**Technical Specification Document( TSD)**

|  |  |
| --- | --- |
| **Title** |  |
| ***Document Name*** | Private Chat application |
| ***Version No.*** | 1.2 |
| ***Date*** | 19/04/2024 |
| ***Document Author(s)*** | Jyotikrishna Behera, Divya Dash, Jyotishree Pradhan, Aripra Singh |

**Version History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Change** | **Sections Affected** |
| ***1.0*** | 17/04/2024 |  | Add details |  |
| ***1.1*** | 19/04/2024 |  | Made minor changes |  |

**Reviewer(s)**

|  |  |
| --- | --- |
| **Name** | **Title** |
|  |  |
|  |  |
|  |  |
|  |  |

**Approver(s)**

|  |  |
| --- | --- |
| **Name** | **Title** |
|  |  |
|  |  |
|  |  |
|  |  |

**Document References**

|  |
| --- |
| **Related Documents:** |
| [Using Lambda with Amazon SQS - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html)  [Amazon Simple Queue Service Documentation](https://docs.aws.amazon.com/sqs/) |
| [Using AWS Lambda with Amazon DynamoDB - AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html)  [Amazon Simple Notification Service Documentation](https://docs.aws.amazon.com/sns/) |
| [Working with WebSocket APIs - Amazon API Gateway](https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html)  [AWS Lambda Documentation (amazon.com)](https://docs.aws.amazon.com/lambda/)  [How to build real-time applications using WebSockets with AWS API Gateway and Lambda | by Janitha Tennakoon | We’ve moved to freeCodeCamp.org/news | Medium](https://medium.com/free-code-camp/real-time-applications-using-websockets-with-aws-api-gateway-and-lambda-a5bb493e9452) |

**Table of Contents**

[1. Introduction 3](#_Toc1232272363)

[2. Business Requirement 4](#_Toc1498684019)

[2.1.1. Functional 4](#_Toc1560658684)

[2.1.2. Non-Functional 4](#_Toc815383799)

[2.1.2.1. Timings & Frequency 4](#_Toc596333736)

[2.1.2.2. Security 4](#_Toc1709418893)

[2.1.3. 1.1.2.3 Training 4](#_Toc1597202759)

[Technical Approach 5](#_Toc1637057951)

[Scope 5](#_Toc2059887552)

[3.1.1. In Scope 5](#_Toc1148970969)

[3.1.2. Out of Scope 5](#_Toc1435241824)

[Other Affected Applications 6](#_Toc99605826)

[Assumptions 6](#_Toc1723271899)

[Issues 6](#_Toc1436295078)

[Risks 6](#_Toc1722181036)

[4. Design Elements 6](#_Toc1749605519)

[Online Objects 7](#_Toc69952237)

[4.1.1. Project Details 7](#_Toc1886289451)

[4.1.3. Menus 7](#_Toc511571427)

[4.1.4. Components 7](#_Toc162806500)

[4.1.5. Pages 8](#_Toc333495287)

[4.1.6. Records 8](#_Toc34507581)

[4.2.1. Code 8](#_Toc1383257963)

[4.2.2. Field 9](#_Toc1485187243)

[4.2.3. Message Catalog 9](#_Toc822280544)

[4.2.4. Process Definition 9](#_Toc323922770)

[4.2.6. Security Requirements 9](#_Toc1746125581)

[Batch Programs 10](#_Toc698869643)

[4.2.7.1. Batch Program: <Program Name> 11](#_Toc1708447526)

[5. Testing 12](#_Toc648498115)

[Test Plan 12](#_Toc1972795649)

[Test Data 12](#_Toc2035393567)

[Third Party Requirements 12](#_Toc489400423)

[6. Issues 13](#_Toc362560107)

[7. Appendix 13](#_Toc51064707)

# Introduction

The organization requires a secure, user-friendly, and efficient internal chat platform that enables real-time messaging and collaboration among employees. The platform should prioritize data privacy and security, ensuring that internal information remains confidential and protected. The current communication system lacks the necessary security measures, leading to potential data breaches and unauthorized access to sensitive information. To address these challenges and enhance data security, the organization aims to develop a comprehensive chat application that leverages secure AWS services.

