CSCI 5410

Serverless Data Processing

DALVacationHome

Group 6: Sprint-1 Report

Group Members:

Nikita Davies	nk548914@dal.ca
Mrunal Mangesh Patkar	mr396180@dal.ca
Rameez Parkar	rameez.parkar@dal.ca
Axata Darji	ax583820@dal.ca

GitLab Link: https://git.cs.dal.ca/ndavies/csci5410-s24-sdp-6

Table of Contents

Background Research and Learning

Topic: Cloud Application based Research

AWS Services

Summarization

AWS Cognito

- AWS Cognito handles user authentication and authorization for web and mobile applications.
- AWS Cognito user pool will allow users to sign-up and then sign-in with user id and password and has the forgot/reset password feature.
- Cognito provides the option of choosing whether to use email or user id or phone number as credentials. It also allows the admin to set the password conditions, whether to generate a random password for account creation or set a predefined password, etc.

Primary Reference:

- AWS Official Documentation – Amazon Cognito [1] https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html

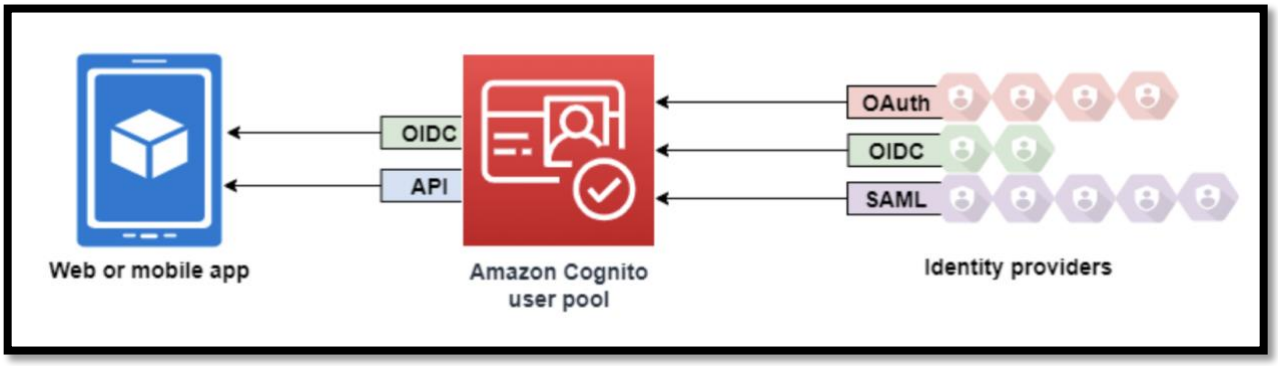

Figure 1: User authentication by AWS Cognito [1]

AWS DynamoDB

- AWS DynamoDB is a serverless, NoSQL database with no server provisioning, patching, or management required. There's no software to install, maintain, or operate, with zero downtime maintenance.
- In our project, DynamoDB would be the main source of data storage where we would store all of user profile information, and the security questions and answers for each specific user.
- Also, the encrypted cipher keys can be store in DynamoDB.

- We can have the following 3 tables for the authentication scenario:
    - **Users table:** Stores user profile details such as UserID, Username, Email, PhoneNumber, and registration timestamps.
    - **UserSecurityQA table:** Stores security questions and hashed answers, indexed by UserID and QuestionID.
    - **UserCiphers table:** Stores Caesar cipher keys and encrypted data associated with each UserID.
    - **Registration table:** Stores the registration details of a UserID against the room they've booked.
- DynamoDB will also contain information about reservations which can be accessed by a registered customer via Lex bot integration or normal frontend view.

Primary Reference:

- AWS Official Documentation – Amazon DynamoDB[2]

  https://aws.amazon.com/pm/dynamodb/

AWS Lambda

- AWS Lambda is a serverless, event-driven compute service that lets user run code for virtually any type of application or backend service without provisioning or managing servers.
- We have decided to go with Node.js v20 as our preferred language for building backend services on AWS Lambda.
- In our DalVacationHome application, AWS Lambda will connect with DynamoDB to fetch the necessary data and then process it on the Lambda functions.
- The following is a high-level overview of the steps we will require for a lambda function and how we plan to execute it:
    - **User Registration:** Lambda function saves user details to DynamoDB.
    - **Second Factor Authentication (Question/Answer)**: Lambda function retrieves and validates security questions and answers from DynamoDB.
    - **Third Factor Authentication (Caesar Cipher)**: Lambda function retrieves cipher key and data from DynamoDB, performs Caesar cipher decryption, and validates the user's input.
    - **Handling Queries via Lex**: Lambda function processes and responds to user queries from AWS Lex, such as "how to register?" or "details about booking id #", etc.
    - **Searching Room Numbers and Stay Duration**: Lambda function queries DynamoDB for room numbers and stay details based on booking ids.
    - **Customer Communication**: Lambda function processes customer queries, concerns and necessary communication like booking notifications and forwards them via email to customers using AWS Simple Notification Service.

Primary Reference:

- AWS Official Documentation – AWS Lambda [3]

  https://aws.amazon.com/lambda/

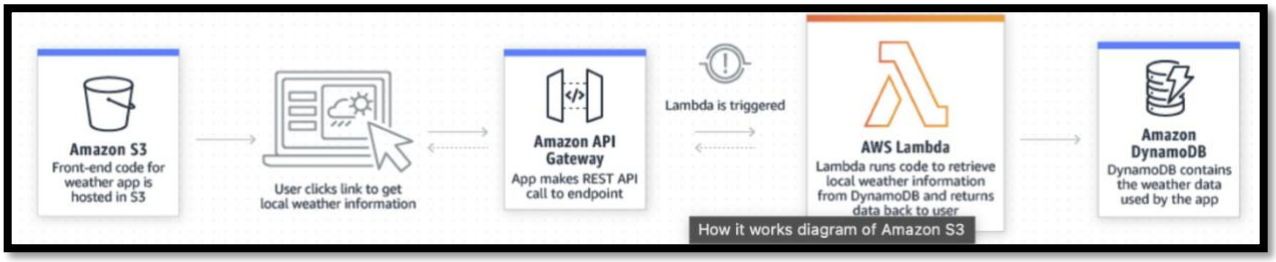

Figure 2: Usage of AWS Lambda[3]

AWS Lex

- AWS Lex is a chatbot service provided by AWS that can be fully customized according to specific business needs.
- In our project, we will be using AWS Lex v2, since it has significant improvements as compared to v1 in terms of session and context management, having multiple bots with a single model, improved multi-language support, etc.
- The interface to communicate with the Lex chatbot using its APIs will be built in React.
- We will train our application to answer to basic customer queries, provide reservation details, assist them with their account or profile information and help them with information to reach out to property agents.

Primary Reference:

- AWS Official Documentation – Amazon Lex 1[4]

  https://docs.aws.amazon.com/lex/latest/dg/what-is.html

AWS SNS

- Simple Notification Service (Amazon SNS) is a managed service that provides message delivery from publishers to subscribers. We will be using these services to notify users regarding various actions they perform in the application.
- Simple Notification Service contains many options to provide notifications to users such as email, SMS (text messages), and mobile push notifications. We will use the email option for our application.
- The registered customers will be getting notifications for successful sign-up and sign-in, booking success and booking failure on customer's registered email ID.
- The SNS topic will be triggered to send an email via Lambda function, as the Lambda function would use the SDK to call the specific topic with the email message and placeholders to send the

email.

Primary Reference:

- AWS Official Documentation – Amazon Simple Notification Service [5]

  https://docs.aws.amazon.com/sns/latest/dg/welcome.html

Research on deploying serverless web applications on AWS services

- The research paper explores the implementation of serverless computing to enhance cost efficiency and operational speed for organizations.
- Traditional on-premises server computing requires significant infrastructure investment, adding a financial burden for companies. In contrast, serverless computing, managed by cloud service providers like AWS, eliminates the need for physical server management. The paper highlights the benefits of serverless computing, including reduced expenses and increased reliability, as it allows developers to concentrate on code and business logic without worrying about server maintenance.
- The paper details the use of various AWS services to deploy a serverless web application, focusing on four main services: Amazon API Gateway, AWS Lambda, Amazon DynamoDB, and Amazon S3. Amazon API Gateway manages API calls and handles functions such as traffic management and authorization. AWS Lambda executes code in response to events, providing scalable and cost-effective computing power. Amazon DynamoDB offers a fully managed NoSQL database service, ensuring high performance and scalability. Amazon S3 serves as an object storage service, securely storing and retrieving large amounts of data. The paper provides a practical guide to setting up these services, creating APIs, and integrating Lambda functions to manage backend processes efficiently.
- The paper also discusses various cloud computing security challenges and solutions. It highlights the advantages and ongoing developments in serverless computing, emphasizing the potential for further research and innovation in this field. The proposed methodology outlines a step-by-step approach to creating and deploying a serverless web application, demonstrating the process through practical examples. The conclusion underscores the cost-saving benefits and efficiency improvements achieved through serverless computing, making it an attractive option for modern digital management. The authors advocate for the adoption of serverless architecture, predicting

Primary Reference:

- Research Paper: "**Deployment of a Serverless Web application using AWS Services**" [6]

  https://ajse.us/wp-content/uploads/2023/03/AJSE-V3-I4_paper1.pdf

GCP Services

GCP Pub/Sub

- Pub/Sub is a messaging middleware that enables asynchronous communication
- In our application, GCP Pub/Sub will be used for the message-passing module , which facilitates asynchronous communication between customers and property agents.

