



CSCI 5409 – FINAL PROJECT REPORT

Group Name – Apes Together Strong

Instructor – Dr. Lu Yang

Dhruvil Amish Shah (B00857606)
Jaspreet Kaur Gill (B00879409)
Nachiket Niranjanbhai Panchal (B00882397)
Samiksha Narendra Salgaonkar (B00865423)

1. Application Overview

In modern times, travel has become incommodious due to plenty of information available on the Internet. Excessive information leads to confusion and affects decision-making. These elevate the need for an application that helps in planning and managing vacations and business trips. **TravelMemories** is a web application that helps its users to plan trips for their leisure time or business easily. The application deals with multiple travel destinations and manages the reviews of each of the destinations with the help of comments and ratings. User Authentication, Posts Management, Security, and Performance are the key features of the web application. Each of the features handles the tasks given below:

- **User Authentication:** The User Authentication feature deals with the creation of user profiles (registration), logging in to the user account, logging out of the user account, and deleting user profiles in the web application.
- **Post Management:** The post-management feature deals with viewing all the posts posted so far, viewing details of a specific post, viewing the posts created and updated by a specific user account, creating a new post, updating an already existing post, and deleting a post, commenting on posts about the details of the post, and rating the posts. The posts here will be images that will be posted by the users in the web application.
- **Security:** The web application developed will be deployed taking into account the security of the data and services being used by the users. The frontend and backend integration will be established via Amazon Virtual Private Cloud ensuring that the connection is made to a private cloud. Moreover, AWS Secret Manager ensures the security and privacy of the application data.
- **Performance:** The performance of the application is ensured using the Elastic Load Balancing service of AWS. This manages the load of many users on the web application.

The web application will be available for use as software and hence, the delivery model used will be Software as a Service (SaaS). The users will use the services directly, they will neither have any access to the building blocks of the web application such as the infrastructure components and modules, nor the platform to build the web application.

The AWS services used in the application are **Amazon Cognito**, **Amazon Simple Storage Service (S3)**, **Amazon DynamoDB**, **AWS Lambda**, **GCP App Engine**, and **API Gateways**. Each of these services will be used for developing the backend of the web application. **AWS S3** provides file storage via a web interface, **AWS DynamoDB** offers document storage via the web interface. **AWS Cognito** will be used to handle the user authentication and management module. **GCP App Engine** is a service for load balancing that guarantees the efficient performance of the web application irrespective of the number of active users using the web application. The application will be deployed using **GCP App Engine** where the frontend and the backend will be deployed using different servers over the Google Cloud.

TravelMemories is a web application that will help people to view any travel destinations of their choice to plan their vacations and travel in a hassle-free manner.

2. Responses to the questions asked

1. Business Considerations:

- What would your organization have to purchase to reproduce your architecture in a private cloud while providing relatively the same level of availability as your cloud implementation? Try to give a rough estimate of what it would cost, don't worry if you are far off. These systems are complicated and you don't know all the exact equipment and software you would need to purchase. Just explore and try your best to figure out the combination of software and hardware you would need to buy to reproduce your app on-premise.

Response: To implement the architecture of the **TravelMemories** in a private cloud, our organization needs multiple services to set up and purchase while maintaining the same degree of availability of cloud implementation. The organization can either use the AWS Virtual Private Cloud (VPC) or can have its own on-premise infrastructure to develop and deploy the application. AWS VPC allows the deployment of the AWS resources into the defined virtual network that closely resembles the regular network in the owned data centre, but with the added benefit of the AWS scalable infrastructure. No additional costs are associated with setting up and using the VPC. After creating the virtual private gateway to establish a VPN connection to your VPC, you will be charged for each "VPN Connection-hour". Every half-hour of VPN Connection usage will be invoiced as a full hour. According to our analysis, using AWS for current demands costs between \$1500 to \$2000 per month. On the other hand, if the organization has to build its own private cloud infrastructure then the organization has to develop it from scratch. To begin, the organization will require virtual machines and physical servers to deploy and run the application. The system must be scalable and dependable. The bare hardware must then be transformed into the private cloud using one of the OpenStack or CloudStack options. The crucial thing to remember is that the private cloud architecture will use the platform-as-a-service (PaaS). All of the essential services such as Cognito, API Gateways, Lambda functions, DynamoDB, S3, secret manager, and many more need to be configured. Before setting up the infrastructure, the organization has to plan ahead of the compute services required for their requirement. It will be a wastage of money if the compute services are exceeding the expected requirement and the organization is wasting money on unused services. Also, if the compute services will be less as compared to the requirement, then the organization cannot host the application that handles the traffic over the internet. Besides all this, the organization has to spend money on maintaining and managing the infrastructure. According to our estimates, the monthly cost of establishing and maintaining the running infrastructure to run and maintain the **TravelMemories** with equivalent reliability and availability would be roughly 4000 to 5000 dollars. This is the very approximate estimate, but the increased cost is due to the fact that a private cloud is built and maintained from the scratch rather than using a pay-as-you-go cloud provider. It is impossible to cover all of the aspects involved in constructing and maintaining a private cloud for managing the **TravelMemories** application in great detail, but I have attempted to touch base on the most critical and vital ones.

