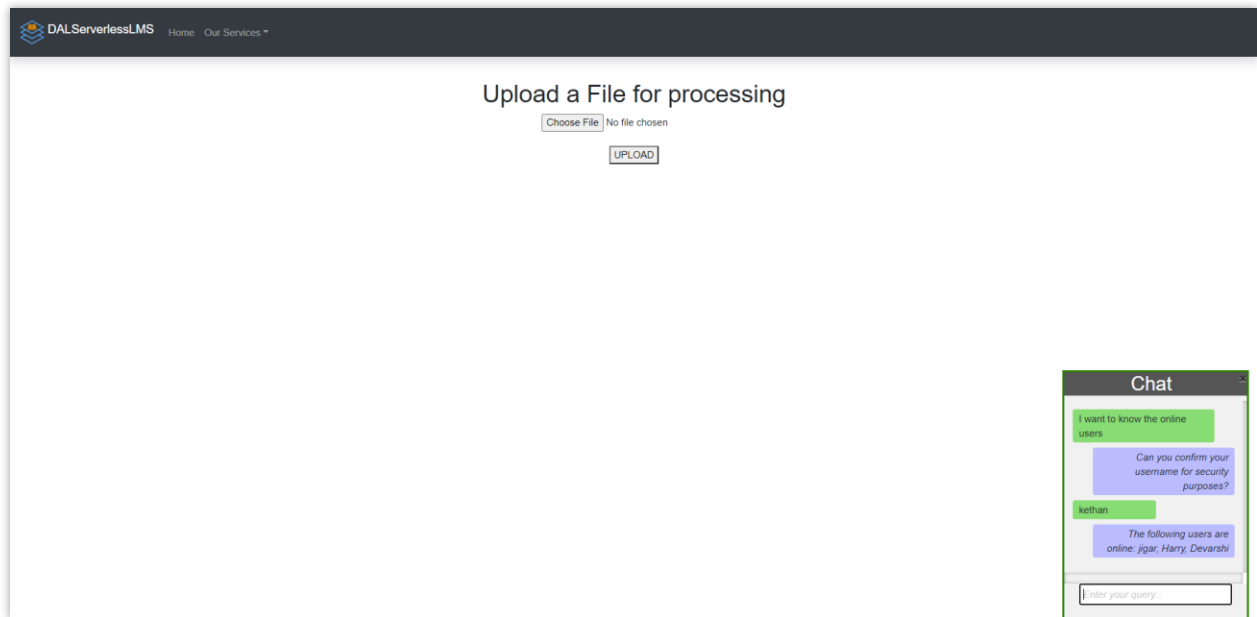
### 2.3.1 Serverless Architecture



*Figure 7: Online Support module architecture*

As shown in the above figure, the user will enter their query in the DALServerlessLMS web application and click on submit button, the query is sent to the Amazon Lex Chatbot. The Lex Chatbot then checks whether any data needs to be retrieved from the database or not. If the query does not require any information from the RDS, then it immediately sends the response back to the user which will be displayed in the chat window. If the query requires any user information from the database, then an API call is invoked to the AWS Lambda to get the data to form the RDS. The Lambda function will run the appropriate SQL queries to retrieve the data from the RDS database and sends it back to the Amazon Lex which will display the results in the chat window of the application.

### 2.3.2 Prototypes



*Figure 8: Online Support module architecture*

### 2.3.3 Use Cases

#### 2.3.3.1 Use Case 1: Simple query without Database connection

In this use case, the user will enter any query that does not require any information from the database. The query contains the pre-defined utterances that are given by the user while building the Chatbot in Amazon Lex. Once the user enters the query and click submit, the query is sent to the Amazon Lex. The query will be parsed by the Amazon Lex and the corresponding response is sent to the user. [5]

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** Any User
- **Goal:** To get a response from the Chatbot for a simple query
- **Task:** Send a simple query to the Lex Chatbot
- **Defining our user persona:** Any user can use this feature
- **Defining our application:** The application would allow users to get information by entering queries

#### 2.3.3.1.1 Test Cases

1. Go to the DALServerlessLMS application
2. Log in to the application using the Username and Password
3. Go to the application Home page after successful login
4. Select 'Online Support' in 'Our Services' drop-down button



5. A chat window is displayed on the web page
6. The user enters the basic query into the chat window
7. Lex Chatbot responds with the answer to the user query
8. User can ask the follow-up questions
9. The user ends the chat by clicking the End Chat button.

#### 2.3.3.2 Use Case 2: Query requires using Lambda and RDS

In this use case, the user will enter any query that requires information from the database. The query contains the pre-defined utterances that are given by the user while building the chatbot in Amazon Lex along with the unknown values which are to be retrieved from the database. Once the user enters the query and clicks submit, the query is sent to the Amazon Lex. The Amazon Lex then processes the query and sends a request to AWS Lambda using API calls. AWS Lambda receives the request and retrieves the information present in the RDS by running SQL queries. The retrieved data is then sent to the Amazon Lex via AWS Lambda.

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** Any User
- **Goal:** To get a response from the chatbot by connecting to RDS via AWS Lambda
- **Task:** Send a complex query to the Lex chatbot
- **Defining our user persona:** Any user can use this feature
- **Defining our application:** The application would allow users to get information by entering queries

#### 2.3.3.2.1 Test Cases

1. Go to the DALServerlessLMS application
2. Log in to the application using the Username and Password
3. Go to the application Home page after successful login
4. Select 'Online Support' in 'Our Services' drop-down button
5. A chat window is displayed on the web page
6. The user enters the complex query into the chat window
7. Lex Chatbot sends the query to the AWS Lambda using APIs
8. AWS Lambda gets the query response from the RDS and sends back to Lex Chatbot
9. Lex Chatbot displays the response in the Chat window
10. User can ask the follow-up questions
9. The user ends the chat by clicking the End Chat button.

## 2.4 Chat Module

### 2.4.1 Serverless Architecture

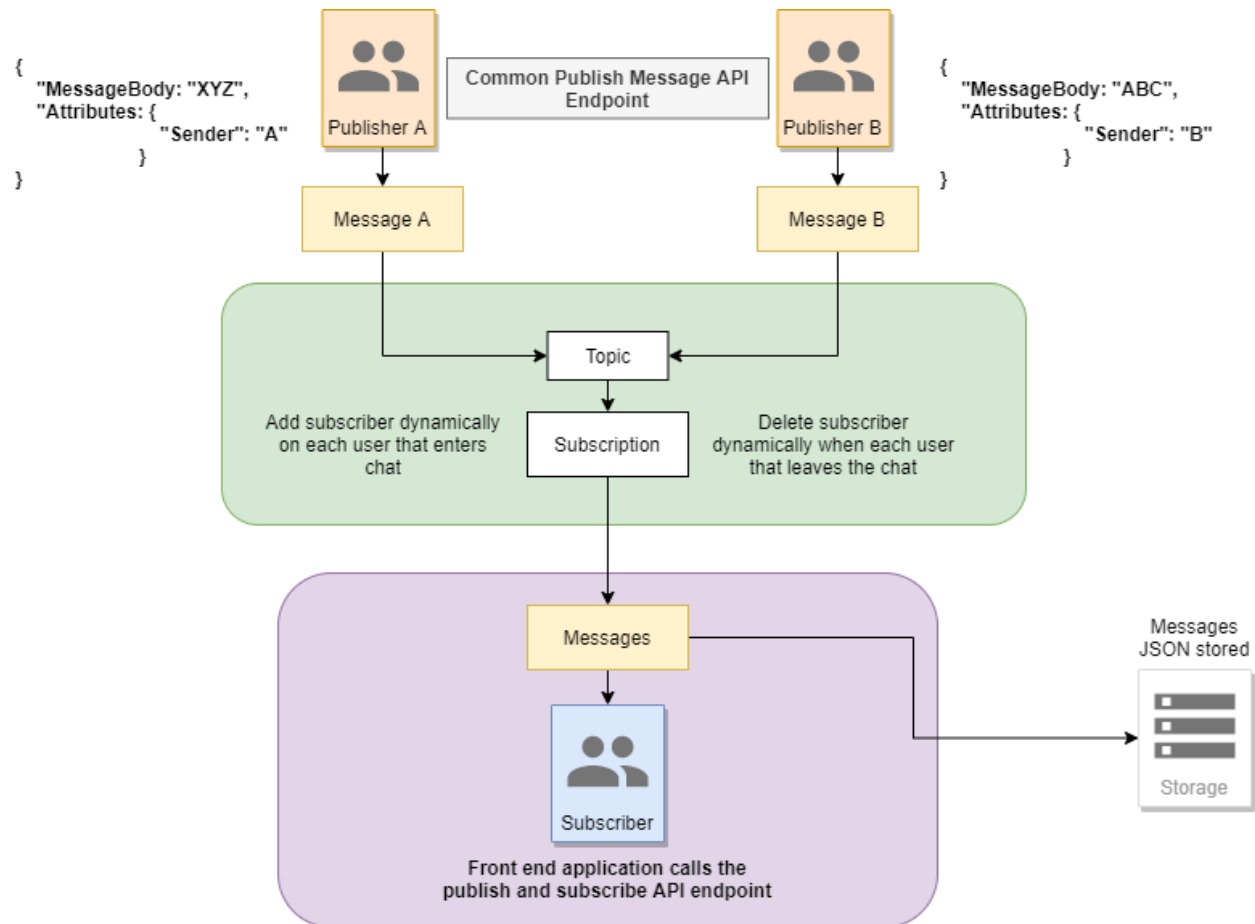
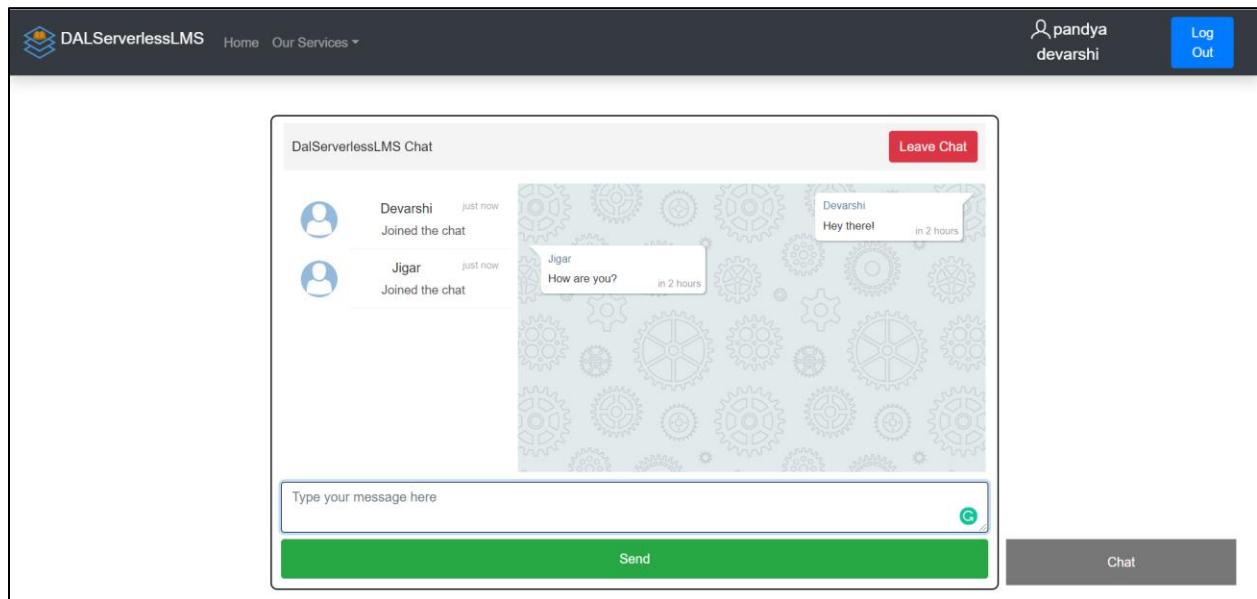