We employ AWS WebSocket API Gateway to manage messaging, both for public and private conversations. Chat messages are securely stored in a DynamoDB table after processing in an SQS queue, enabling decoupling of services and ensuring data integrity. AWS SNS is utilized to send automated email notifications when new users join the platform, enhancing user engagement and awareness. Additionally, AWS Cognito is employed for user authentication, ensuring secure access to the chat platform and maintaining control over user identities.

By leveraging these secure AWS services, our chat application enables the organization to maintain the privacy and confidentiality of internal information. Employees can securely communicate and collaborate in real-time, ensuring that sensitive data remains within the organization's control. This enhances data security, minimizes the risk of unauthorized access, and fosters a more efficient and transparent communication environment within the organization.

# Business Requirement

### Functional

1. User Registration: Users should be able to create an account by providing their name, email address, and password. This information will be used for authentication and identification purposes.
2. User Authentication: The application should provide a secure authentication mechanism to verify the identity of users during login. This can include email verification, password hashing, and session management.
3. Sending Messages privately: Users should be able to send messages in real-time to a specific user mentioning them privately.
4. Sending Messages publicly: Users should be able to send messages in real-time within chat rooms.
5. Message History: The message history within chat rooms should be maintained, allowing them to catch up on previous conversations.
6. New user notification: all users should be notified whenever a new user logs-in

### Non-Functional

### Timings & Frequency

The chat application should provide instant message delivery and support high message throughput to handle many concurrent users.

### Security

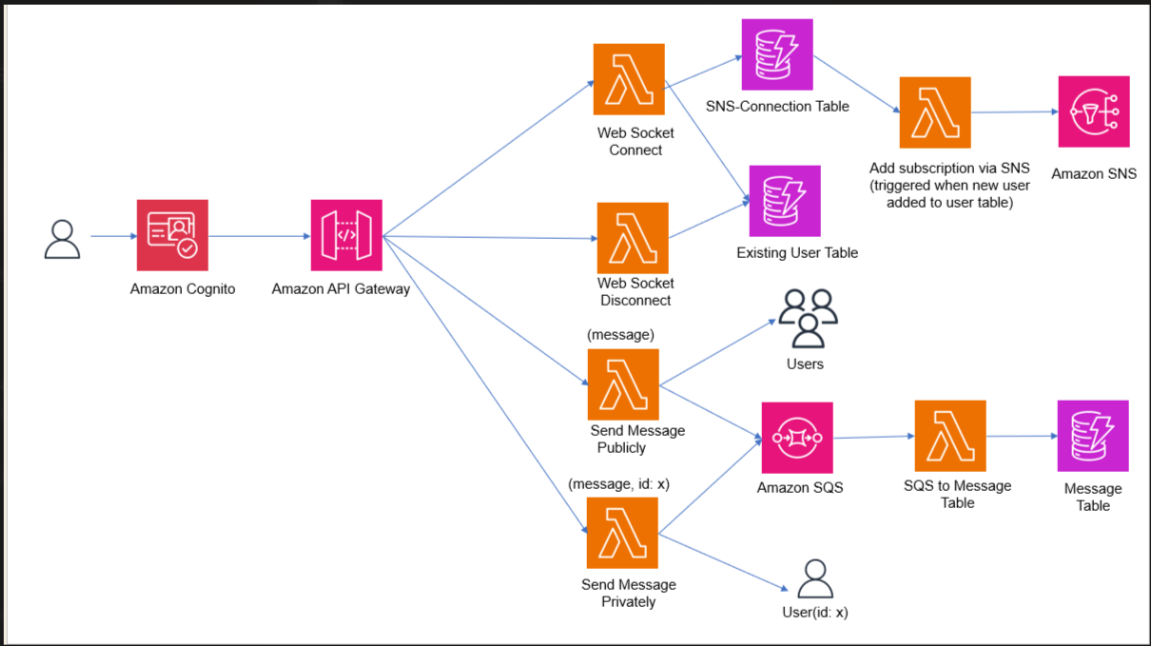
The application should ensure secure communication by encrypting messages and implementing user authentication and authorization mechanisms