- Which cloud mechanism would be most important for you to add monitoring to in order to make sure costs do not escalate out of the budget unexpectedly? In

other words, which of your cloud mechanisms has the most potential to cost the most money?

Response: TravelMemories application allows the registered users to upload the posts, add comments, and rate the posts. So, our application's most features are relying on AWS DynamoDB and AWS S3 storage. So, DynamoDB and AWS S3 mechanisms of the AWS cloud provider are the essential cloud mechanisms for our website. As a result, it is crucial that we keep an eye on the DynamoDB and S3 services to ensure that they do not go over budget. Other than that we have deployed our application on the Google App Engine that is load balancing the application automatically and makes the website scalable. If the incoming traffic increases, the application needs to run smoothly otherwise the users would not be able to access the application leading the business loss for the owner of the application. So, the Google App Engine service provided by the google cloud is also crucial for the application and this cloud mechanism needs to be monitored as there are chances that it can also consume the credits when traffic over the internet increases and many users are visiting the application URL.

2. Describe your final architecture:

- How do all of the cloud mechanisms fit together to deliver your application?

Response: The various cloud mechanisms causing the key requirements of the web application i.e. Compute, Storage, Network, Security, and Serverless Computing are the cloud services utilized in the project for implementation. Each of these services fit as a fiddle to deliver an end-to-end cloud-based web application. **GCP App Engine** and **AWS Lambda** handle the compute part of the project for deployment to be made available remotely, **AWS Cognito** manages the security of the application by supporting the User Authentication and User Management module efficiently thereby ensuring the security of the cloud application. **AWS S3** and **AWS DynamoDB** support the storage of the web application. **AWS API Gateway** to create routes to route to the lambda function and navigate to the front-end of the application. All these services together help in building the essential services of the application. These services, however, form the backend of the **TravelMemories** applications. The front end of the application is built using React.js. The user will be able to view the application as a web-based service. The key functionality of the **TravelMemories** application will be handled by Amazon Web Services. The frontend of the application is a web application where the user will be offered use cases such as registering into the application, logging into the application, managing or updating details in the application, deleting user account, creating a travel memory, viewing travel memories, rating a travel memory, and commenting on a travel memory. The backend services will handle the primary functionalities as stated. The communication channel between the frontend and backend is established using the Application Program Interfaces (APIs). These APIs are used to post, fetch, and update the data stored on the cloud (**AWS S3** and/or **AWS DynamoDB**). The data will be posted or updated to the backend with the help of the user interface and fetched from the backend and rendered on the user interface for the user to view the requested data. The process and data flow between the various components of the application are established with the integration of all the cloud-based services with the responsive user interface to allow seamless delivery of services of the **TravelMemories** application.

- Where is data stored?

Response: Data in the TravelMemories application is organized to store the user details such as first name, last name, email address, password, and address in AWS Cognito. These details are stored to AWS Cognito directly using Node.js SDK named **client-cognito-identity-provider** for AWS Cognito. User details stored in AWS Cognito during registration are used for authentication. Additionally, the travel memory posts created by the users require to store post details such as post image, post name, the location where the image was captured, zip code of the location, description of the post, comments made by the users on the post, and the ratings provided by the users on the post. All the post details in the text will be stored in Amazon's DynamoDB database whereas the image of the post is stored in the S3 bucket i.e. the cloud storage service offered by Amazon. Upon uploading the post image on the AWS S3 bucket, a URL of the image is passed on to the table in DynamoDB where the image URL is updated for the post. The comments on the posts are stored in a separate table in DynamoDB. The database and storage components of the TravelMemories application is designed keeping in mind the best practices thereby avoiding any complexities, interdependencies and tight coupling between components of the application. The security policy for the S3 bucket containing the post images is updated to the public to make get and post on the application possible. The data flow for the TravelMemories application passes from the user providing the information from the frontend on events such as click, hover and many more. These events trigger functions that call Application Program Interfaces (APIs) for establishing a connection with the backend cloud services. The APIs used are GET, POST, and PUT in order to fetch, send, and update data to AWS S3 and AWS DynamoDB.

- What programming languages did you use (and why) and what parts of your application required code?

Response: The frontend of the web application is developed using the React.js framework and the backend of the application is developed using Node and Express. The programming language primarily used for implementing the **TravelMemories** application is JavaScript. All the cloud-based services hosted by Amazon utilized for building this project are integrated with the frontend programmatically using JavaScript and the libraries provided by it in the Node framework. Precisely, the use of **AWS Cognito**, **AWS S3**, and **AWS DynamoDB** is programmatically done using JavaScript and Node framework.

- How is your system deployed to the cloud?

Response: The system is deployed on the app engine of the Google Cloud Platform. To deploy the frontend and backend on the App Engine, two projects are created in the google cloud. One project is to deploy the backend of the application written in Node.js while the other is for deploying the frontend of the system developed in React.js. The backend and frontend of the application are running on different servers. Serever.js file is written in the backend code that is using the front end routes for navigation and displaying the frontend pages. App engine is decided to go for hosting the web application as it handles the server provisioning and the scalability for the application instances based on the demand.