Figure 9: Chat module architecture

We created five APIs using cloud functions [6] – Publisher API, Subscriber API, Create Subscriber API, Delete Subscriber API, Store to GCS API. When a user enters the chat room, a unique subscriber is created, and the user can publish messages using the common publisher API using GCP Pub/Sub [7]. When another user logs in, a new subscriber is created, and the user pulls messages from the respective subscriber using the subscriber API. When the user leaves the chat, the delete subscriber API is called which deletes the respective subscriber. Meanwhile, all the messages are stored on Google Cloud Storage using the store to GCS API created using the cloud function.

## 2.4.2 Prototypes



*Figure 10: Chat module prototype*

## 2.4.3 Use Cases

**User Scenario:** Jigar wants to send a message to Devarshi regarding the code update of the login module.

Use Case:

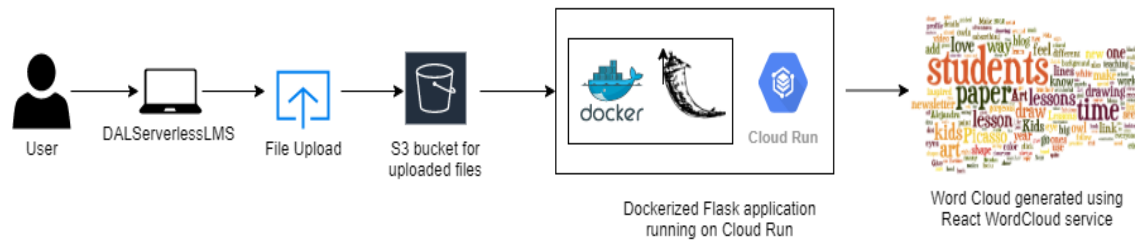
- The user opens the web application and lands on the landing page.
- User logs in into the system using his credentials.
- The user then navigates to the chat system.
- The chat screen opens up.
- The user searches for another user in the search box.
- The user types a message in the message field and sends it to the user by pressing the send button

### 2.4.3.1.1 Test Cases

- Whether the subscription is created when the user logs in.
- Whether the subscription is deleted when the user leaves the chat.
- Whether the message is received by the other user.
- Whether the message is acknowledged after it is pulled.

## 2.5 Data Processing

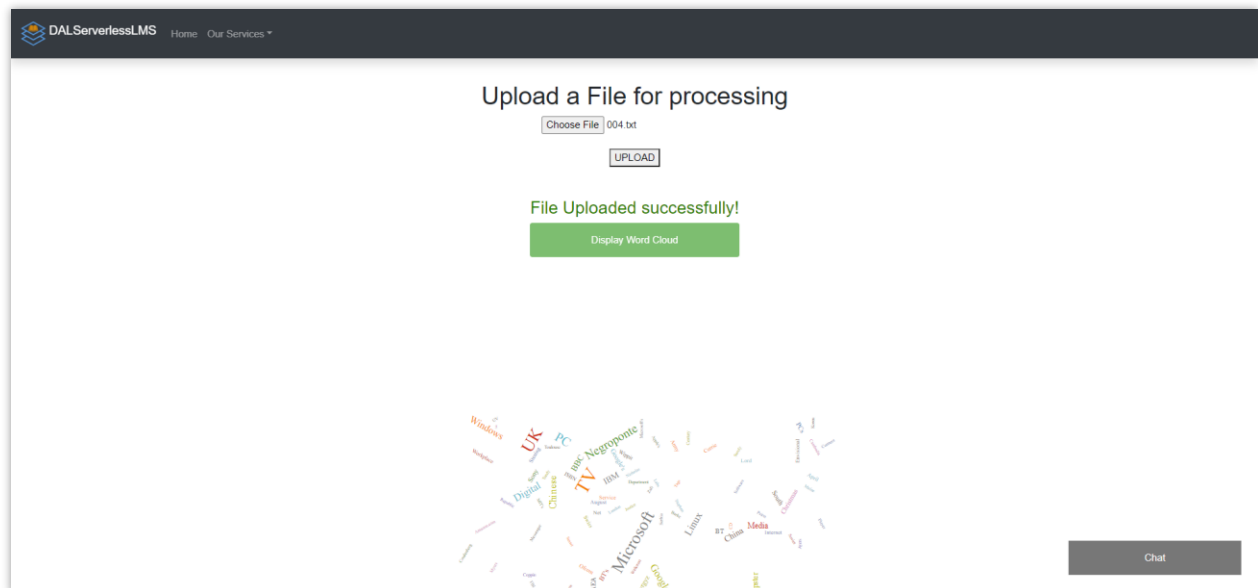
### 2.5.1 Serverless Architecture



*Figure 11: Data Processing architecture*

As shown in Figure 13, the user opens the DALServerlessLMS application and log in to it. They will then upload the file using the file upload option in the application. The uploaded file stored in the AWS S3 bucket. When the user clicks on the process button the uploaded file is processed by the Dockerized Flask application that is running on the Cloud Run. The extracted named entities then will be retrieved using a REST API and the word cloud is displayed in the application using the React WordCloud package in the React.js framework. [8]

### 2.5.2 Prototypes



*Figure 12: Data Processing prototype*

### 2.5.3 Use Cases

#### 2.5.3.1 Use Case 1: Build the word cloud

In this use case, the user will upload a file into the DALServerlessLMS application. Once the file is uploaded, the file is sent to the Flask service running on the Docker container. The Flask service will process the document and upload it to the S3 bucket. The Flask service also extracts

all the named entities from the uploaded file and create a file in another S3 bucket with the names. A word cloud is created by the named entities that are present in the S3 document.

What we identify from this scenario:

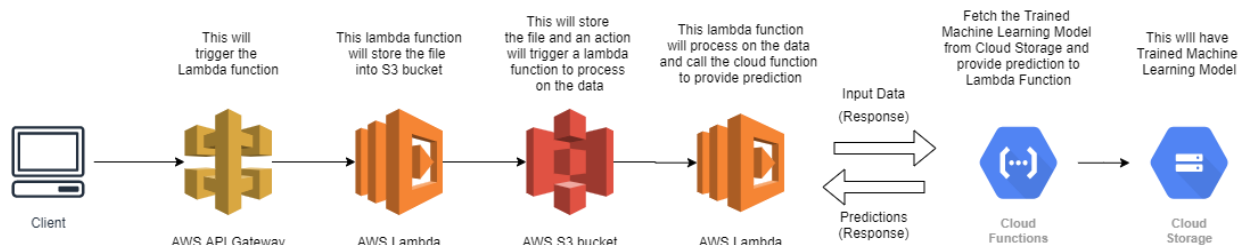
- **Application:** DALServerlessLMS
- **User Persona:** Any User
- **Goal:** To build the word cloud from the named entities extracted from the uploaded file
- **Task:** Generate the word cloud
- **Defining our user persona:** Any user can use this feature
- **Defining our application:** The application would allow users to generate a word cloud

#### 2.5.3.1.1 Test Cases

1. Go to the DALServerlessLMS application
2. Log in to the application using the Username and Password
3. Go to the application Home page after successful login
4. Upload the file for named entity extraction
5. Once the upload button is clicked, the file is sent to the S3 bucket
6. Once the file is uploaded to S3 bucket the display word cloud button appears on the screen
7. On click of the Display word cloud button, the Flask API running on the Docker container is called and entities are extracted from the file uploaded into the S3 bucket
8. The extracted entities are then displayed as a word cloud on the web page