  Planned Implementation:

- **Publishing Messages:** When a customer submits a concern or query (based on a booking reference code), it will be published to a topic in Pub/Sub.
- **Subscribing to Messages:** A subscriber function which would be a cloud function will be triggered upon receiving a message in the topic. The trigger type of the Cloud function is set as pub and sub.This function will forward the message to a randomly selected property agent for response.
- **Logging Communication:** All messages and their responses will be logged in Firestore for tracking and record-keeping purposes.
- Firestore is selected for logging over DynamoDB as firestore is a native GCP service and integrates seamlessly with other GCP services such as Cloud Functions, Pub/Sub. This can simplify the overall architecture and reduce latency when services need to interact frequently. While DynamoDB can be integrated with GCP services, it requires additional setup and configuration, potentially increasing complexity.

Primary Reference:

- Google Cloud Official Documentaion – Pub/Sub [7]

  https://cloud.google.com/pubsub?hl=en

- Medium – Pub/Sub in GCP [8]

  https://medium.com/@teja.ravi474/pub-sub-in-gcp-27a21554d158

LookerStudio

- Looker Studio (formerly known as Data Studio) is a powerful data visualization tool by Google that allows users to create interactive dashboards and reports. It is especially useful for aggregating data from various sources and presenting it in an accessible and visually appealing way.
- In our application, Looker Studio will be used for data visualization and analysis, providing a dashboard for property agents to view user statistics and login data.

  Planned Implementation:

- **Data Sources:** Connect Looker Studio to data stored in DynamoDB which contains user and booking data. CData Connect Cloud can be used to gain access to Amazon DynamoDB data.

- **Dashboards:** Create dashboards that display total users, login statistics, and other relevant metrics.
- **Embedding in Frontend:** Embed Looker Studio dashboards within the admin page of the React frontend application, allowing property agents to easily access and visualize data.

Primary Reference:

- LookerStudio – Official Documentation [9]

  https://lookerstudio.google.com/data

Google Natural Language API

- Google Natural Language API is a machine learning-based service provided by Google Cloud that offers powerful tools for text analysis. It can be used to understand the structure and meaning of text, making it an excellent choice for tasks like sentiment analysis, entity recognition, and syntax analysis
- In our application, The Google Natural Language API will be used to perform sentiment analysis on customer feedback.

  Planned Implementation:

- **Feedback Analysis:** When customers submit feedback, the text data will be sent to the Natural Language API for sentiment analysis.
- **Storing Results:** The results of the sentiment analysis will be stored in Firestore.
- **Displaying Results:** The analyzed feedback will be presented in the frontend application in a tabular format, making it accessible to guests, registered customers, and property agents.

Primary Reference:

- Google Natural Language API – documentation [10]

  https://cloud.google.com/natural-language?hl=en

CloudRun

- Cloud Run is a fully managed platform that enables you to run your code directly on top of Google's scalable infrastructure.
- Cloud Run will be used to host the entire React front-end application, ensuring a scalable and serverless deployment.

Planned Implementation:

- **Containerization:** Containerize the frontend application built using React using Docker.
- **Deployment:** Deploy the containerized application to Cloud Run, leveraging its ability to automatically scale based on traffic and its pay-per-use pricing model.
- **API Endpoints:** Set up API endpoints in Cloud Run to interact with various backend services (AWS services for authentication, DynamoDB for data storage, etc.).

Primary Reference:

- Google Cloud Run – documentation [11]

  https://cloud.google.com/run?hl=en

Topic: Architecture Oriented Research

To ensure the success of our DALVacationHome application, it is crucial to understand and implement best practices from existing serverless cloud architectures. By using insights from research papers on serverless architectures, we can enhance the design, performance, and scalability of our application. Here is an overview of key research findings and how they can be applied to the DALVacationHome architecture.

Primary Reference:

- Research Paper: "**Serverless Computing: Design, Implementation, and Performance**" - This paper discusses the benefits and challenges of serverless architectures, providing insights into scalability and cold start latency.[12]

  https://ieeexplore.ieee.org/document/7979855

Key Insights:

Event-Driven Architectures

- **Insight:** Serverless platforms excel at handling event-driven workloads triggered by HTTP requests, database updates, or message queue events.
- **Learning:** Use AWS Lambda and Google Cloud Functions in our application to handle user interactions, such as room booking requests and feedback submissions. Integrate Google Cloud Pub/Sub for seamless event management.

Scalability and Cost Efficiency

- **Insight:** Serverless architectures scale automatically based on demand and follow a pay-per-use model, making them cost-efficient.
- **Learning:** Implement auto-scaling functions for fluctuating workloads, such as user

authentication and virtual assistant queries, to optimize resource usage and reduce costs.

Challenges in implementing Serverless Architecture and how to Overcome:

1. **Complexity in Debugging and Monitoring:**

   - **Challenge:** Debugging and monitoring distributed serverless applications can be complex.
   - **Learning:** Use AWS CloudWatch and Google Stackdriver in our application for comprehensive monitoring and logging. Implement distributed tracing with AWS X-Ray or Google Cloud Trace to diagnose issues effectively.

2. **Application Design Constraints:**

   - **Challenge:** Applications need to be designed to fit a stateless, serverless model.
   - **Learning:** Ensure that application logic is stateless and relies on external storage like AWS DynamoDB or Google Firestore for maintaining state.

3. **Vendor Lock-In:**

   - **Challenge:** Heavy reliance on a single cloud provider can lead to vendor lock-in.
   - **Learning:** Adopt a multi-cloud strategy by using services in both AWS and GCP to build the application

Primary Reference:

- Serverless Architecture , Prisma Data Guide - “Introduction to common serverless challenges“[13].
  https://www.prisma.io/dataguide/serverless/serverless-challenges

Topic: Application Perspective Research

Primary Reference:

- Gorilla Logic Website [14]
  https://gorillalogic.com/blog/winston-a-virtual-hr-assistant-created-with-amazon-lex-and-slack

Summarization

- As a company grows, more and more information is generated. This information comprises HR policies, administrative and technical processes, company benefits and IT know-how. As the resources within the company increase, the need to place them in an easily-accessible repository becomes critical.
- In response to this issue, a few of us at Gorilla Labs decided to create a Slack-integrated chatbot. The chatbot was implemented using Amazon Web Services Lex, and the final solution only incorporated AWS and Slack-provided services. Use of servers was limited, as the cost of the project to is minimum. This chatbot is called “Winston.”
- While Amazon Lex provides both speech recognition and natural language understanding, scope

for this project was to create a text-based chatbot using only Lex's natural language understanding. This functionality makes Winston a useful Slack-integrated application without limiting its ability to one day be integrated it into a speech interface such as Amazon Alexa.

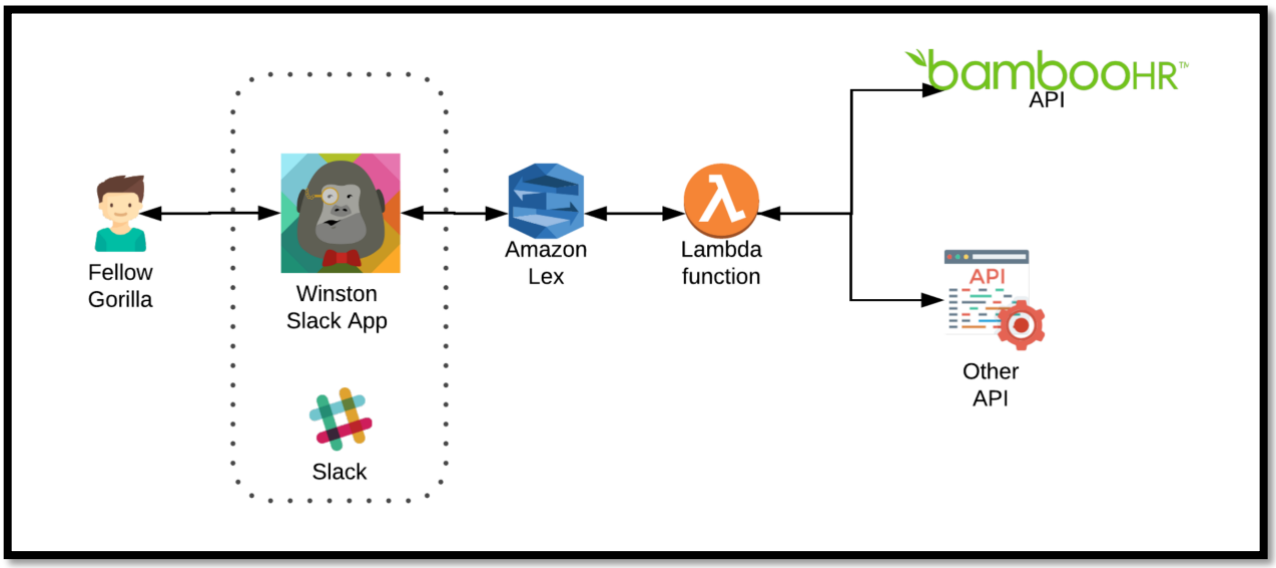

Figure 3: Serverless architecture used in Gorilla Logic Website [14]

Primary Reference:

- Amazon SNS [15]
  https://www.serverless.com/guides/amazon-sns

Summarization

- Sending events to mobile clients via push notifications Mobile push notifications can be useful for sending secure, time-sensitive updates to your users. Push notifications usually rely on a service that maintains a connection between each mobile device subscribed to the notifications and sends any incoming app notifications to the end users. While Amazon doesn't run a service that actually sends the app notifications for you, it does allow you to plug into SNS the following push notification services:
  - Amazon Device Messaging (ADM) for Amazon devices
  - Apple Push Notification Service (APNs) for both iOS/iPadOS/tvOS/watchOS and macOS devices
  - Baidu Cloud Push (Baidu): multi-platform
  - Firebase Cloud Messaging (FCM): Android, iOS, web, and desktop
  - Microsoft Push Notification Service for Windows Phone (MPNS)
  - Windows Push Notification Services (WNS): Universal Windows Platform applications
  - When to use SNS vs. SQS?