3. If your **final** architecture differs from your original project proposal, explain why that is. What did you learn that made you change technologies or approaches?
 - a. If your final architecture is the same as your original project proposal, reflect now on whether there is something you could have implemented with what you know now that would have resulted in a better architecture or a more cost-efficient product?

Response: The final architecture of the TravelMemories application differs from the architecture submitted during the project proposal. Instead of using the AWS EC2 and AWS Elastic Load Balancer, we have decided to go with Google App Engine to deploy and host the application on the cloud. This is decided because the Google App Engine helps to run the application over multiple servers. Also, the Google App Engine automatically scales the application along with load balancing the traffic by managing the incoming traffic over the internet. The application is designed to weigh the pros and cons of the cloud services offered by numerous cloud service providers such as Amazon, Google, and Microsoft. Another feature that can be implemented with the existing architecture of the TravelMemories application is searching for a post by name or a keyword. Additionally, a messaging service can be integrated with the application where the user gets notified when another user comments or rates on his/her travel memory post. The scope of the application will broaden dynamically aligning to the changing needs and competitive capabilities of similar application brands. The TravelMemories application is designed keeping in mind the cost and performance comparisons of the services offered by cloud service providers.

4. How would your application evolve if you were to continue development? What features might you add next and which cloud mechanisms would you use to implement those features?

Response: The **TravelMemories** is developed with basic features such as registration, login, reset the password, upload a post, rate, and comment on the post by using the cloud services. This application can be enhanced further by adding new features. There are multiple features that can be integrated with **TravelMemories** to improve the competitive edge. We are planning to add the next feature where users can search the post based on the post name. As of now, users can add comments to the post but in future adding the like or dislike option can also be added to the posts. Complex features can be added to the application that improves the user experience like based on the uploaded posts, the application will start suggesting the places to the user that the user can visit or travel in the future. This is a complete machine learning feature, where the application will consider the posts uploaded by the user and based on the preferences of the user, the system will provide suggestions to the user. This feature will be implemented using the AWS Sagemaker service provided by Amazon Web Services.

3. Implementation Highlights

3.1 Process Flow

The process flow for the various components of the **TravelMemories** application is described below:

3.1.1 User Authentication and Management

This module contains features like registration, login, and forget the password. Any new user who wants to use the features of the application has to first register to the application by entering the details such as first name, last name, email id, address, and password. On successful registration, the email address of the user is verified by sending the code to the registered email address. The user has to enter the code to verify its email. After email verification, the user can use the login feature of the **TravelMemories** to start utilizing the post-management module. Users can log in by entering a registered email address and password. User can also reset their passwords in the case when the user forgets its password. To reset the password, the user has to click on forget password option on the login page and it will ask the user to enter the registered email address to send the code. The user has to enter the code along with the new password on the reset password page. Now, the user can log in to the application with the new password. The user management module is developed using the Node.js SDK of AWS Cognito storing the entire user details in Cognito. The user details are stored in Cognito while the user is registering to the application. The stored user details are utilized when the user logs in to the application. In the same way, during password reset, the new password set by the user is stored in the AWS Cognito across the same user email address.

The figure displays the registration page of the **TravelMemories** asking for the user details before registering the user (see Figure 1)