## 2.6 Analysis 1

### 2.6.1 Serverless Architecture



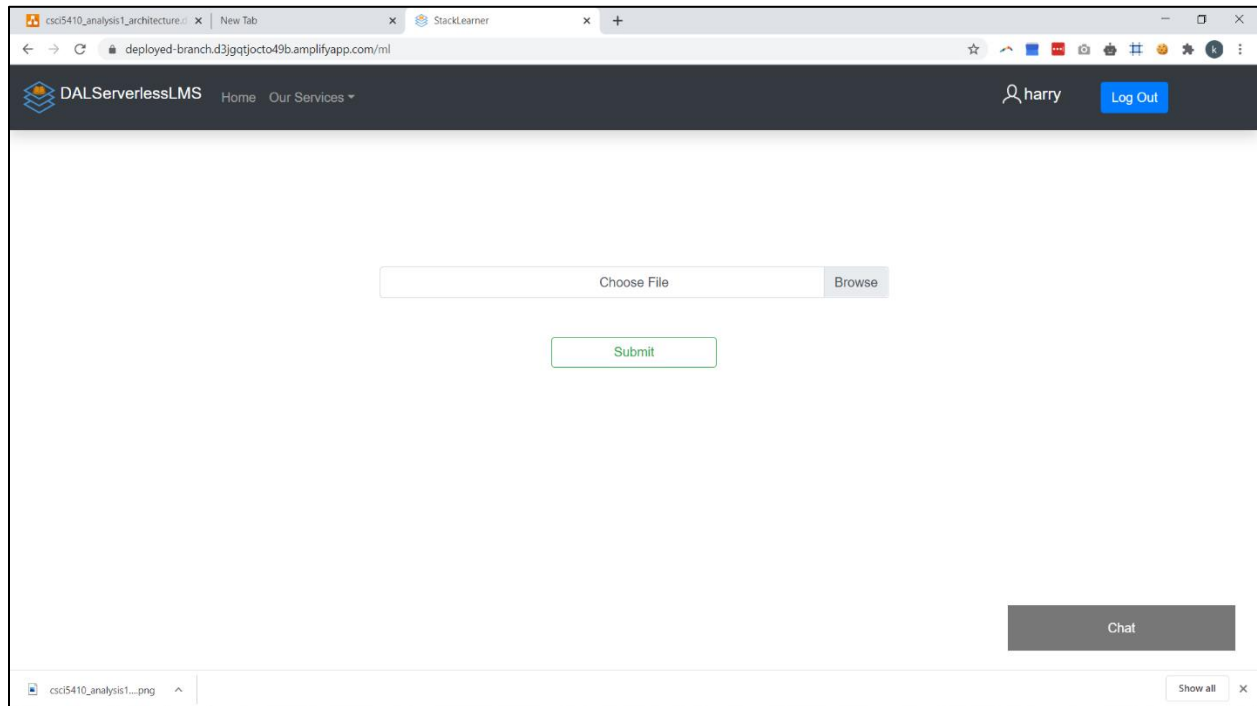
*Figure 13 Serverless Architecture of Analysis 1*

In this architecture, the user will first use the application and hit the Amazon API gateway [3] through which it will trigger an AWS lambda function [3]. Now the user will pass the file when hitting the API gateway. Thus, the lambda function will have the File to work on. First Lambda function will store the file in the S3 bucket. That bucket has an action attached to it which will trigger another lambda function and pass the file to that lambda function. Now, the file will be

processed by lambda and then call a cloud function [4] to make prediction using the data passed by this lambda function. The cloud function will have a trained machine learning model to make the prediction. That model is stored in the cloud storage as pickle file. The cloud function will fetch the model from cloud storage. After making prediction the results will be passed to lambda function.

NOTE: We are using K-means as our algorithm and for the output (response from Cloud Function) we will be creating clusters in the S3 bucket.

## 2.6.2 Prototypes



*Figure 14 Prototype for uploading documents*

## 2.6.3 Use Cases

### 2.6.3.1 Use Case 1: Upload all the document

The user wants to upload all the documents and then the user wants to separate them using their file names.

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** A new user or existing one
- **Goal:** To upload all files in this application and the future, he wants to separate/organize it.
- **Task:** Upload all the files and perform Analyses.
- **Defining our user persona:** A new or existing user who is interested in uploading and performing some analysis on files and organize them.

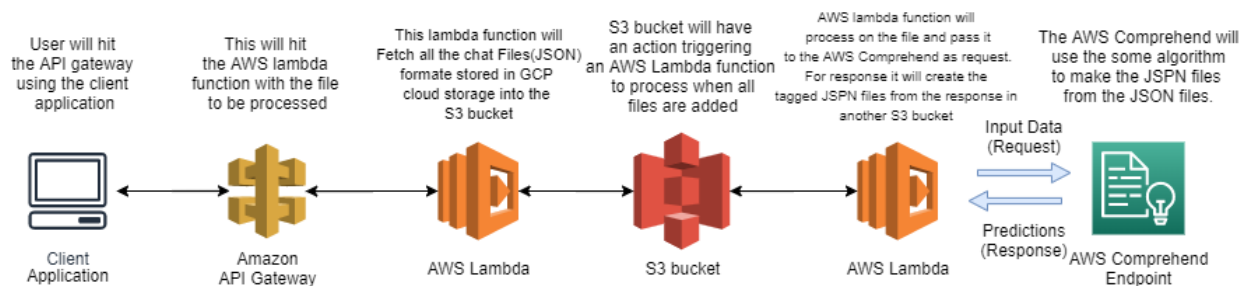
- **Defining our application:** The application would allow users to upload all the files, analyze them, and organize them.

#### 2.6.3.1.1 Test Cases

1. The file is present or not at the destination.
2. The file is should not be null.
3. Data cleaning should not miss any important data.
4. There should be a proper pipeline to store the data in the S3 bucket and fetching it into the lambda function.
5. AWS lambda function should have access to files from the S3 bucket.
6. AWS lambda function should have access to cloud function.
7. Does User must have proper authenticity to perform the task.
8. Cloud function should return some values to be considered as performed analysis.
9. Cloud function should receive proper values to perform analysis.
10. Authenticity to access the cloud function.
11. Authenticity to access cloud storage.
12. The model should be stored in cloud storage.
13. The model should be trained properly to make a genuine prediction.

## 2.7 Analysis 2

### 2.7.1 Serverless Architecture



*Figure 15 Serverless Architecture for Analysis 2*

In this architecture, we will first get all the raw data from GCP cloud storage [4] using a lambda function [3]. This lambda function is also responsible for storing the fetched data into an AWS S3 bucket [3]. All the fetched data will in the format of JSON. Now, the bucket will have an action attached to it which will trigger an AWS lambda function to process the data and pass it to AWS Comprehend. Now, AWS Comprehend will tag all the chat messages and store them in the form of JSON. This JSON will be the response for the Lambda function and it will store it into other AWS S3 buckets.

### 2.7.2 Prototypes

System will perform active sentiment analysis on chat data thus, it will not have a frontend.

### 2.7.3 Use Cases

#### 2.7.3.1 Use Case 1: Performing Sentiment Analysis

The system wants to perform sentiment analysis on chat data.

What we identify from this scenario:

- **Application:** DALServerlessLMS
- **User Persona:** System(Application owner or System itself if it is set to do this task automatically)
- **Goal:** To perform sentiment analysis on chat data and create tagged chat messages and store them into the AWS S3 bucket.
- **Task:** perform sentiment analysis on chat data
- **Defining our user persona:** System wants perform sentiment analysis on chat data.
- **Defining our application:** The application would allow system to perform sentiment analysis on the chat data.

#### 2.7.3.1.1 Test Cases

1. The file is present or not at GCP cloud storage.
2. The file is should not be null.
3. Data cleaning should not miss any important data.
4. There should be a proper pipeline to store the data in the S3 bucket and fetching it into the lambda function.
5. AWS lambda function should have access to files from the S3 bucket.
6. AWS lambda function should have access to files from GCP cloud storage.
7. Does User must have proper authenticity to perform the task.
8. AWS Comprehend should return some values to be considered as performed analysis.
9. AWS Comprehend should receive proper values to perform analysis.
10. Tagged data must be in the form of JSON and stored into different S3 buckets.

## 2.8 Web Application Building and hosting

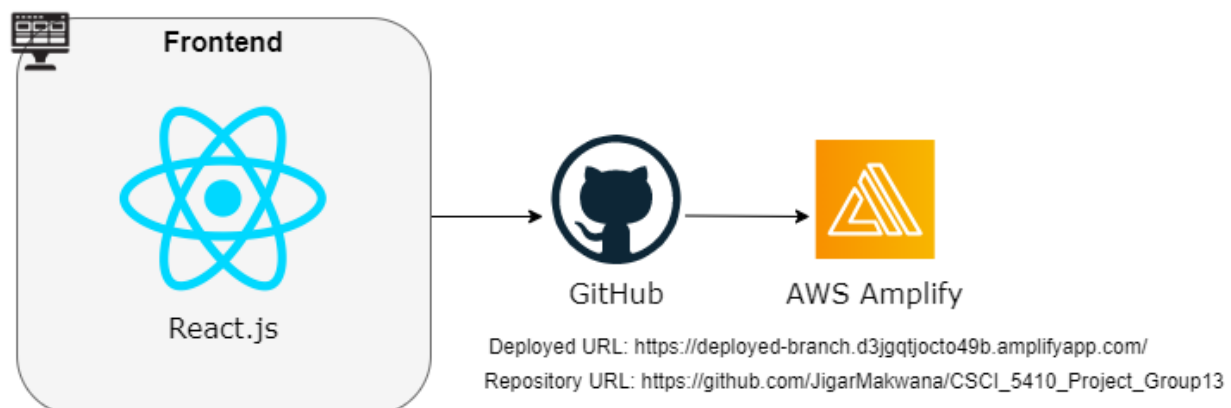
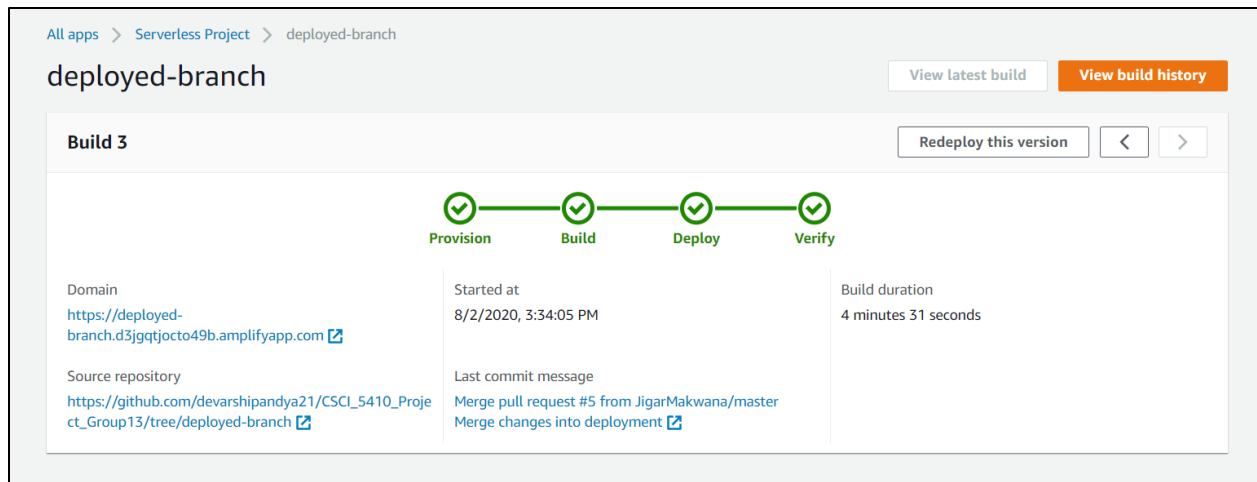


Figure 16: Deployment Architecture





*Figure 17: Deployment on AWS Amplify*

The app has been deployed on AWS Amplify [9] using our repository on GitHub. During each new update in the branch, the app is set on auto-deployment. The app can be accessed using the deployment URL provided, and all the features are up and running there.