  While SNS and SQS are related, they serve different purposes.
- SQS is a queuing system. The tasks it excels at are maintaining a queue of messages and ensuring that each message is successfully pulled from the queue exactly once. As a result, SQS requires polling for items—"pulling" data, where it's the responsibility of the client to request the data it is ready to process.
- On the other end, SNS is a publish/subscribe system. The task for which it was designed is to send all the messages it receives to each of its many subscribers. SNS follows the "push" principle, where it's system's responsibility to make sure all subscribers are notified of the new

message. No polling on the part of subscribers is required.

- In some cases you might want to use only one of these services, sometimes using SQS together with SNS can be a good option, as in the one-to-many use case.

Primary Reference:

- Niveus – Google Cloud Pub/Sub-Benefits,Integration,Use Cases &More [16] https://niveussolutions.com/google-cloud-pub-sub-use-cases-pricing-benefits/

Summarization

- With its reliable and scalable messaging capabilities, Pub/Sub is well-suited for IoT (Internet of Things) communication and device management. IoT devices can publish messages to Pub/Sub, and subscribers can consume those messages to trigger actions or perform real-time analytics. Pub/Sub's ability to handle high message throughput and its support for both ordered and unordered message delivery are valuable in IoT scenarios where device-to-cloud and cloud-to-device communication is critical.
- Google Cloud Pub/Sub incorporates various mechanisms to ensure reliable and scalable messaging. One such mechanism is message acknowledgment, where subscribers confirm the receipt of messages. This enables Pub/Sub to track message delivery and handle retries if necessary, ensuring reliable message transmission.
- Additionally, Pub/Sub offers a dead letter policy, allowing undeliverable messages to be redirected to a separate dead-letter topic. This feature facilitates further analysis and troubleshooting of problematic messages. Moreover, Pub/Sub supports both ordered and unordered message delivery, granting developers the flexibility to control the sequencing of messages when required.
- Another key feature is Pub/Sub's horizontal scalability, which automatically scales to accommodate high message volumes. This scalability empowers applications to handle increased workloads without experiencing performance degradation. By leveraging these mechanisms, developers can construct robust and scalable applications that effectively process and deliver messages in real-time

Initial Architecture

User Management and Authentication Module

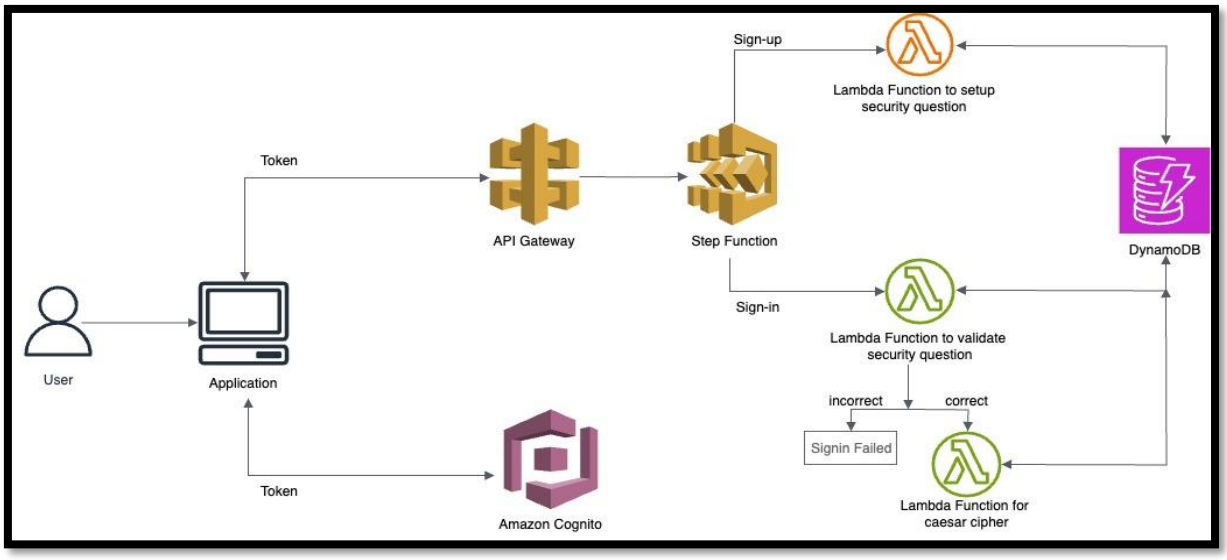

Figure 4: User Management and Authentication Module Architecture

The registered user or guest wanting to register to the app would first connect to Amazon Cognito. Amazon Cognito would perform the first step of the sign-up/sign-in process with username and password and on successful authentication, it would return a token. This token would be passed as an authorization header on any further calls made to lambda. We have used step functions to streamline the signup and sign-in steps to reach the appropriate lambdas for security questions and Caesar cipher. The lambdas would connect to DynamoDB to set and retrieve data.

Virtual Assistant Module

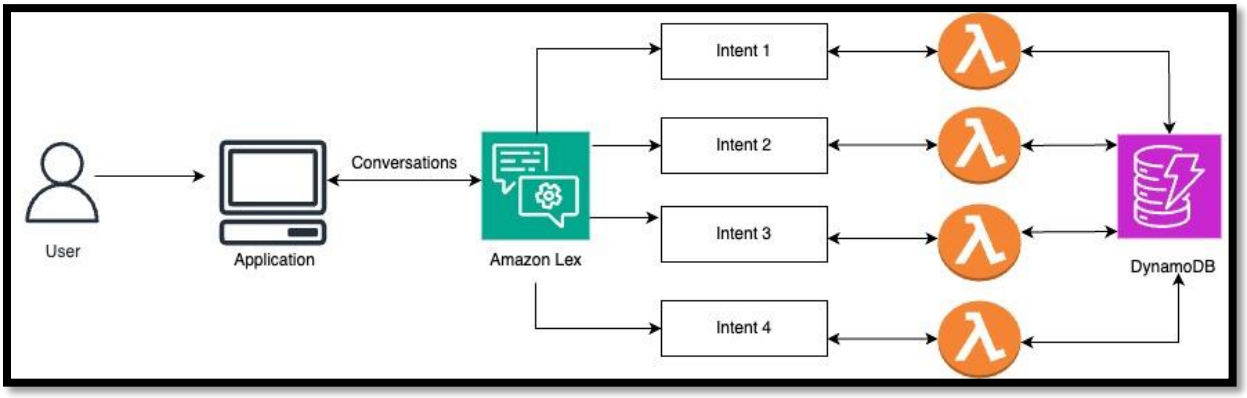

Figure 5: Virtual Assistant Module Architecture

We would set up various intents on Amazon Lex such as for reservation details, queries, etc. in order to give our chatbot the context of some common questions and how to answer them. We would create our own UI for the chatbot, and the application would interact with Amazon Lex. For some questions such as reservation details, Lex would need to fetch data from DynamoDB. To achieve this, Lex can connect to Lambda first, which can fetch data from DynamoDB, process it and return the necessary output to Lex. Lex can then return this response to the user as a response to their question.