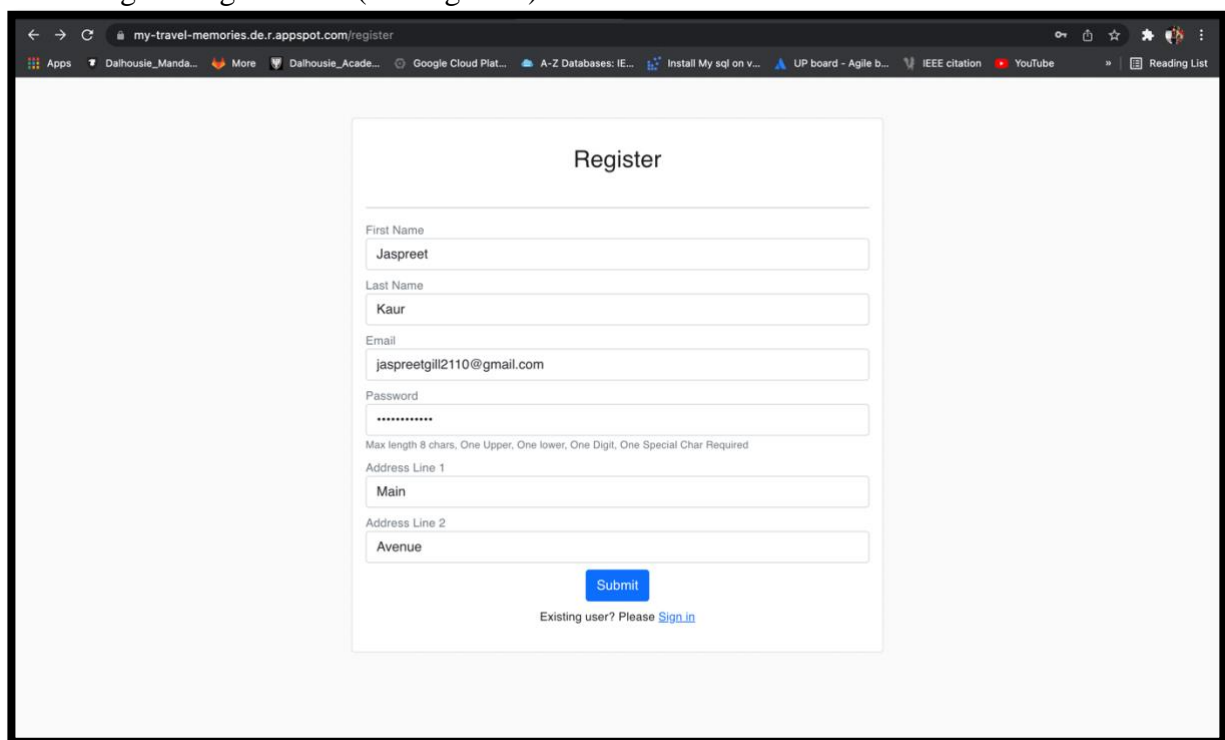
The image shows a web browser window displaying the registration page of an application named 'TravelMemories'. The browser's address bar shows the URL 'my-travel-memories.de.r.appspot.com/register'. The page has a light gray background. In the center, there is a white rectangular box with the title 'Register' at the top. Below the title, there are several input fields: 'First Name' with the value 'Jaspreet', 'Last Name' with the value 'Kaur', 'Email' with the value 'jaspreetgill2110@gmail.com', 'Password' which is masked with dots, and two address fields: 'Address Line 1' with the value 'Main' and 'Address Line 2' with the value 'Avenue'. Below these fields is a blue 'Submit' button. Underneath the button, there is a link that says 'Existing user? Please Sign in'.

Figure 1: Registration page of the TravelMemories [1]

On successful registration, the user will be redirected to the email verification page. The user will receive a verification code on the registered email address. The received code needs to be added in the text box asking for the verification code and then the user can click on verify button to verify the email address. If the user wants to resend the code, then click on resend button to receive a new verification code. Figure 2 displays the email verification page of the **TravelMemories**.

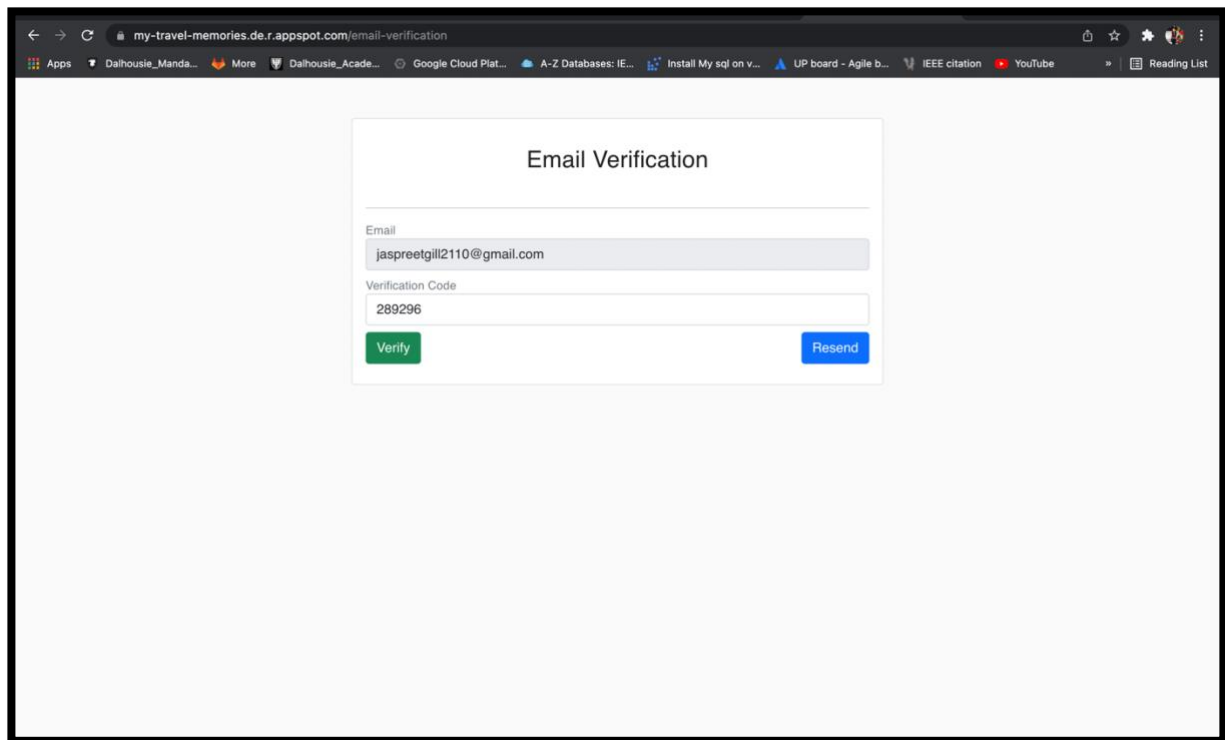


Figure 2: Email Verification page of the TravelMemories

On successful verification of an email, the user will be redirected to the login page. Users can enter the credentials like email addresses and passwords to log in to the application (see Figure 3).