1. **Technical Design**

The technical design of this application leverages AWS services to build a scalable and secure web-based chat application. It utilizes WebSocket API Gateway for real-time communication, DynamoDB for chat message storage, SQS for decoupling services, SNS for email notifications, and Cognito for user authentication. The frontend is built using Node.js and TypeScript, providing a robust and efficient client-side experience.

## Technical Approach

**<Architecture Diagram>**



**GitHub Repository Link:**

<https://github.com/jkbpwc/AWSChatApp>

## Scope

### In Scope

* User registration and authentication
* Real-time messaging functionality
* Chat room creation and management
* Contact management

### Out of Scope

* File sharing functionality
* Voice or video calling features

## Assumptions

* Users will have access to modern web browsers that support WebSocket technology.
* The application will be hosted on a scalable cloud infrastructure.

## Issues

* Integration of WebSocket API with AWS services like Cognito and sending media files through it is a complex task.
* Implementation of AWS services keeping in mind the cost constraint of accounts.

## Risks

* Potential scalability challenges due to many concurrent users.

# Design Elements

## Online Objects

### Components

***Component Properties***

|  |  |  |
| --- | --- | --- |
| **Component** | **Search Record** | **Action** |
| Connect Button | Connected to user Database | Calls connect Lambda |
| Send Private Button | Connected to Chats Database | Calls sendPrivate |
| Send Public Button | Connected to Chats Database | Calls sendMessage |
| Disconnect Button | Connected to user table | Calls the disconnect lambda |

### Pages

* + - 1. Page:

***Page Operations / Description***

1. ***Hosted UI Cognito: It*** *is the page where the user signs up using email and then gets a verification code.*
2. *Connection Page: The user enters the email id and connects and then receives a SNS notifications for subscription and if already subscribed then gets a email whenever a new user is added.*
3. *Send Message Page: The user can either choose to send message publicly or privately.*

*3.1) Send Public: The user sends a public message, and everyone connected gets the*  *message.*

*3.2) Send Private: The user is first is asked to enter the user id of the person whom they*  *wants to send a message.*

*3.3) Disconnect: The user gets deleted from the user table ones they click the disconnect*  *button*

## Batch Programs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl.** | **Program Name** | **Program Type** | **Interface Type** | **Frequency** | **Description** |
| 1  2  3  4  5  6 | sendPublic  SendPrivate  Connect  Disconnect  SNS notification  SQS | Lambda function  Lambda funtion  Lambda function  Lambda function  Lambda function  Lambda function | Websocket api  Websocket api  Websocket api  Websocket api  Email service  Queue service | On demand  On demand  On login  On logout  On login  Real time | This function is triggered whenever the user sends a public message, it gets all the user ids from the user pool and sends the message to each.  This function is triggered whenever the user sends a private message to a specific user.  Whenever a new user login their email is stored in user table and all existing users get a mail about the new user with his private connectionId.  When the user wants to leave the room, he disconnects from the websocket connection and his connectionId also gets removed.  Whenever a new email is stored in user table and all existing users get a mail about the new user with his private connection id.  When a new message is generated, it gets into the SQS queue, and then it gets stored in chats table. Enabling loose coupling architecture. |

* + - 1. Programming Logic

The programming logic of the chat application involves several key components and functionalities. Here's an overview of the programming logic:

1. User Registration and Authentication:

* Implement a user registration process where users can create an account by providing their name, email, and password.
* Utilize AWS Cognito for user authentication, including sign-up, sign-in, and token-based authentication.