Message Passing Module

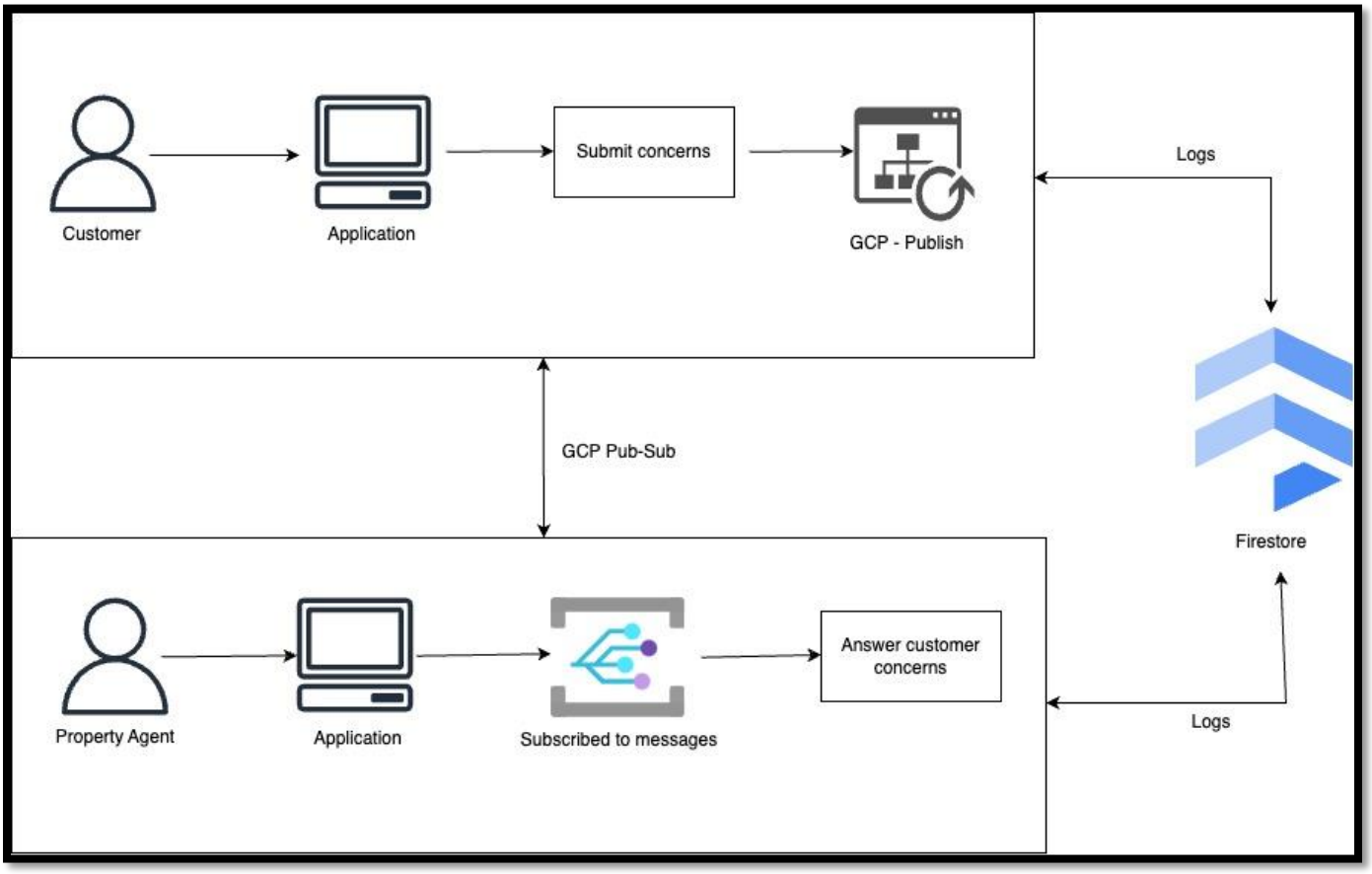

Figure 6: Message Passing Module Architecture

The registered customer would publish their concerns on the portal based on the booking code which would reach the property agent that has subscribed to it. The property agent would then answer the concern/query which would reach the customer. The communication/message passing is carried out through GCP Pub-Sub. The logs of these conversations are maintained on Firestore.

Notifications module:

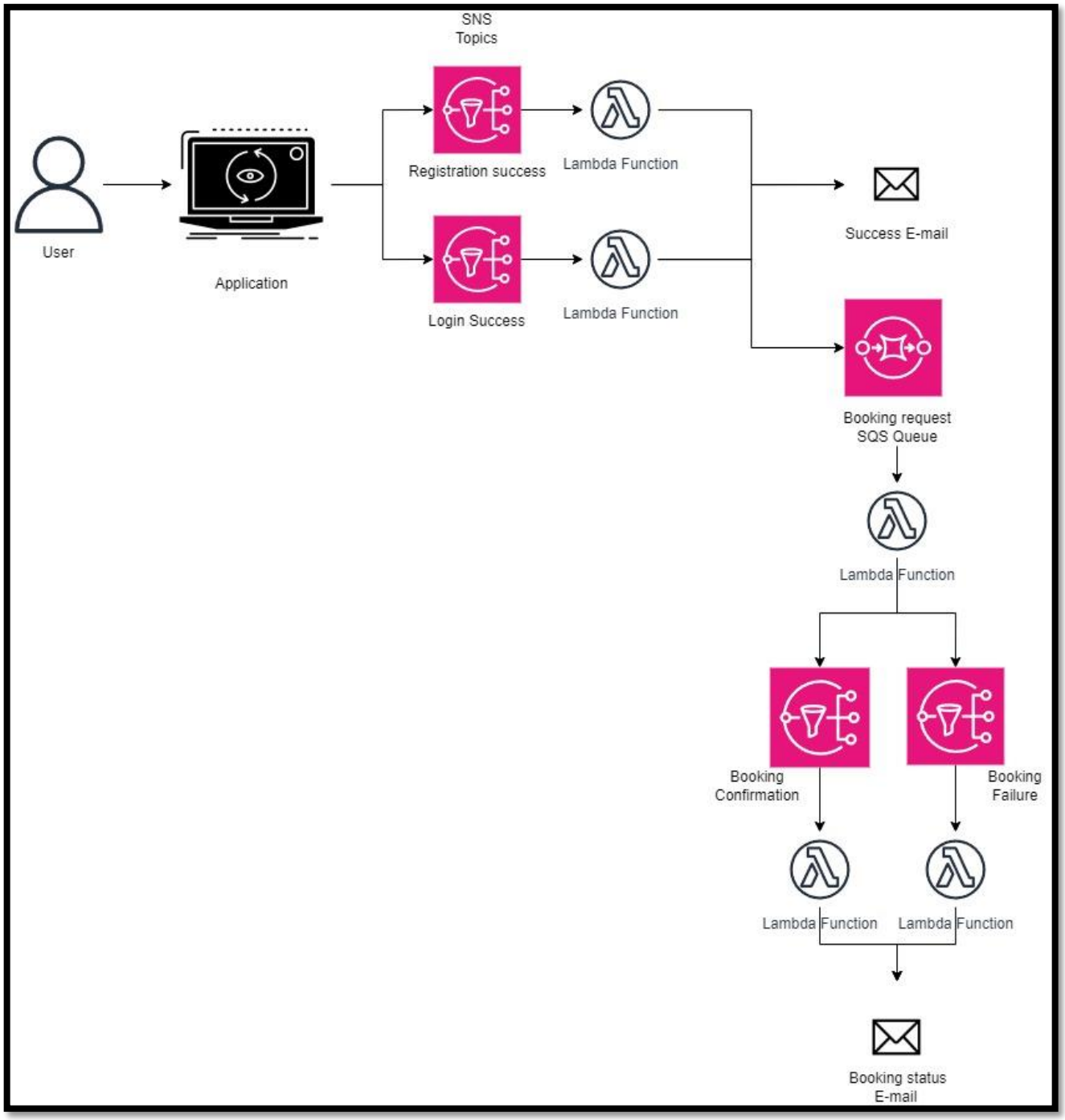

Figure 7: Notification Module Architecture

When users register or log in, the app would send notifications via SNS, which would trigger Lambda functions to send success emails. For bookings, requests would go to an SQS queue, triggering a Lambda function to process them. Depending on the outcome, SNS would send either a confirmation or failure notification via Lambda, and users would get an email about their booking status. This system

would ensure users are promptly notified about their registration, login, and booking status.

Data Analysis & Visualization:

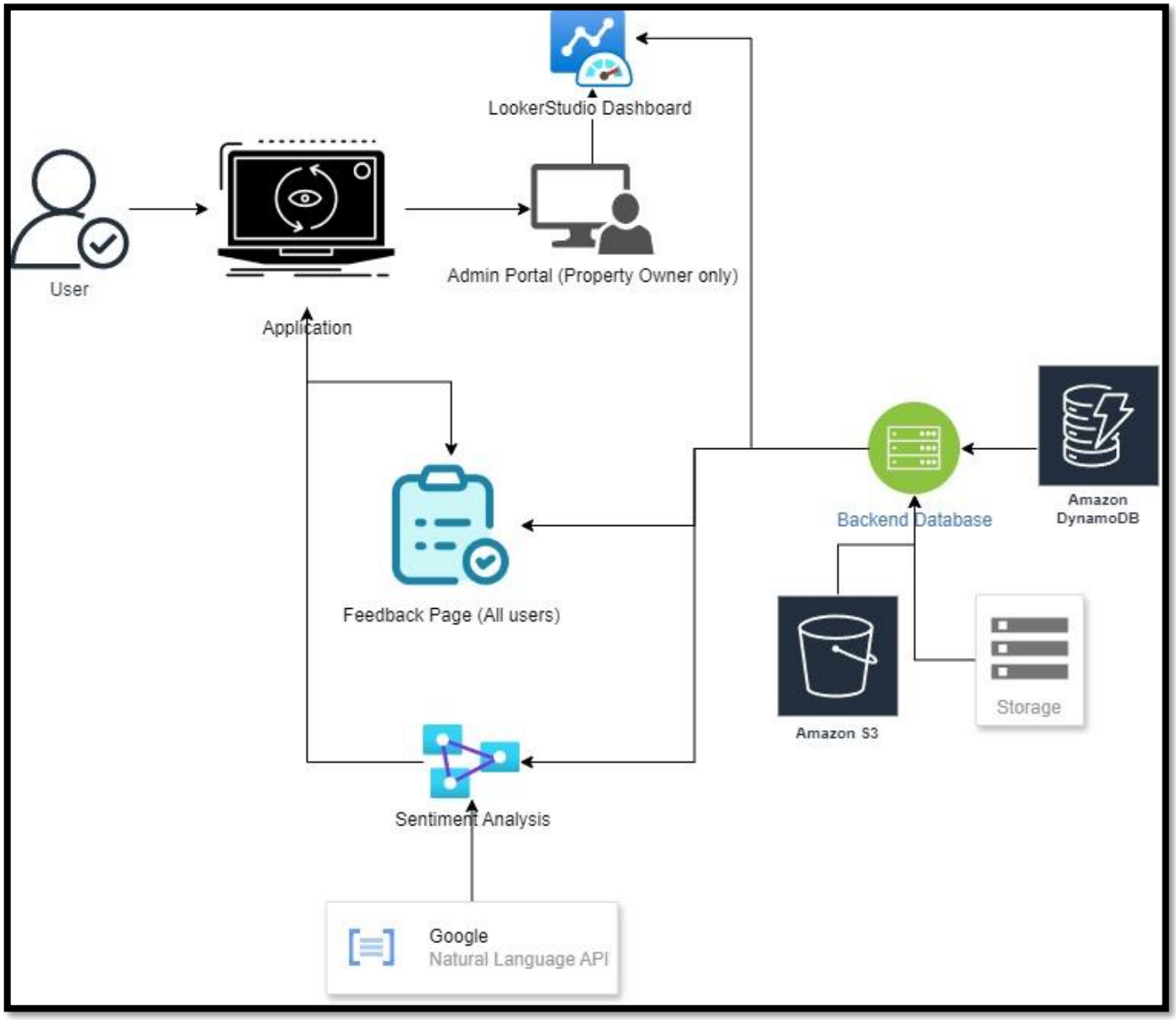

Figure 8: Data Analysis and Visualization Module Architecture

Property agents would be able to check user and login stats on an admin page with a LookerStudio dashboard. All users, including guests and customers, would be able to view customer feedback in a table format on the frontend, with data retrieved from DynamoDB, S3, or other storage services. Feedback would be analyzed using Google's Natural Language API to determine sentiment, and this analysis would be displayed for all users on the frontend.

Web Application Building and Deployment:

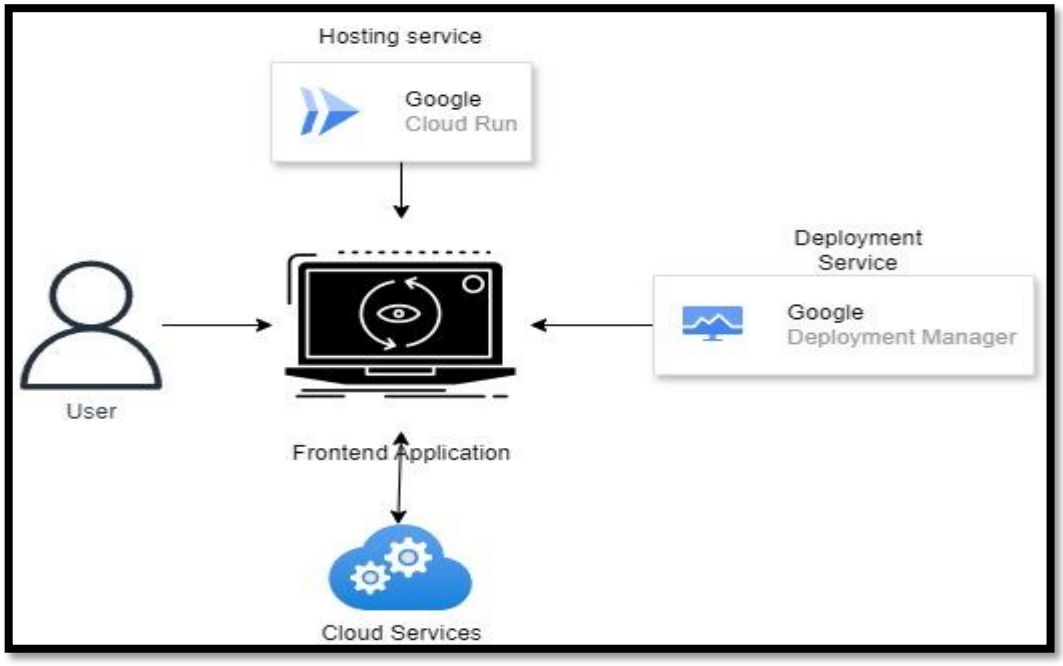

Figure 9: Frontend deployment using CloudRun

The entire application and user/client-facing interface would be hosted using GCP Cloud Run. Automation of service deployment would be handled using CloudFormation or GCP Cloud Deployment Manager.

Project Planning

We have decided to use Gitlab as our project management tool.

Link to our issue board: https://git.cs.dal.ca/ndavies/csci5410-s24-sdp-6/-/boards/1927

Gannt chart:

Task	Start Date	End Date	Duration
Sprint 1: Background research	5/15/2024	5/21/2024	6
Sprint 1: Basic architecture and projet planning	5/22/2024	5/29/2024	7
Sprint 2: User Management & Authentication Development	5/30/2024	6/5/2024	6
Sprint 2: Virtual Assistant Development	6/6/2024	6/12/2024	6
Sprint 2: Message Passing Development	6/13/2024	6/19/2024	6
Sprint 2: Notifications Development	6/20/2024	6/26/2024	6
Sprint 2: Data Analysis and Visualization Development	6/27/2024	7/5/2024	8
Sprint 3: Application Testing and Bug fixing	7/6/2024	7/21/2024	15
Sprint 3: Application Deployment and Hosting	7/22/2024	7/31/2024	9