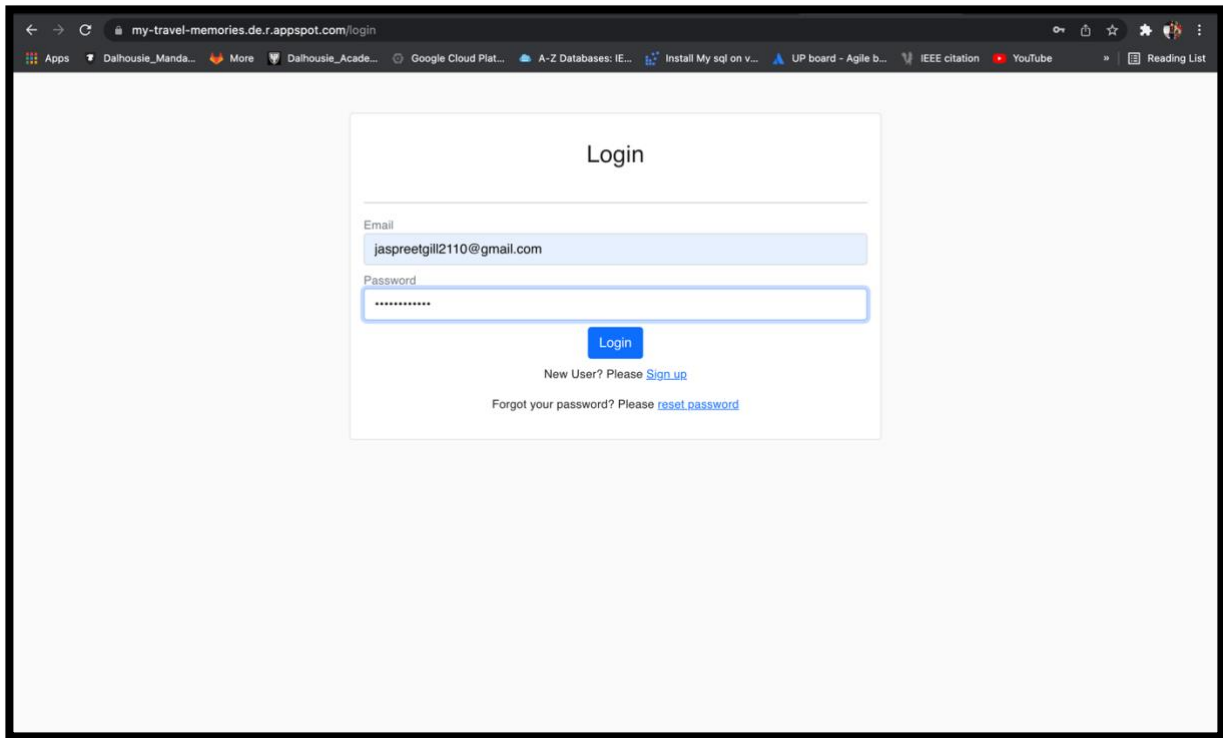


Figure 3: Login page of the TravelMemories

In case, the registered user forgets the password, the user can reset the password by clicking on the reset password link. The user has to enter the registered email address and then click on the get code button. A verification code will be received at the registered email address. The user enters the verification code, the new password, and then clicks on the update button. The new password will be updated in the AWS Cognito for the respective user. Now, the user logs in to the application using the new password. Figure 4 shows the reset password page.