AWS Amplify smartly identifies the language and framework used to develop the project and chooses appropriate build scripts. However, we modified the build scripts in the package.json file because there were some memory issues and the build was failing.

### 3. Overall Serverless Architecture

Figure 18 shows the overall architecture of the DalServerlessLMS.

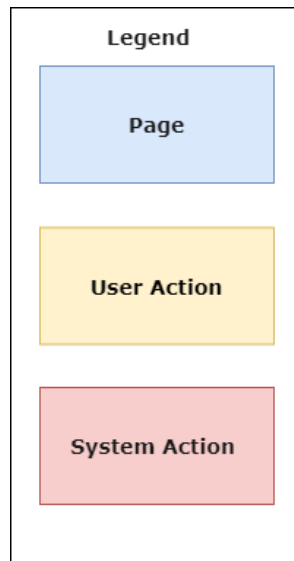


Figure 18 Overall Architecture

## 4. Workflow

The below figure shows the Legend used in the workflow diagrams

---



*Figure 19: Legend*

#### 4.1 User Management Module

Figure shows the workflow diagram of the Registration functionality of the 'DALServerlessLMS' application.

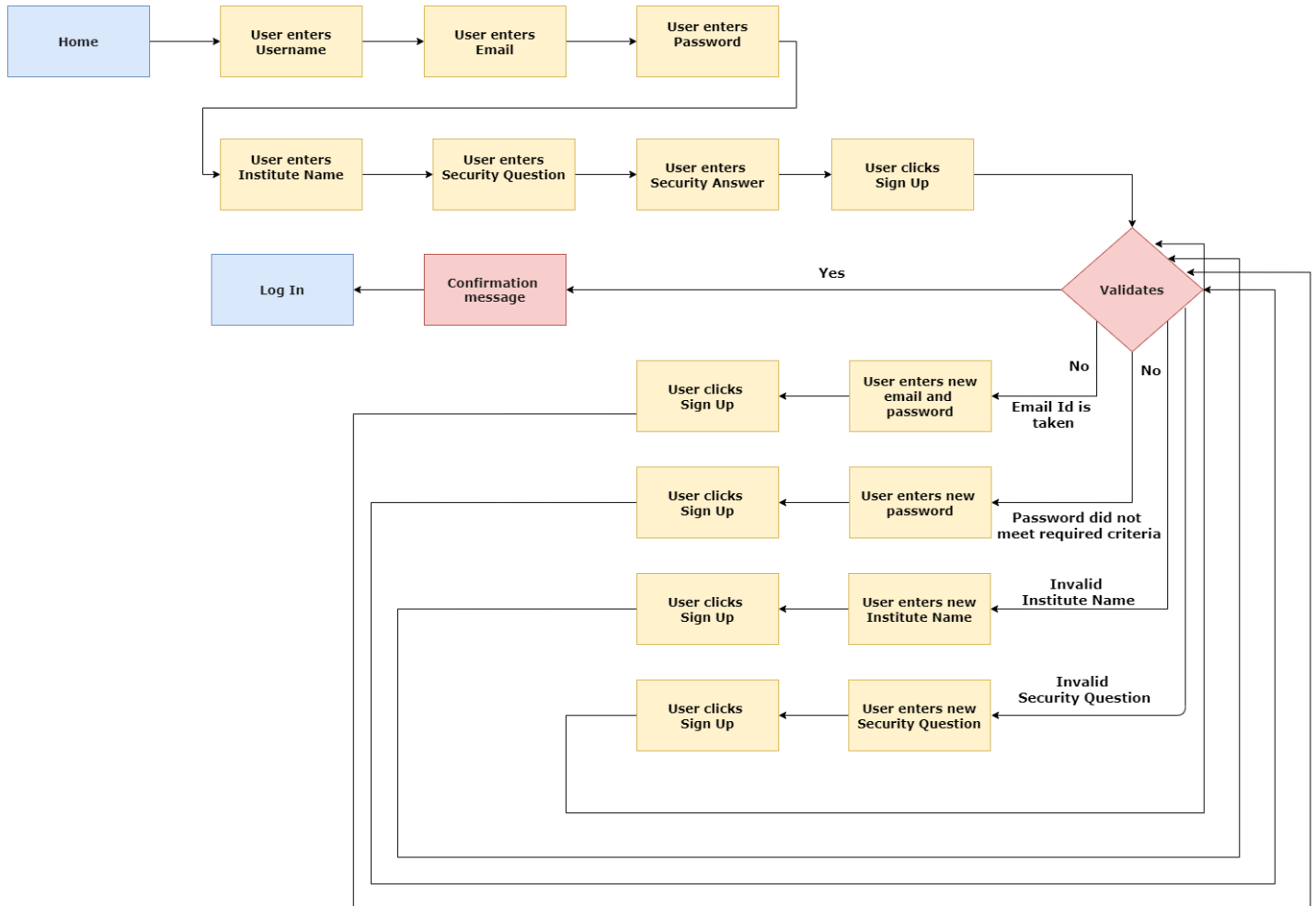


Figure 21 Workflow of User Management Module

## 4.2 Authentication Module

Figure shows the workflow diagram for the Login functionality of the 'DALServerlessLMS' application.

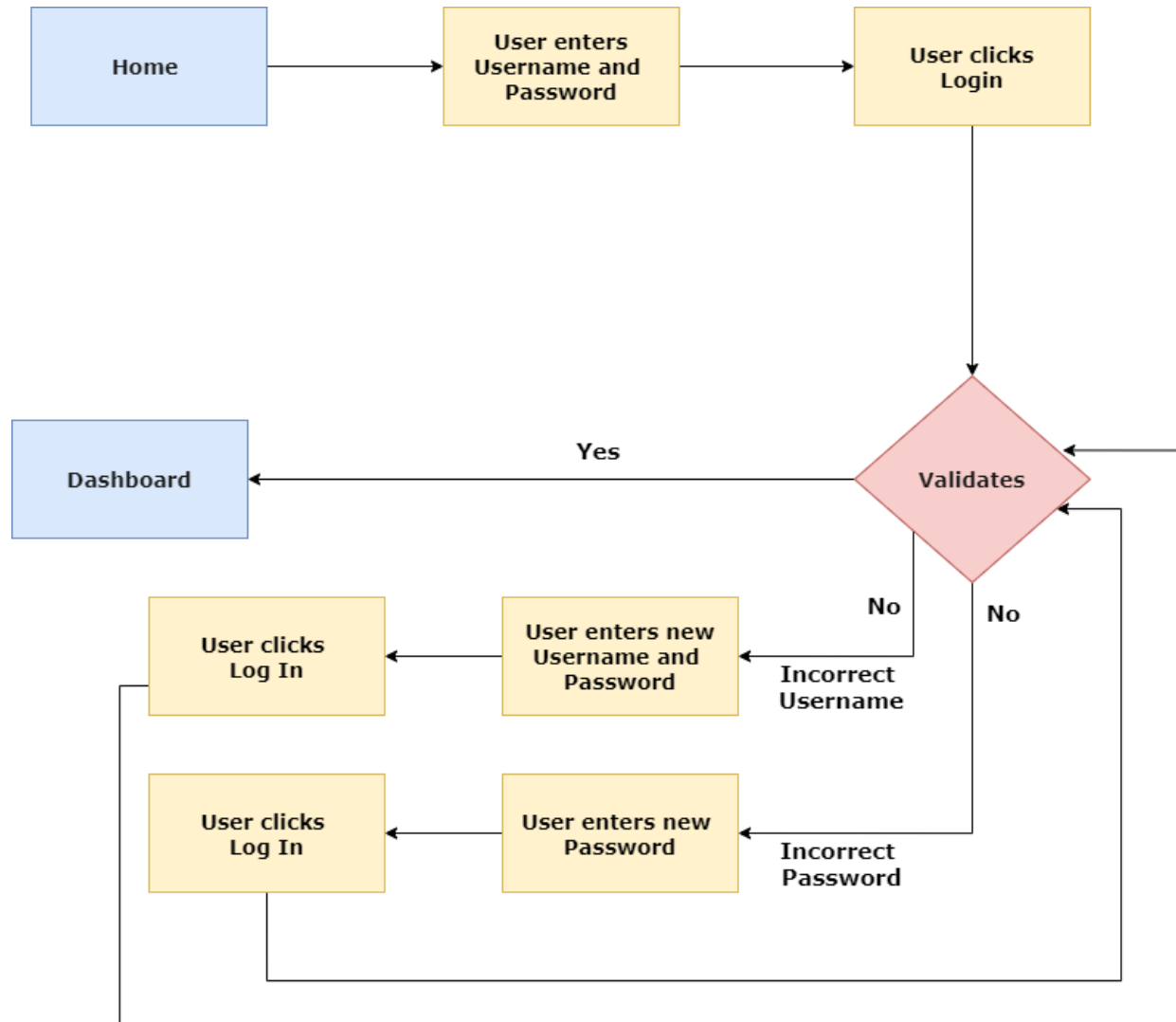
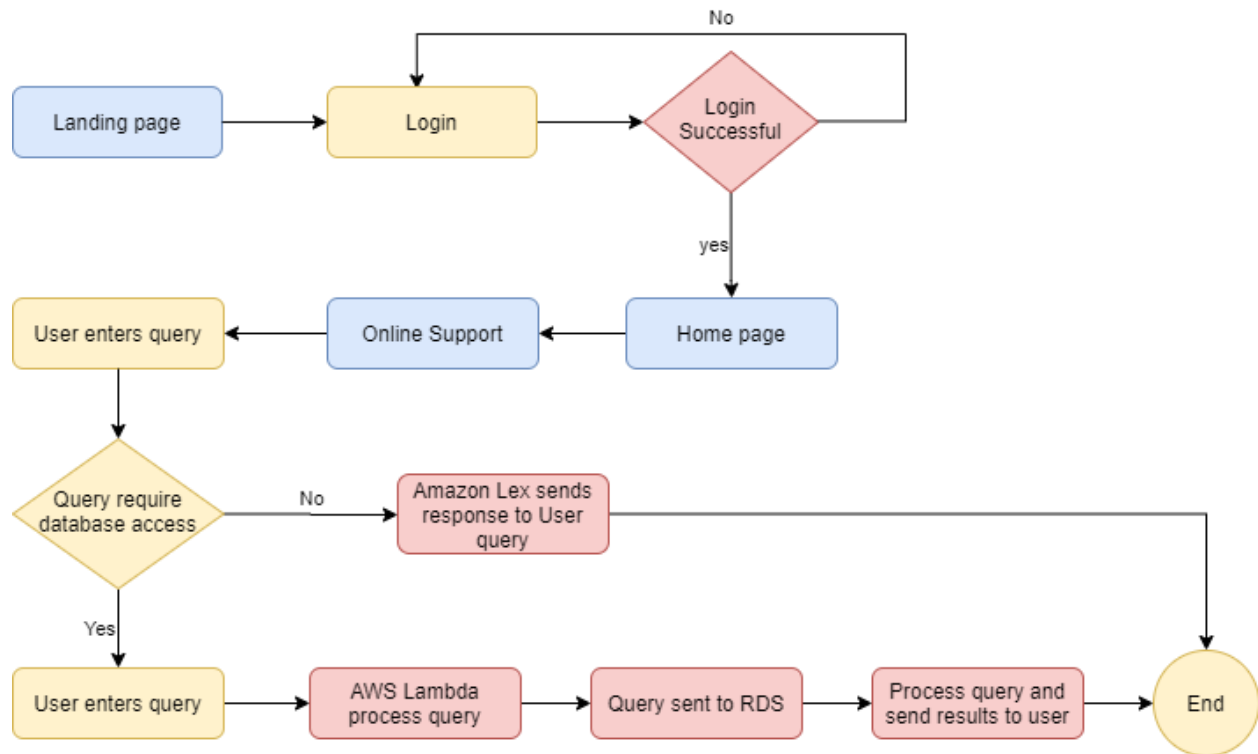