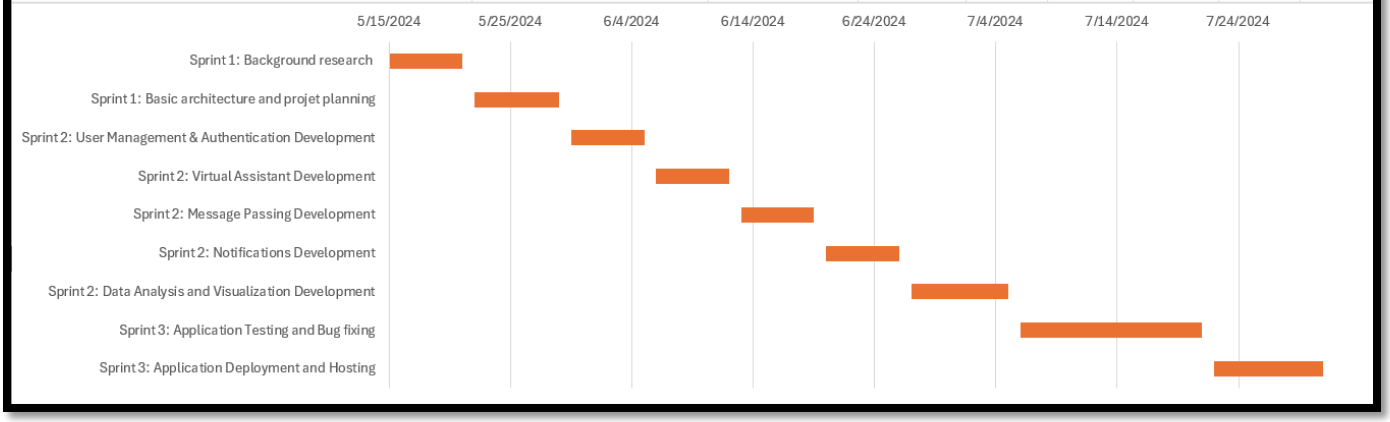

Figure 10: Gantt chart illustrating modules in each sprint

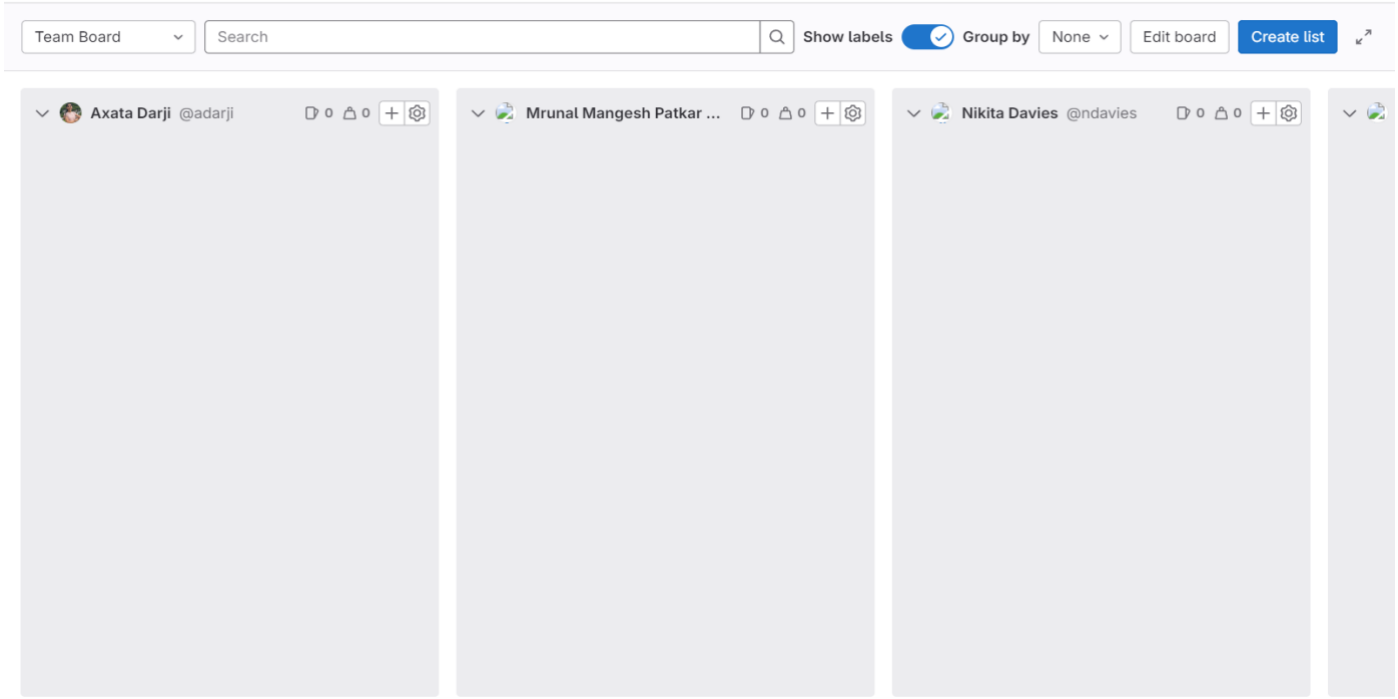

Figure 11: Team Board created on Gitlab for project management

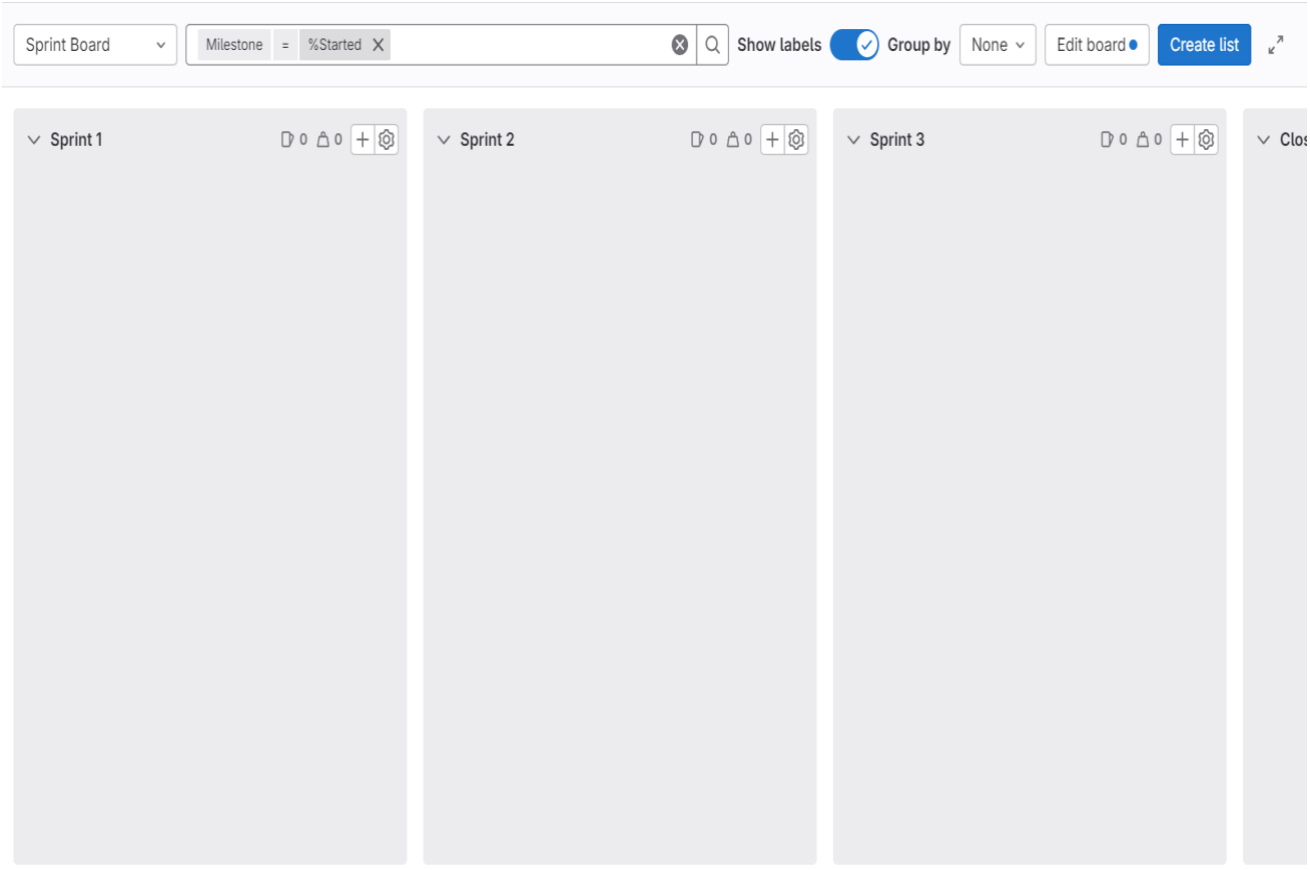

Figure 12: Sprint Board created on Gitlab for project management

Individual Contribution

Table 1: Individual Contribution

Team Member	Contribution

Nikita Davies	• Explored on the following GCP service implementation
	  • GCP Pub/Sub
	  • LookerStudio
	  • Google Natural Language API
	  • Google CloudRun
	• Explored more on serverless cloud architecture.

Mrunal Mangesh Patkar	• Explored the options for Gantt chart creation and project management tools. • Created Gannt chart. • Created the initial project management board. • Explored cloud services integration process. • Contributed towards tentative architecture on the project.

Rameez Parkar	• Explored on the following AWS service implementation.
	  • AWS Cognito
	  • AWS Lambda
	  • AWS DynamoDB
	  • AWS Lex
	  • AWS Simple Notification Service
	  • AWS Simple Queue Service
	• Contributed towards tentative architecture on the project.

Axata Darji	• Identified application feature and flow. Explored on the application perspective research. • Identified few online resources which has already implemented some of the matching functionalities. • Scheduling and maintaining meeting logs and recordings

Meeting Logs - Sprint 1

Table 2: Meeting Logs for Sprint 1

Meeting Date	Meeting Time (Start)	Meeting Time (End)	Meeting Place	Outcomes	Attendees	Recording link
17/05/2024	6:00PM	6:30 PM	online	Introduction and plan for sprint1	All present	https://dalu-my.sharepoint.com/personal/ax583820_dal_ca/_layouts/15/stream.aspx?id=%2Fpersonal%2Fax583820%5Fdal%5Fca%2FDocuments%2FRecordings%2FCSCI%205410%2D%20Team%206%20Introduction%20call%2D20240517%5F180313%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ed151eddf%2D3ac8%2D431c%2Db50a%2D4acdfd69b525&ga=1
22/05/2024	1:30 PM	2:30 PM	In-person	Plan for sprint 1 report and update on the individual task	All present	Meeting in _General_-20240522_140828-Meeting Recording.mp4 (sharepoint.com)
26/05/2024	7:00 PM	7:30 PM	online	Discussed on individual task update and plan for final project report for sprint 1	All present	CSCI 5410- Team 6 Sprint 1 report meeting-20240526_190119-Meeting Recording.mp4 (sharepoint.com)
28/05/2024	2:30 PM	3:00 PM	Online	Finalized the architecture and the project planning	All present	https://dalu.sharepoint.com/teams/CSCI5410_Serverless_Team6/_layouts/15/stream.aspx?id=%2Fteams%2FCSCI5410%5FServerless%5FTeam6%2FShared%20Documents%2FGeneral%2FMeeting%20Recordings%2FCSCI%205410%2D%20Group%206%20sprint1%20report%20final%20call%2D20240528%5F143126%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2E46f1b306%2Df018%2D4077%2Da481%2D9744141b3111