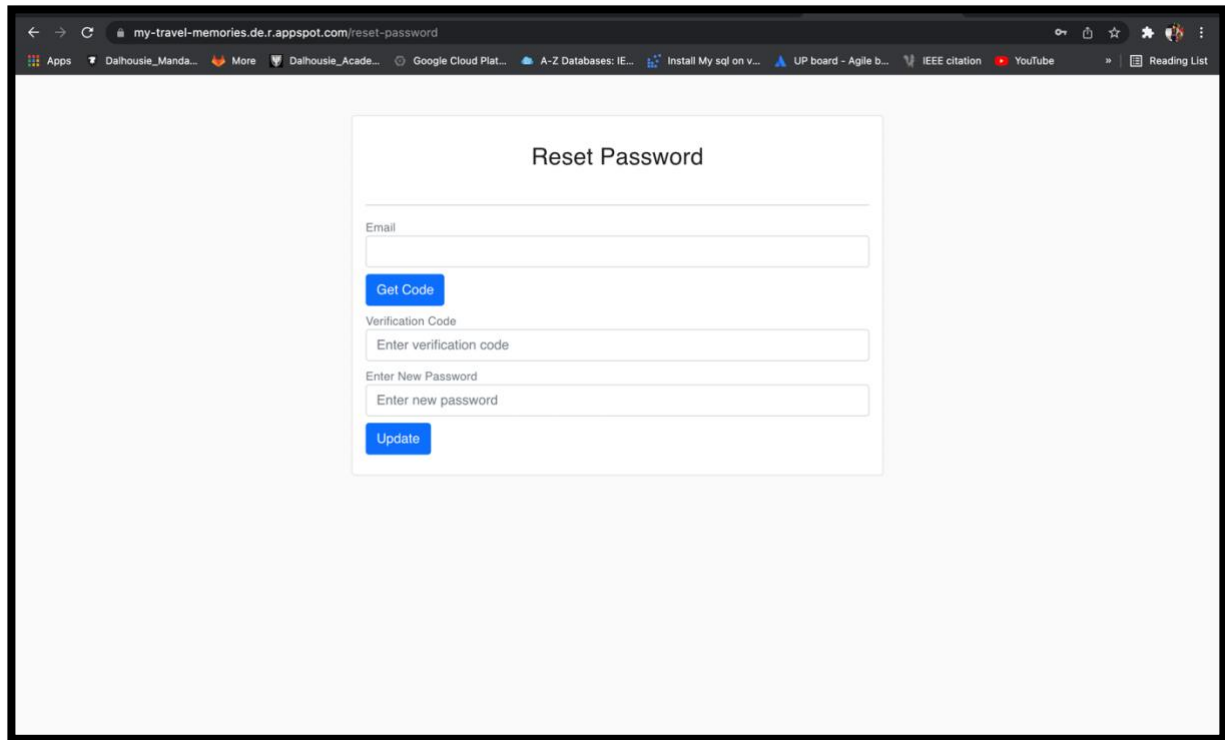


Figure 4: Reset password page of the TravelMemories

3.1.2 Post Management

The Post Management module offers the user to create a travel memory by uploading an image of the travel destination the user has visited along with the details such as the name of the travel memory post, description, location, and zip code of the travel destination. The same is achieved using the create post feature. The post details are sent to the AWS S3 and AWS DynamoDB cloud services using the POST and PUT APIs. The post details posted for this feature are stored in a single table in the AWS DynamoDB database.

Figure 5 displays the creation of a travel memory post using the TravelMemories web service.

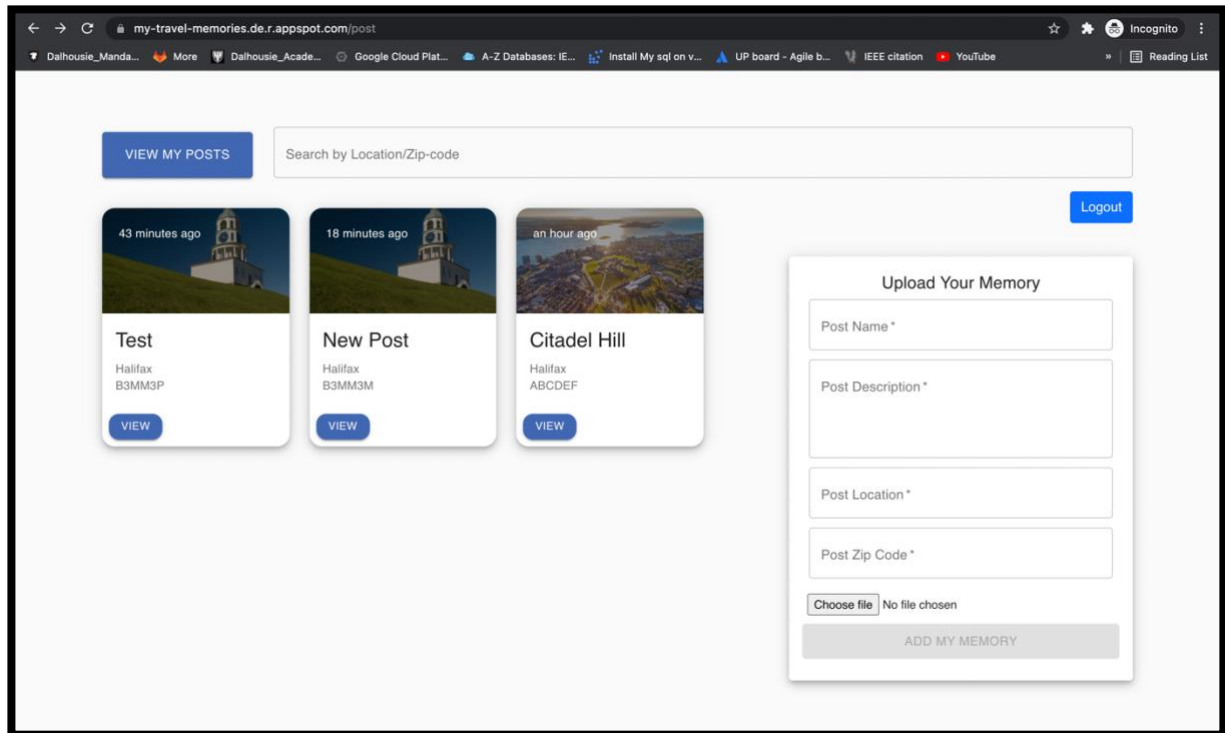


Figure 5: Travel memory creation page of TravelMemories

The users can also view the added posts in the TravelMemories application. The post details are fetched using the GET API from the AWS S3 and AWS DynamoDB cloud-based storage and database system. The figure displays the user interface for viewing the travel memory posts.