Figure 22 Authentication Workflow

### 4.3 Online Support Module



*Figure 23 Workflow of Online Support Module*

#### 4.4 Chat Module

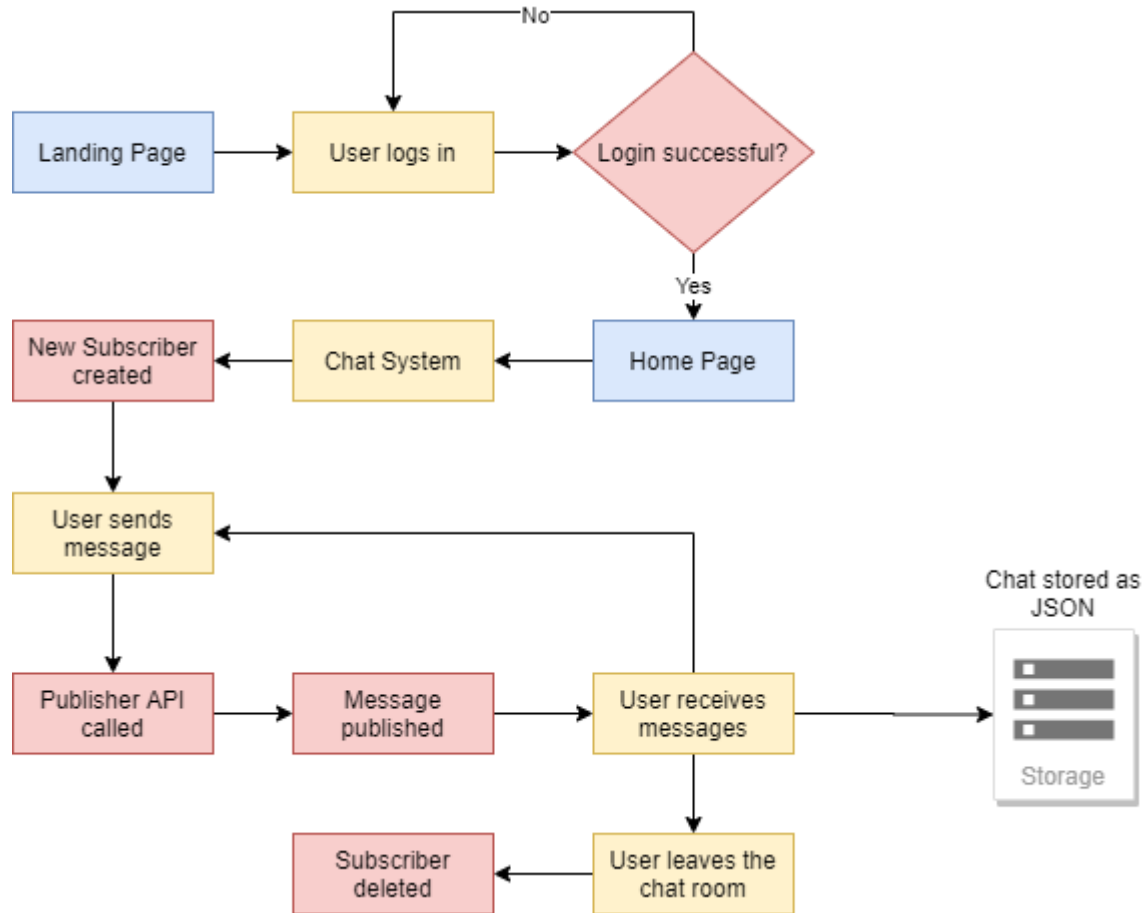
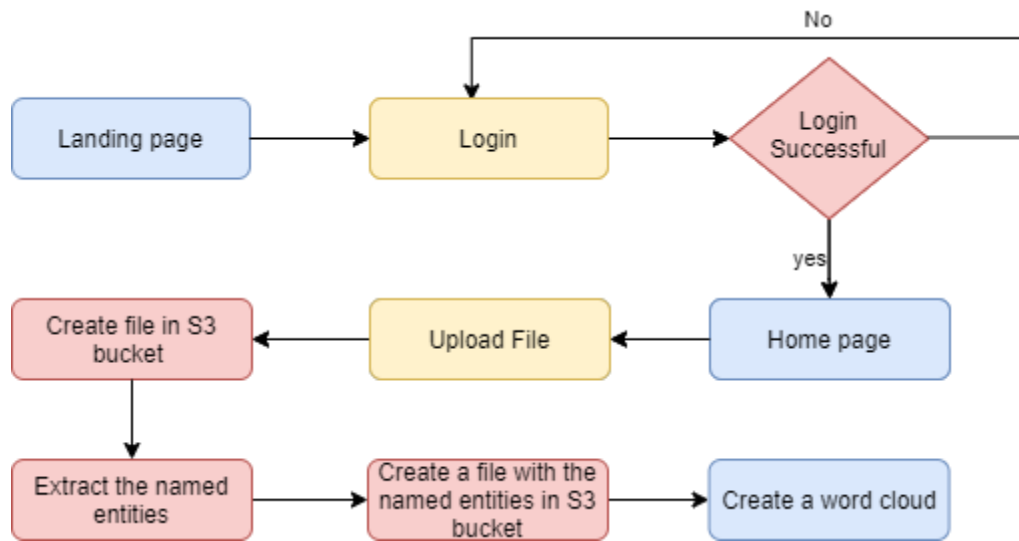