Screenshot of team chats

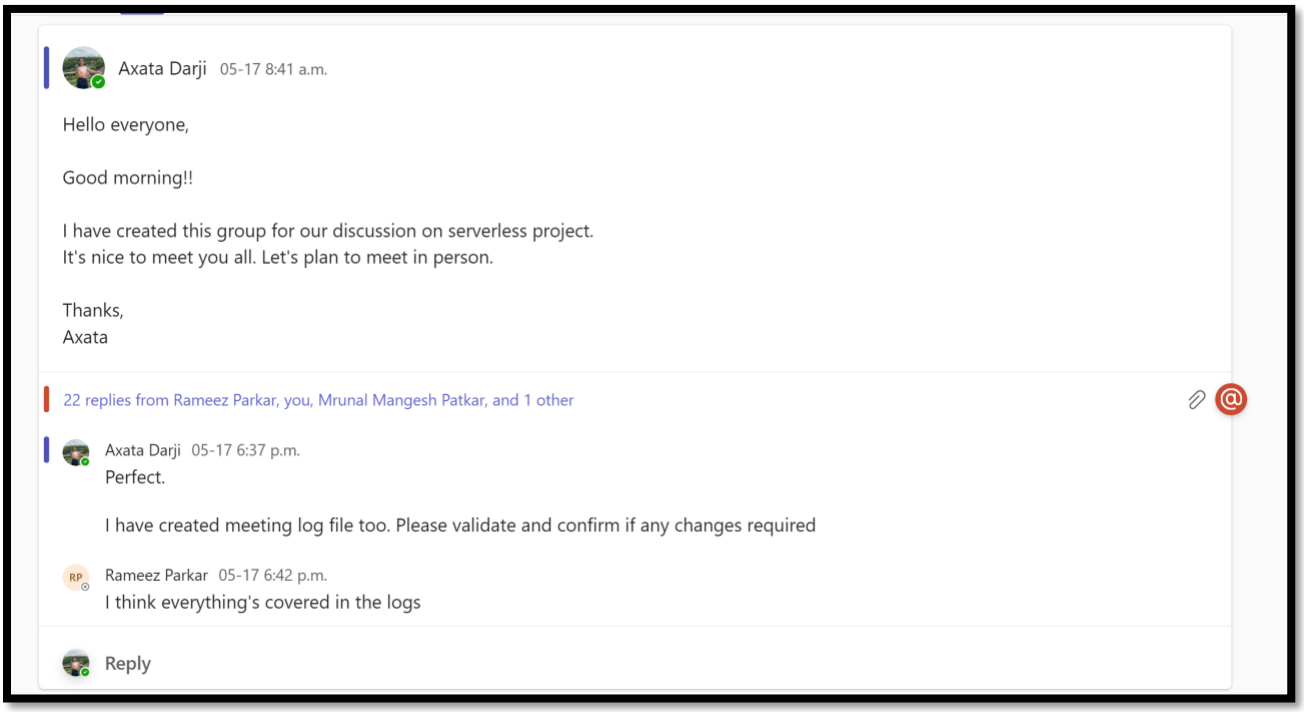

Figure 13: Screenshot of group creation for discussing project

Figure 14: Screenshot showing discussion regarding research

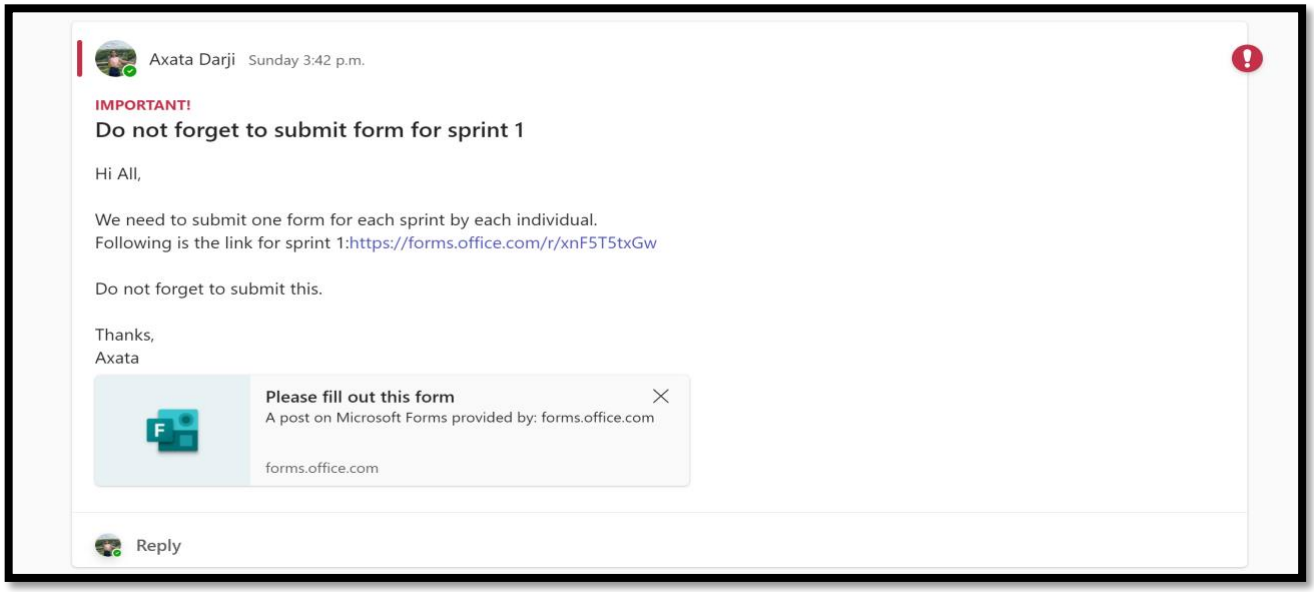

Figure 15: Screenshot of discussion regarding peer feedback form

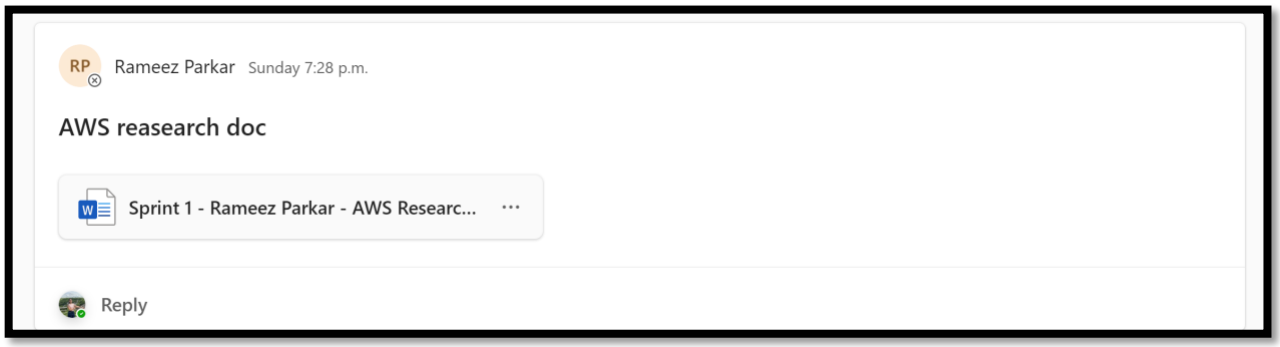

Figure 16: Screenshot showing research conducted on AWS

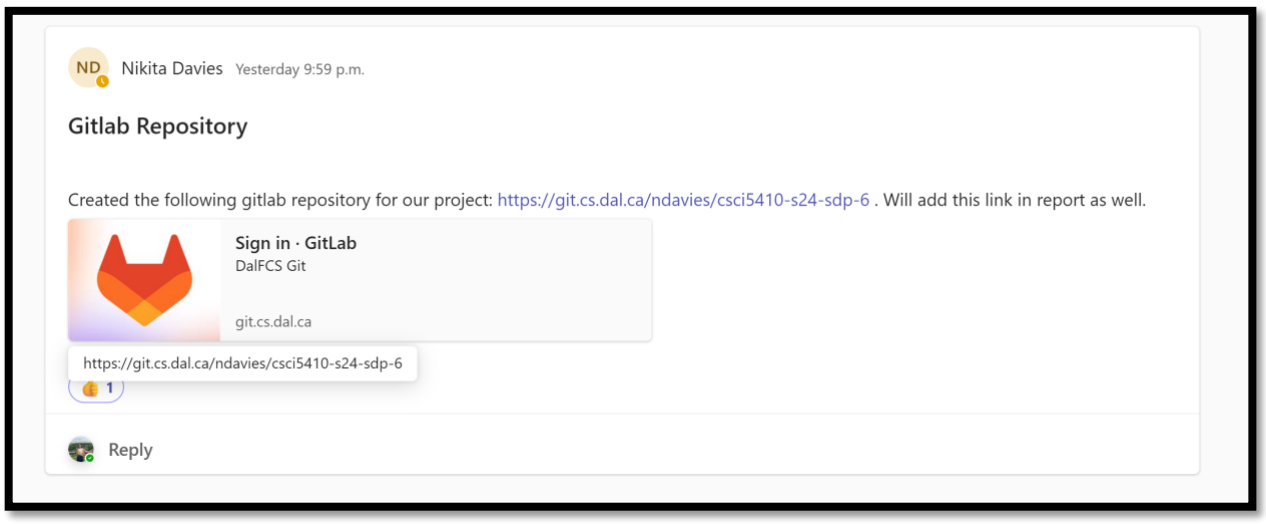

Figure 17: Screenshot of created gitlab repository

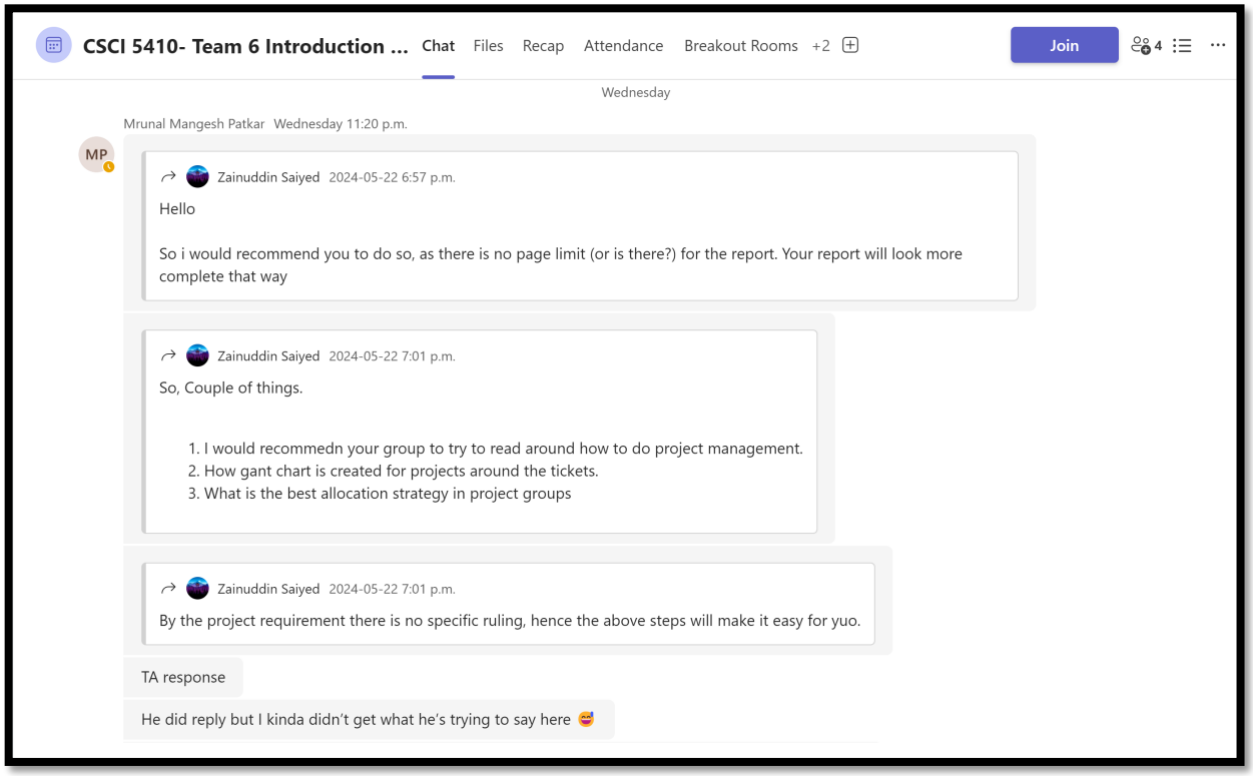

Figure 18: Screenshot regarding doubt clarification with TA

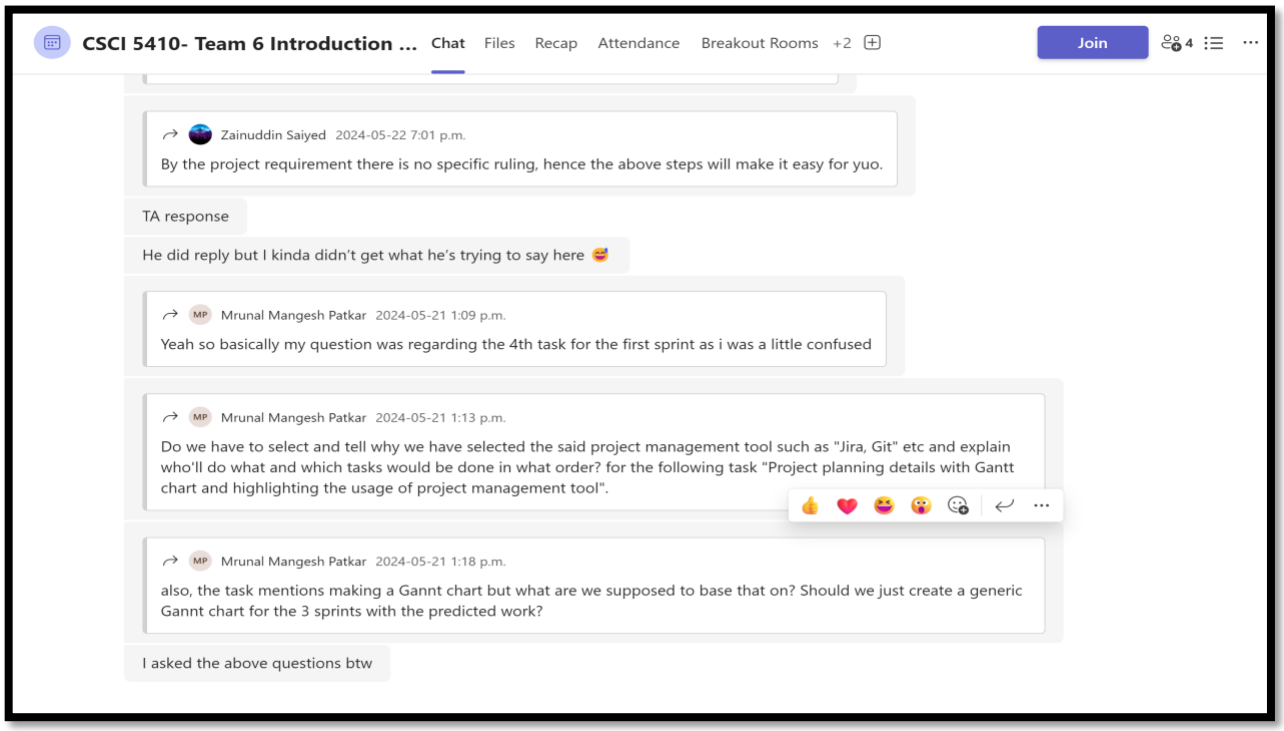

Figure 19: Screenshot regarding doubt clarification with TA regarding Gantt chart

References

[1] "Amazon Cognito user pools", AWS. Accessed: May 25, 2024. [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html.

[2] "What is Amazon DynamoDB?", AWS. Accessed: May 25, 2024. [Online]. Available: https://aws.amazon.com/pm/dynamodb/

[3] "AWS Lambda", AWS. Accessed: May 25, 2024. [Online]. Available: https://aws.amazon.com/lambda/

[4] "What is Amazon Lex?", AWS. Accessed: May 25, 2024. [Online]. Available: https://docs.aws.amazon.com/lex/latest/dg/what-is.htmlCPMA

[5] "What is Amazon SNS?", AWS. Accessed: May 25, 2024. [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html

[6] Harsh Ananad,Satyam Biradar,Naveen Prajapat,Prof. Shripad G Desai, "Deployment of a Serverless Web Application using AWS services," American Journal of Science and Engineering. Accessed: May 25, 2024. [Online]. Available: https://ajse.us/wp-content/uploads/2023/03/AJSE-V3-I4_paper1.pdf