Additionally, TravelMemories allows its users to rate and comment on the travel memory posts created. The ratings and comments on a travel memory are stored in separate tables in the AWS DynamoDB database. The ratings and comments feature, each has a GET and POST API to allow the users to view the average ratings provided to the travel memory post and comment on the travel memory post.

Figure 6 displays the ratings and comments made on one of the travel memory posts.

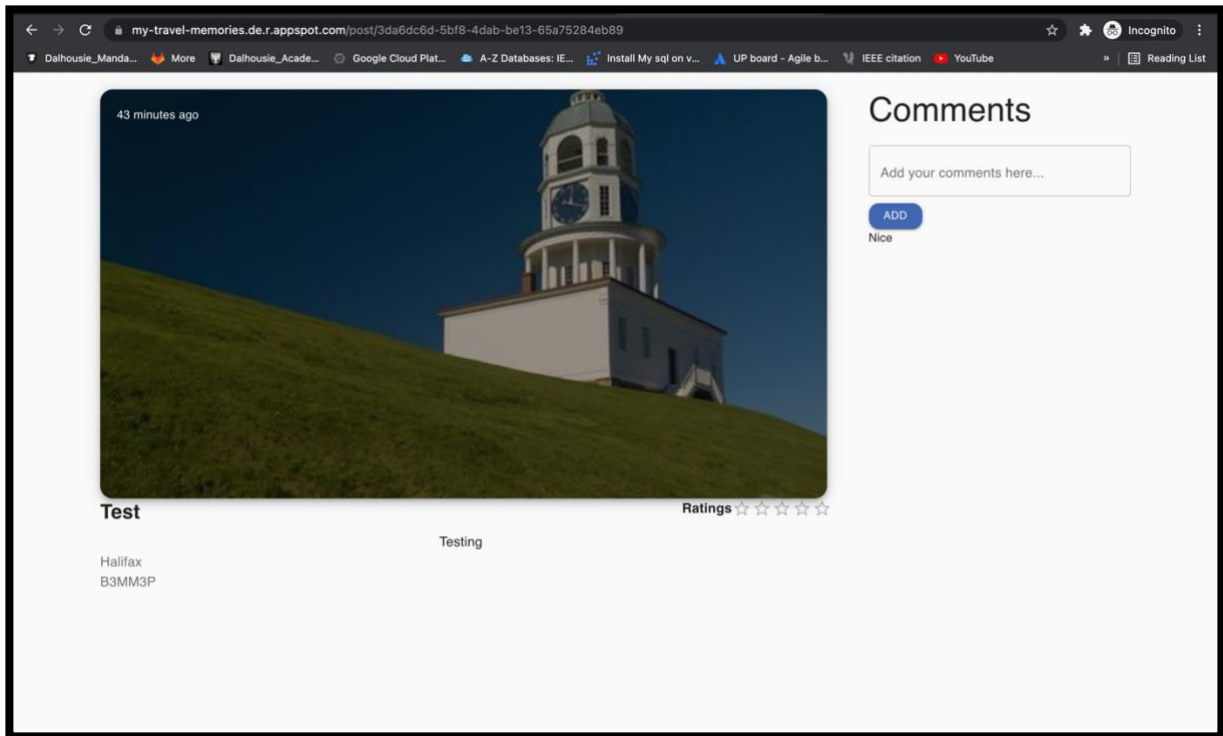


Figure 6: Rating and comment feature of the TravelMemories

3.2 Data Flow

The Data flow for the **TravelMemories** application is mainly handled by two key cloud services provided by Amazon i.e. Amazon Simple Storage Service (S3) and Amazon DynamoDB.

3.2.1 User Authentication and Management

The details of the user such as first name, last name, email address, password, and address are stored in the AWS Cognito. For user authentication and authorization, the flow of the data is to and from the AWS Cognito. The data flow of the user authentication and authorization module is shown in Figure 7.