1. WebSocket Connection:

* Establish a WebSocket connection between the client and server using the AWS WebSocket API Gateway.
* Handle WebSocket events such as connection establishment, disconnection, and error handling.
* Authenticate WebSocket connections using AWS Cognito tokens to ensure secure communication.

1. Real-Time Messaging:

* Enable users to send and receive messages in real-time within chat rooms.
* Implement message broadcasting to all connected clients within a chat room.
* Handle message routing and delivery to the appropriate recipients.

1. Private Messaging:

* Implement private messaging functionality, allowing users to send direct messages to specific individuals.
* Manage private message threads and ensure message privacy and security.

1. Message Storage and Retrieval:

* Store chat messages securely in a DynamoDB table after processing in an SQS queue.
* Implement logic to retrieve and display chat message history within chat rooms.
* Enable search functionality to find specific messages based on keywords or criteria.

1. Error Handling and Logging:

* Implement error handling mechanisms to gracefully handle exceptions and provide meaningful error messages to users.
* Utilize logging frameworks or services to capture and log application events and errors for troubleshooting and analysis.

4.1.1.1. Error Handling

1. Provide Meaningful Error Messages: When an error occurs, provide clear and informative error messages to users. This helps them understand what went wrong and how to resolve the issue. Avoid exposing sensitive information in error messages.
2. Use Try-Catch Blocks: Wrap critical sections of code with try-catch blocks to catch and handle exceptions. This helps prevent unhandled exceptions from crashing the application and provides an opportunity to handle errors gracefully.
3. Centralized Error Logging: Implement a centralized error logging mechanism to capture and log errors that occur during runtime. This can be achieved using logging frameworks or services like AWS CloudWatch Logs. Logging errors allows for easier troubleshooting and analysis of issues.
4. Handle Expected Errors: Identify and handle expected errors specific to the chat application. For example, if a user tries to send a message without being authenticated, return an appropriate error response or redirect them to the login page.

# Testing

## Test Plan

* Test user registration and authentication
* Test real-time message sending and receiving
* Test chat room creation and management
* Test contact management functionality

## Test Data

* Test user accounts with different roles and permissions.
* Test chat rooms and messages for various scenarios.

## Third Party Requirements

1. The User should be part of the Organization.
2. The user should have a valid mail id.

# Issues

|  |  |
| --- | --- |
| **Issue** | **Status** |
| Integrating AWS cognito with websocket api with office network | unsolved |
| Sending media files | unsolved |
|  |  |

# Appendix

1. Architecture Overview:

* The architecture of the Chat App follows a client-server model, with the client built using Node.js and TypeScript, and the server utilizing various AWS services.
* The high-level architecture diagram illustrates the components, including Cognito for user authentication, WebSocket API Gateway for real-time messaging, Lambda functions for message processing, SQS for message buffering, SNS for email notifications, and DynamoDB for message storage.

1. AWS Services Used:
   * Cognito: Used for user authentication and authorization, ensuring secure access to the chat platform.
   * WebSocket API Gateway: Facilitates real-time communication between users, enabling instant messaging capabilities.
   * Lambda functions: Handle the backend logic, processing and storing messages in the DynamoDB table.
   * SQS: Acts as a buffer for incoming messages, ensuring reliable message delivery and decoupling message processing.
   * SNS: Sends automated email notifications to existing users when new users are added to the platform.
   * DynamoDB: A cloud-based database used for secure and scalable storage of chat messages.
2. Implementation Details:
   * Step-by-step instructions for setting up and configuring each AWS service, including Cognito, WebSocket API Gateway, Lambda functions, SQS, SNS, and DynamoDB.
   * Code snippets and explanations for key implementation aspects, such as user authentication, WebSocket API Gateway configuration, Lambda function development for message processing, SQS integration for message buffering, and DynamoDB data storage.
3. Frontend Development:
   * Description of the front-end development approach using Node.js and TypeScript.