Figure 24 Workflow of Chat Module

## 4.5 Data Processing



*Figure 25 Workflow of Data Processing*



## 4.6 Analysis 1

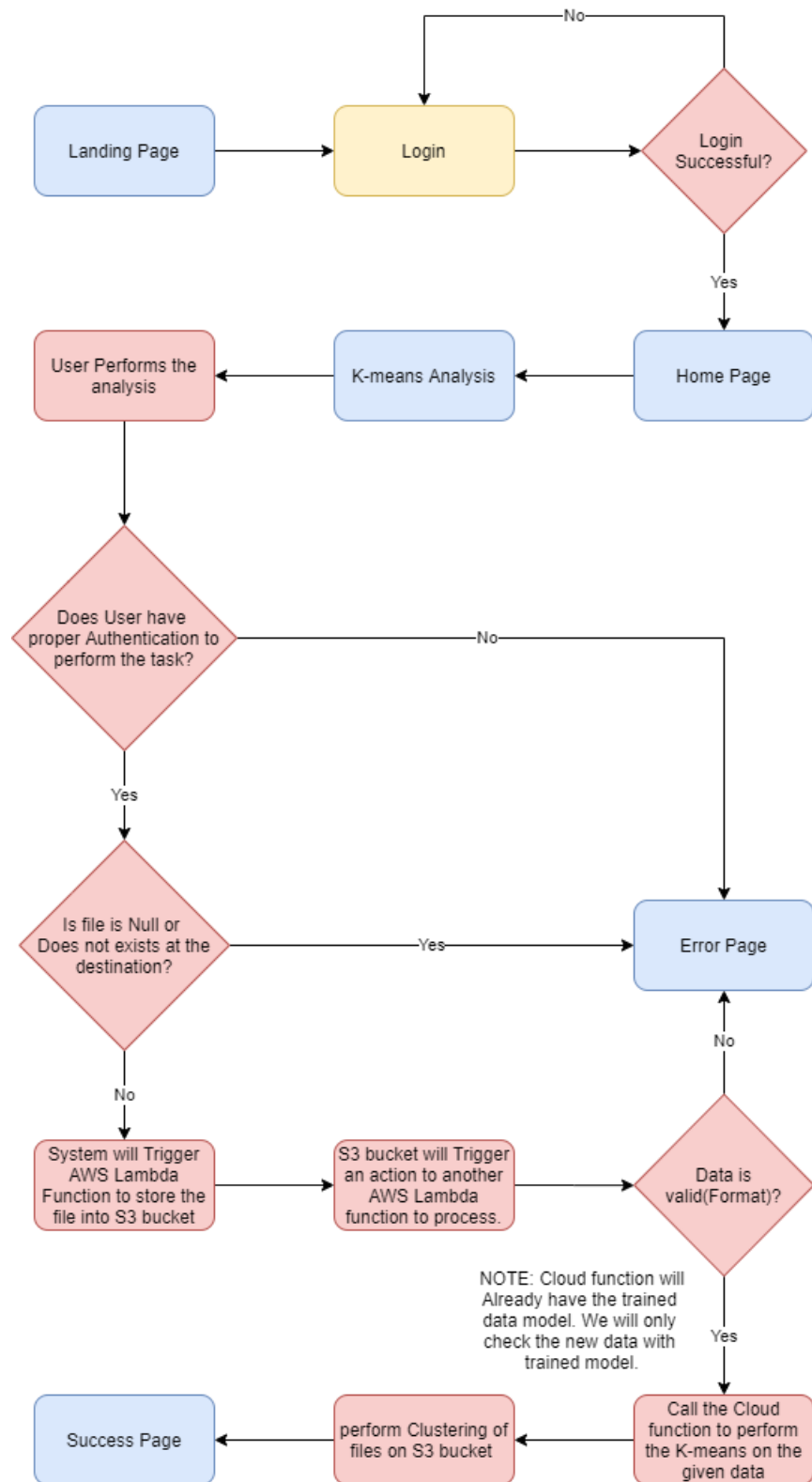


Figure 26 Workflow of Analysis 1

## 4.7 Analysis 2

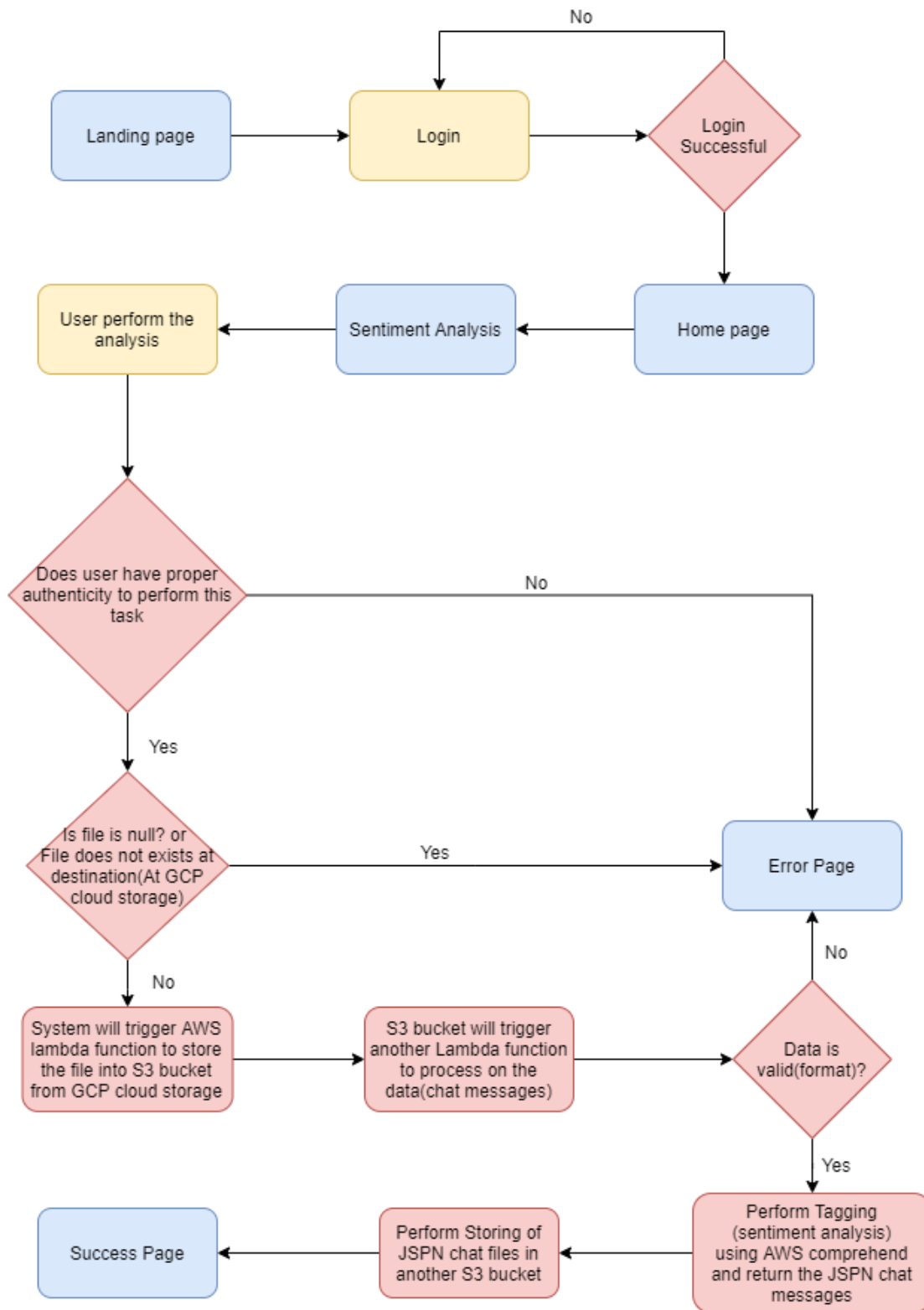


Figure 27 Workflow of Analysis 2

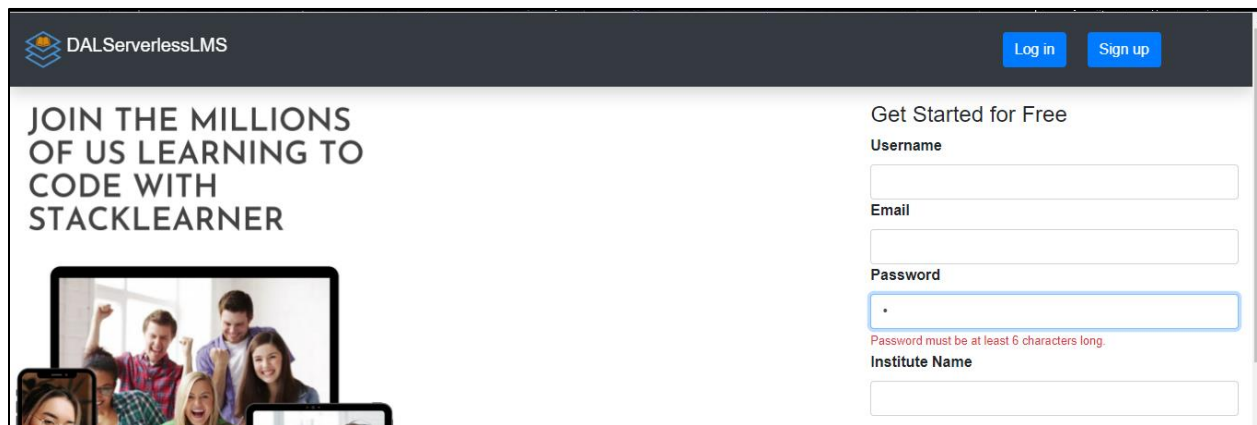
## 5. Testing

To test the DALServerlessLMS application, we have mainly performed the manual testing but the testing can be automated using scripts and unit tests. We have use the following tools for testing:

1. Amazon CloudWatch logs
2. Google Cloud Logs Viewer
3. Amazon Lambda Test Events
4. Google Cloud Function Triggering Events
5. Postman

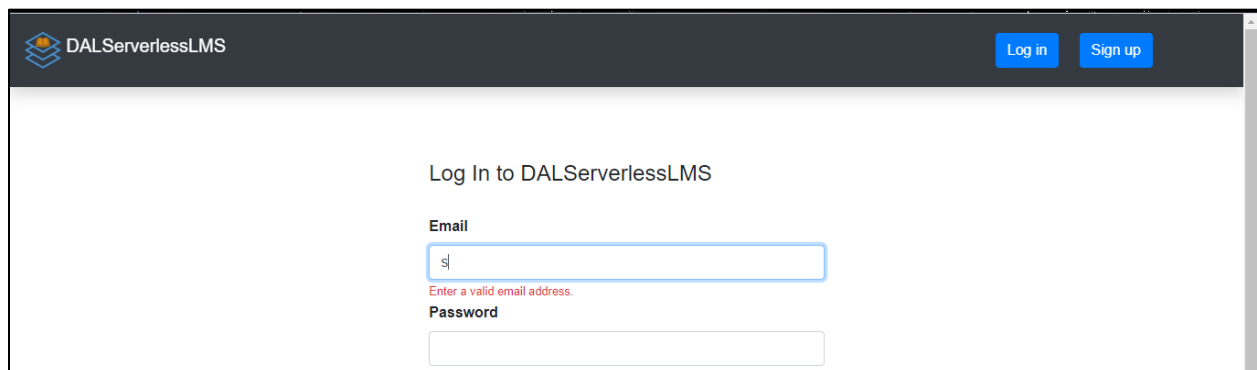
### Screenshots of testing invalid data

The following screenshots demonstrates the validation handled when the user enters the bad or invalid data.