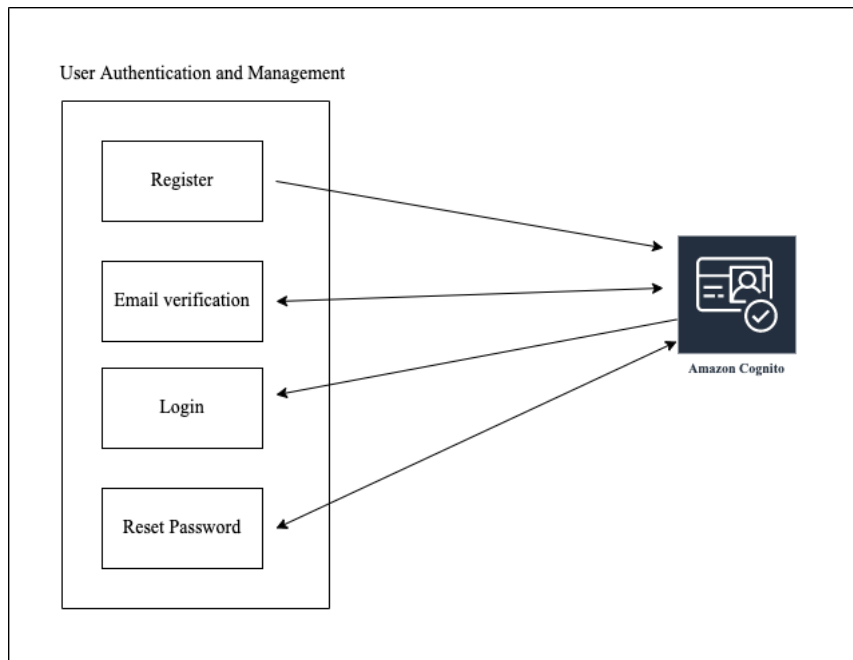


Figure 7: Data flow for user authentication and management module

3.2.2 Post Management

Amazon's DynamoDB database stores data related to the travel memory posts in three tables **travel_memory_post**, **travel_memory_post_comment**, and **travel_memory_post_rating**. The **travel_memory_post** table stores details about the post, **travel_memory_post_comment** stores details about the comments made on the posts, and **travel_memory_post_rating** stores the ratings on the posts made by the user. The data flow for the project is as described in Figure 8.

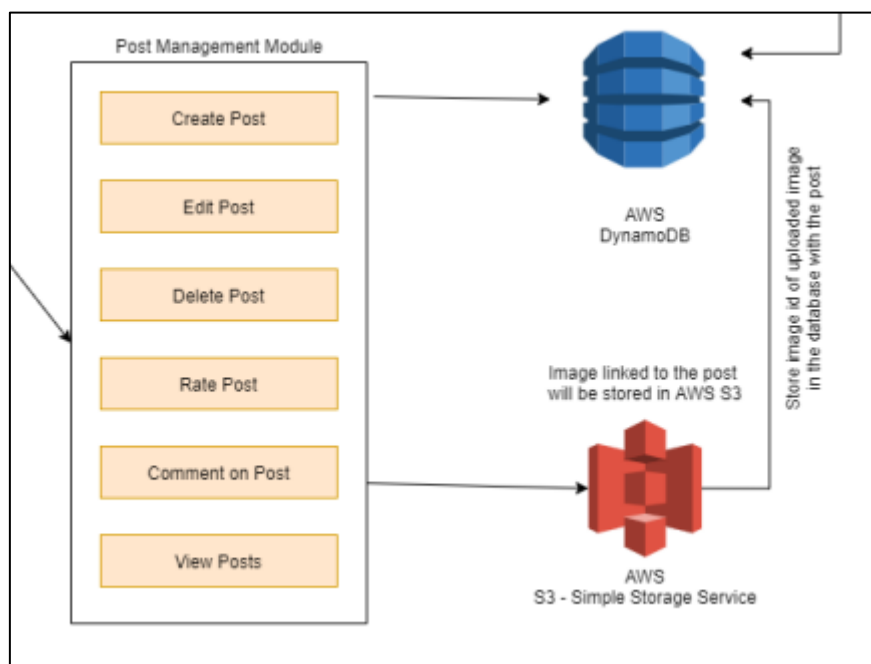


Figure 8: Data flow of the post-management module

All the text-based post data is stored in AWS DynamoDB and post images in AWS S3. The post images as uploaded by the users is stored in AWS S3 and the image URL is stored in AWS DynamoDB. The table definition of the tables in AWS DynamoDB looks like this:

Table: travel_memory_post

postId: {S: Primary Key} - UUID

userId {S}

postName {S}

postDescription {S}

postLocation {S}

postZipCode {S}

postImageURL {S}

createdAt {S}

Table: travel_memory_post_comment

commentId {S: Primary Key} - UUID

postId {S}

userId {S}

comment {S}

Table: travel_memory_post_rating

ratingId {S: Primary Key} - UUID

postId {S}

userId {S}

rating {S}

The **S** stands for type String and **UUID** to generate a unique random string (Primary Key).

REFERENCES

- [1] “VPC: Validación Periódica de La Colegiación,” *Amazon*, 2016. [Online]. Available: <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>. [Accessed: 10-Dec-2021].
- [2] “Five steps to building a private cloud,” *Network Computing*, 10-Jan-2019. [Online]. Available: <https://www.networkcomputing.com/cloud-infrastructure/five-steps-building-private-cloud>. [Accessed: 10-Dec-2021].
- [3] “How is Amazon VPC charged?,” *Intellipaat Community*, 28-Oct-2020. [Online]. Available: <https://intellipaat.com/community/58652/how-is-amazon-vpc-charged>. [Accessed: 10-Dec-2021].
- [4] “Diagrams.net - free flowchart maker and diagrams online,” *Flowchart Maker & Online Diagram Software*. [Online]. Available: <https://app.diagrams.net/>. [Accessed: 10-Dec-2021].