[7] "Pub/Sub", Google Cloud. Accessed: May 26, 2024. [Online]. Available: https://cloud.google.com/pubsub?hl=en

[8] "Pub.Sub in GCP", Medium. Accessed: May 26, 2024. [Online]. Available: https://medium.com/@teja.ravi474/pub-sub-in-gcp-27a21554d158

[9] "Create Reports from Amazon DynamoDB Data in Looker Studio", Cdata. Accessed: May 26, 2024. [Online]. Available: https://www.cdata.com/kb/tech/dynamodb-cloud-google-data-studio.rst

[10] "Natural Language AI", Google Cloud. Accessed: May 26, 2024. [Online]. Available: https://cloud.google.com/natural-language?hl=en

[11] "Build applications or websites quickly on a fully managed platform", Google Cloud. Accessed: May 26, 2024. [Online]. Available:https://cloud.google.com/natural-language?hl=en

[12] Garrett McGrath,Paul R.Brenner,"Serverless Computing:Design,Implementation, and Performance", IEEE. Accessed: May 26, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/7979855

[13] "Serverless architecture: Introduction to common serverless challenges", Prisma. Accessed: May 26, 2024. [Online]. Available: https://www.prisma.io/dataguide/serverless/serverless-challenges

[14] Greivin Lopez, "Winston:A Virtual HR Assistat Created with Amazon Lex and Slack", Gorilla Logic. Accessed: May 24, 2024. [Online]. Available: https://gorillalogic.com/blog/winston-a-virtual-hr-assistant-created-with-amazon-lex-and-slack

[15] "Amazon SNS(from AWS)", serverless. Accessed: May 26, 2024. [Online]. Available: https://www.serverless.com/guides/amazon-sns

[16] "Google Cloud Pub/Sub – Benefits,Integration, Use cases & More", niveus. Accessed: May 26, 2024. [Online]. Available: https://niveussolutions.com/google-cloud-pub-sub-use-cases-pricing-benefits/