The screenshot shows the DALServerlessLMS sign-up page. On the left, there is a promotional banner with the text "JOIN THE MILLIONS OF US LEARNING TO CODE WITH STACKLEARNER" and an image of a group of people. On the right, there is a "Get Started for Free" form with fields for Username, Email, Password, and Institute Name. The Password field is highlighted with a blue border, and a red error message "Password must be at least 6 characters long." is displayed below it. The Username and Email fields are empty, and the Institute Name field is also empty. The Log in and Sign up buttons are visible in the top right corner.

*Figure 28 Password validation*



The screenshot shows the DALServerlessLMS login page. The header is the same as the sign-up page. The main heading is "Log In to DALServerlessLMS". Below it, there are fields for Email and Password. The Email field is highlighted with a blue border, and a red error message "Enter a valid email address." is displayed below it. The Password field is empty. The Log in and Sign up buttons are visible in the top right corner.

*Figure 29 Email Validation*

The screenshot shows the DALServerlessLMS login interface. At the top left is the logo and name 'DALServerlessLMS'. At the top right are 'Log in' and 'Sign up' buttons. The main heading is 'Log In to DALServerlessLMS'. Below this are input fields for 'Email' (containing 's@fst.fgs'), 'Password' (masked with dots), 'Security Question' (containing 'fdsgd'), and 'Security Answer' (masked with dots). Below the 'Security Answer' field is a red error message: 'Invalid credentials. Please try again.' Below the error message is a blue 'Log in' button. At the bottom left, there is a link 'Login with another account' and social media icons for Google, Facebook, LinkedIn, and GitHub. At the bottom right is a grey 'Chat' button.

*Figure 30 Authentication failure*

## 6. Repository and Deployment Links

The following are the URLs for accessing the GitHub repository of code base and deployed application on AWS Amplify. Readme file can be found on the GitHub repository.

6.1 GitHub Repository Link:

[https://github.com/JigarMakwana/CSCI\\_5410\\_Project\\_Group13](https://github.com/JigarMakwana/CSCI_5410_Project_Group13)

6.2 Deployment Link:

<https://deployed-branch.d3jgqtjocto49b.amplifyapp.com/>

## 7. Task Achievement

Even though we have faced a lot of challenges while working on the application, we were able to finish the application successfully and we were able to satisfy all the requirements that are provided to us. We were able to develop individual modules that use serverless components and perform various tasks and these individual modules are integrated into an application using React.js in the frontend and Node.js in the backend.

## 8. Challenges faced

- Providing MFA support using the student account.
- The online support module requires a complex chatbot system, and its building and testing will take a lot of time.

- Working on the Machine Learning module requires a considerable amount of background knowledge and time.
- Working with GCP Pub/Sub was quite tricky as does not have a perfect use case for building an instant messaging engine. Designing and thinking of its architecture were a bit tricky.
- The messages take a few seconds to send and receive due to timeout functions on the frontend which call the publisher and subscriber API every few seconds. Also, pulling messages take a few seconds.
- The build scripts need to be modified to deploy the application on AWS Amplify. The application was using more memory than the allotted memory, and so the build process kept failing. Finally, we modified the build scripts and changed the environment variables on AWS Amplify to limit memory usage.

## 9. Methods of knowledge transfer

---

- We did meetings to know everyone's module. To get a clear picture in our mind we explained our module to each other. We taught each other about our understanding of the module. We used Microsoft Teams to achieve this task.
- Every team member shared their resources for their modules thus, we all are assured to be on the same knowledge as other team members.
- Each team member gave some tasks to every other team member from which we got a better understanding of every other module.

## 10. Team

---

All the members of the team are the graduate students at Dalhousie University in the Master of Applied Computer Science.

### Team Lead:

Team selected Jigar Makwana as a Team Leader as he possesses knowledge of complete Software Development Life Cycle and Team Planning and Management. He also possesses corporate experience, so team wanted to leverage his skills to better organize the team work.

### Team Members:

#### Devarshi Pandya

He has quite good experience with web development frameworks and libraries like Angular and ReactJS. His knowledge of Python language was quite helpful in building cloud functions for the chat module and the frontend for the chat module was beautifully designed using his user experience and ReactJS skills.

#### Krutin Trivedi

Krutin has very good experience in web development with 4 months of industry experience. He also has very good exposure on the end to end testing of the application. His best strength is UI/UX. He is skilled with MERN stack, docker, UI/UX, different frontend libraries.

Jigar Makwana

Jigar has 3 years of work experience in programming languages such as C, C++, C#, and Java. He is familiar with the ReactJS frontend library. He possesses a good amount of knowledge in backend development using NodeJS. He is good at planning and management using Agile methodology.

Kethan Dasari

He has prior experience in Web application development. He has worked with frameworks like AngularJS, ReactJS, Express, and Node.js for Web application development.

## Appendix

### A.1 Minutes of Meeting

The following are the screenshots of the minutes of meetings conducted during the project planning, development, testing, and deployment. The minutes of the meetings were recorded by Jigar Makwana.

#### CSCI-5410- Serverless- Minutes of Meeting June 30, 2020

Notebook: Study

Created: 2020-06-30 9:42 PM

Updated: 2020-07-08 9:43 PM

Tags: CSCI5710

Date & Time	June 30, 2020. 9:00 PM
Goal	Discuss about design document of DALServerlessLMS and assigning the tasks
Attendees	<input checked="" type="checkbox"/> Jigar <input checked="" type="checkbox"/> Krutin <input checked="" type="checkbox"/> Kethan <input checked="" type="checkbox"/> Devarshi

#### Agenda

- Discuss about design document of DALServerlessLMS
- Dividing the work among team members

#### Notes

- All members are responsible for developing unit testing for their modules
- All members are expected to test the system (developed solution) thoroughly
- All members are expected to collaborate on documentation

Action Items	Assignee	Status
<input type="checkbox"/> Provide a format of Design Document which also outlines the structure of Use Cases	Jigar	In progress
<input type="checkbox"/> Provide a tool name and a dummy diagram for creating Serverless Architecture Diagrams <input type="checkbox"/> Provide a tool name and a dummy diagram for creating Workflow Diagrams	Krutin	In progress
As of now preferences of modules <ul style="list-style-type: none"><li>• User Management Module (Jigar)</li><li>• Authentication Module (Jigar)</li><li>• Online Support Module (Kethan)</li><li>• Chat Module (Devarshi)</li><li>• Data Processing (Kethan)</li><li>• Analysis 1 (ML) (Krutin)</li><li>• Analysis 2 (ML) (Krutin)</li><li>• Web Application Building and hosting (Devarshi)</li><li>• Documentation (All need to work)</li></ul>	All members	Assigned
Create prototypes for front end of the DALServerlessLMS	All members	Assigned

## CSCI-5410- Serverless- Minutes of Meeting July 20, 2020

Notebook: Study

Created: 2020-07-20 5:33 PM

Updated: 2020-07-20 6:12 PM

Date & Time	July 20, 2020. 5:30 to 6:00 pm
Goal	Deciding deadlines for project deliverables
Attendees	<input checked="" type="checkbox"/> Jigar <input checked="" type="checkbox"/> Krutin <input checked="" type="checkbox"/> Kethan <input checked="" type="checkbox"/> Devarshi

### Agenda

Deciding deadlines for project deliverables

Discussing the following points regarding Report

- a. What is the given problem?
- b. Who is in the Team? - Their names, expertise, contribution in project, is he/she a teamLead?, if so - who selected the team Lead?
- c. What platforms and services are used to develop this application? Did you use any platform to share knowledge and to communicate with the team?
- d. Please provide Log (ideally screenshots) of all meetings as part of Appendix 1.
- e. Did you achieve what was expected? According to you, what percentage is not implemented?
- e. Your final Serverless Architectures for the most appropriate use case(s), and modules - This could be a variation of the design architecture that you have submitted. This is the final version that you have implemented. It should not contain any component or service that you failed to use.
- f. What are the challenges that you faced while implementing? These are different than Design challenges. Please write only about implementation challenges
- g. Provide PseudoCode, or Activity Diagram or WorkFlow of the major operations
- h. Provide link to your code repository. Please add a Readme file.

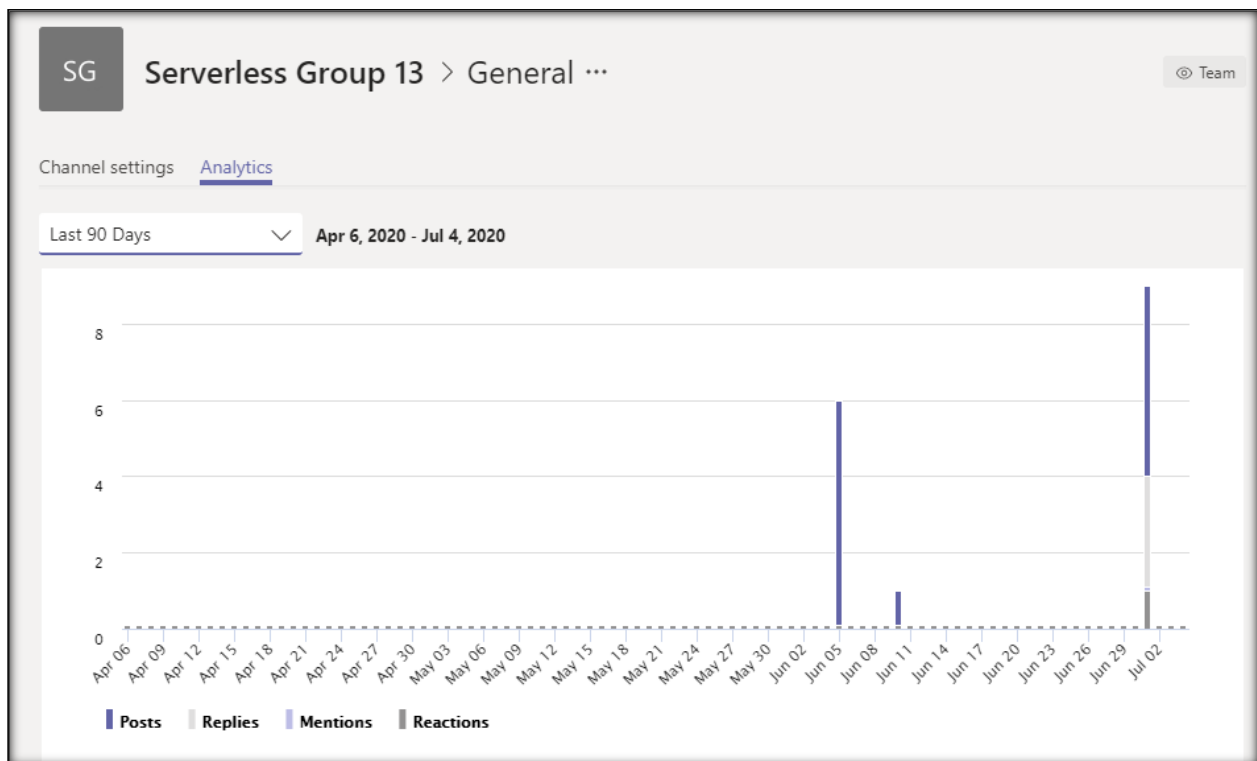
Action Items	Date	Assignee
Making repository and adding members, professor, TA	July 20, 2020	Jigar
Individual modules (Frontend + Backend)	July 23, 2020	All members
Integration (Frontend + Backend)	July 24, 2020	All members
Video Recording (Two 30 minutes videos)	July 26, 2020	All members

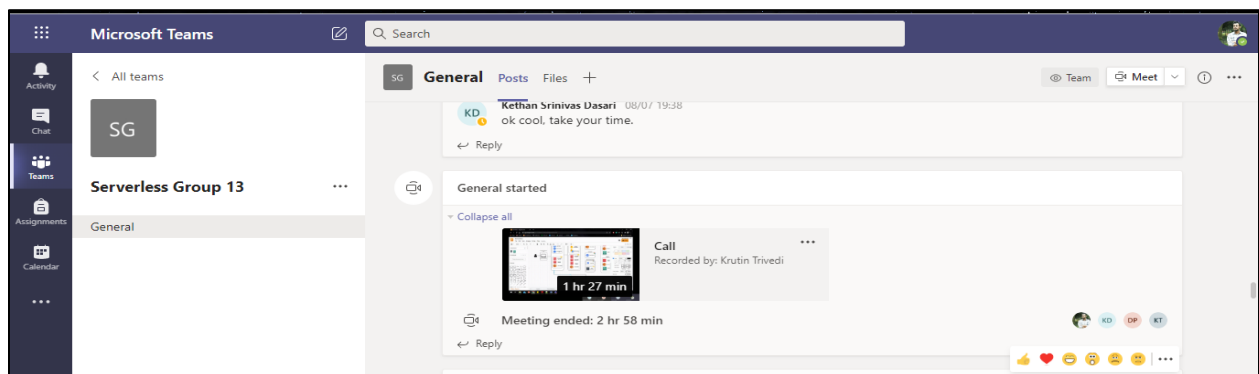
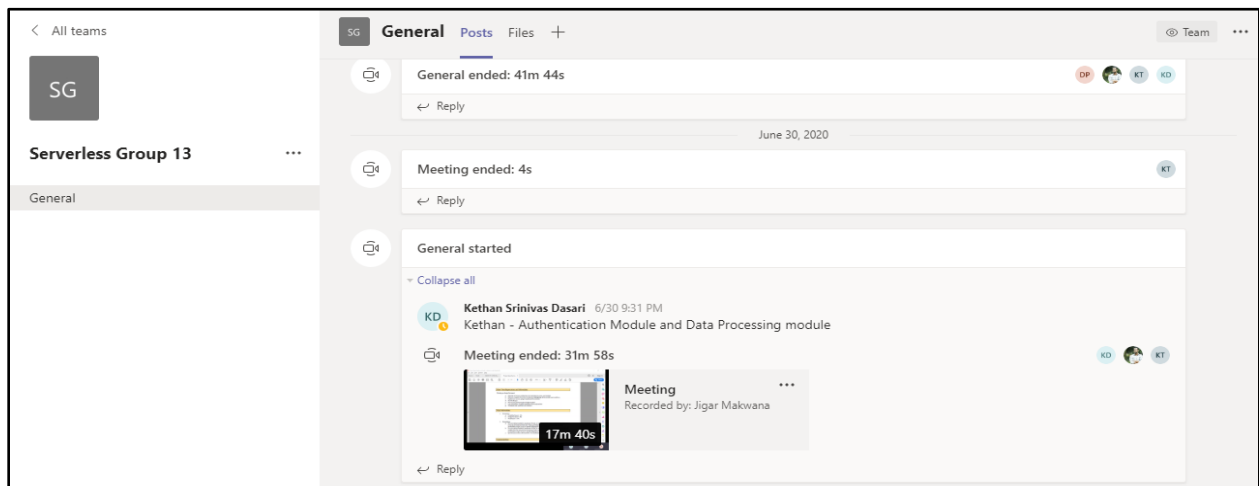
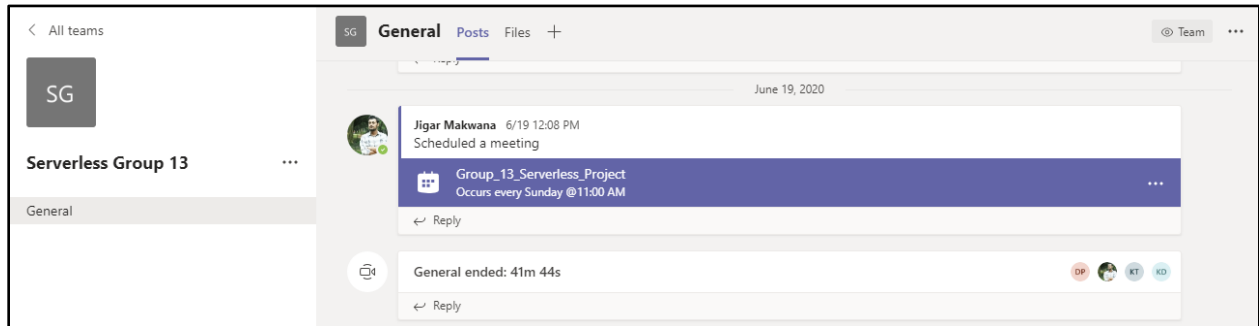
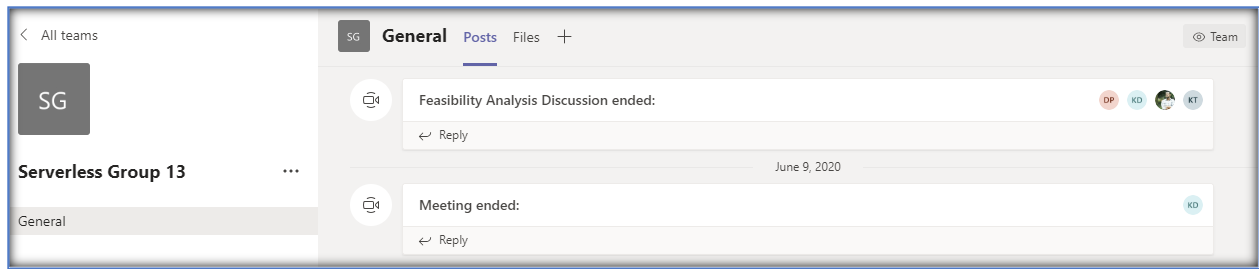


Deployment <b>(Frontend + Backend)</b>	July 26, 2020	Devarshi
Demo <b>(5:30 to 6:00)</b>	July 28, 2020	All members
Documentation	July 29, 2020	All members
Report submission	July 31, 2020	All members

## A.2 MS Teams logs

The following are the screenshots of the online meetings conducted on team collaboration platform Microsoft Teams.





Microsoft Teams

Search

Activity

Chat

Teams

Assignments

Calendar

< All teams

SG

Serverless Group 13

General

SG: General Posts Files +

Team Meet

Am I audible now?

Wait for me

Meeting ended: 34 min 23 sec

Reply

Jigar Makwana 20/07 18:11

Date & Time

July 20, 2020. 5:30 to 6:00 pm

1

Microsoft Teams

Search

Activity

Chat

Teams

Assignments

Calendar

< All teams

SG

Serverless Group 13

General

SG: General Posts Files +

Team Meet

General ended: 34 min 38 sec

15 min 46 sec

Call Recorded by: Kethan Srinivas D...

Reply

## References

- [1] S. Dey, "Project Specification – Version 2.0 (Final Version)," Dalhousie University, Halifax, 2020.
- [2] Facebook, "ReactJS," [Online]. Available: <https://reactjs.org/>. [Accessed 06 August 2020].
- [3] Amazon, "Amazon Web Services," AWS, [Online]. Available: <https://aws.amazon.com/>.
- [4] Google, "Google Cloud Platform," GCP, [Online]. Available: <http://cloud.google.com>.
- [5] AWS, "Amazon Lex," [Online]. Available: <https://aws.amazon.com/blogs/machine-learning/integrate-your-amazon-lex-bot-with-any-messaging-service/>. [Accessed 06 08 2020].
- [6] Google, "Cloud Functions," GCP, [Online]. Available: <https://cloud.google.com/functions/docs>. [Accessed 06 08 2020].
- [7] Google, "Pub Sub," GCP, [Online]. Available: <https://cloud.google.com/pubsub/>. [Accessed 06 08 2020].
- [8] R. w. cloud, "React Word Cloud," [Online]. Available: <https://www.npmjs.com/package/react-wordcloud>. [Accessed 06 08 2020].
- [9] Amazon, "AWS Amplify," [Online]. Available: <https://aws.amazon.com/amplify/>. [Accessed 06 08 2020].
- [10] G. 13, "Draw.io," [Online]. Available: <https://www.diagrams.net/>. [Accessed 08 07 2020].
- [11] AWS, "Boto3," Amazon, [Online]. Available: <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>. [Accessed 06 08 2020].