ZCONNECT

Email-Chat-Softphone

# 

# Table of Contents

1. Executive summary

1.1. Key features and highlights

1. Technology Stack

2.1. Programming languages

2.2. Databases

1. Configurations
2. Email

4.1. Email application overview

4.2. User interface

4.3.Incoming Mail Workflow

4.4. Email Management Features

4.5. Handling Attachments

4.6. Incoming and Outgoing Mails Management

4.7. Wrap-Up Process

4.8. Rejected or Unaccepted Mail Handling

4.9. Composing a New Mail

4.10. Handling New Mails in the SMTP Server When No Agents Are Available

1. Chat Module Overview

5.1. Chat component

1. Technology stack

6.1. Programming languages

6.2. Databases

7.User Registration for Chat

8.Handling User Messages in Specific Queues

9.Chat Handling Workflow for Agents:

9.1. Message Handling

9.2. Attachment Support

9.3. Agent Capacity

9.4. Chat Session Management and Interaction Count

9.5. Rejected or Unaccepted Chat Handling

# Executive Summary

# This document provides an overview of the functionality and integration of the **Chat, Email, and Softphone Application**, highlighting its ability to handle these communication channels simultaneously.

# The application is designed to streamline communication and enhance productivity by enabling users to manage chats, emails, and softphone interactions within a unified interface. The document details how each component works and how they seamlessly integrate to provide a cohesive user experience.

# **1.1. Key Features and Highlights**

# **Chat Management**:

# Real-time messaging with users.

# Ability to manage multiple chat conversations simultaneously.

# **Email Handling**:

# Send, receive, and reply to emails.

# Attachments supported in various formats (e.g., images, PDFs, text files, and spreadsheets).

# Integration with popular email services like Gmail and Outlook.

# **Softphone Functionality**:

# Make and receive voice calls directly through the application.

# **Simultaneous Communication**:

# Handle chats, emails, and calls concurrently.

# Notifications ensure no interaction is missed, even during multitasking.

# This document serves as a guide to understanding the core functionalities and operational flow of the application, emphasizing its efficiency and user-friendly design.

**2.Technology Stack**

The following technologies are used in the development of the application:

**2.1. Programming Languages:**

* **Python**:
  + Used for backend development, handling server-side logic, and managing database interactions.
* **React.js**:
  + Used for frontend development to create a dynamic and responsive user interface.

**2.2. Databases:**

* **PostgreSQL**:
  + A robust relational database system used for permanent storage of wrapped-up mails and other application data.
* **Redis**:
  + A high-performance in-memory data store used for temporary storage, such as cache memory for managing active and rejected mails.

This technology stack ensures the application is scalable, efficient, and user-friendly, with a strong emphasis on performance and reliability.

**3.Configurations  
Versions Used**

* **Python Version**: 3.11.0
* **Node.js Version**: v16.14.0
* **NPM Version**: 8.3.1

**Configuring SMTP Server**

To configure your SMTP server for sending emails, follow these steps:

1. Generate an SMTP Password:
   * Log in to your Gmail account.
   * Go to Manage Your Google Account > Security.
   * Under "Signing in to Google", look for App Passwords.
   * Generate a new app password by selecting the app (e.g., Mail) and the device (e.g., Custom).
   * Copy the app password.

# Backend Commands to start the application

python emailpolling.py

python Chatserverapi.py

python ChatServer.py

# Frontend Command to start the application

npm start

# **4.Email**

**4.1. Email Application Overview**

Our email application is designed to function similarly to popular mail platforms like Gmail or Outlook. It includes key features that facilitate seamless communication between agents and users. Below is an outline of its features and functionality:

**Key Features**

* **Mail Management**:
  + Receive emails from users.
  + Reply to emails received by agents.
  + Send new emails to users.
* **Compatibility**:
  + Supports integration with Gmail and Outlook for sending and receiving emails.
* **Attachments**:
  + Allows sending and receiving attachments in various formats, including:
    - Images: JPEG, JPG
    - Documents: PDF, TXT, Excel sheets (e.g., .xls, .xlsx).

**4.2. User Interface**

Once both the backend and frontend applications are started successfully, the UI will be accessible, enabling users to perform tasks such as viewing emails, replying, composing new emails, and managing attachments.

A screenshot of a computer

Description automatically generated

**4.3. Incoming Mail Workflow**

The incoming mail workflow ensures efficient handling of emails and prioritizes seamless user-agent communication. Below is the detailed process:

* **Notification of New Mail**:
  + When a new email arrives from a user to an agent, a **pop-up notification** is displayed.
  + The notification includes essential details such as:
    - **Username**: Identifies the sender.
    - **Subject**: Provides the subject of the email.
    - **Action Buttons**: Options to **Accept** or **Reject** the email.
* **Agent State Management**:
  + Upon receiving the incoming email notification, the agent's state changes from **Available** to **Available on Demand**.
  + In this state, the agent is prevented from receiving other chats, calls, or emails, ensuring undivided attention to the current interaction.
  + This restriction guarantees that the agent handles only **one incoming item** at a time.

**A screenshot of a computer

Description automatically generated**

* **Email Handling**:
  + Once the agent **accepts** the email:
    - The email is listed in the **left panel** of the interface.
    - Clicking on the email in the left panel opens its details in the **right panel**, allowing the agent to view and respond.

A screenshot of a chat

Description automatically generated

* **Returning to Availability**:
  + After addressing the interaction, the agent's availability is reassessed.
  + The agent becomes **Available** again if their total interactions (including chats, calls, and emails) are fewer than **3**.
* This structured approach ensures that agents manage their workload effectively, providing focused attention to each user while maintaining optimal productivity and service quality.

A screenshot of a chat

Description automatically generated

**4.4. Email Management Features**

The email application offers a comprehensive suite of features to efficiently manage incoming and outgoing emails. Below is an overview of the key functionalities:

**Email Reply and Forwarding**

* **Reply and Reply All**:
  + Respond to the sender (**Reply**) or all recipients (**Reply All**) of the email.
  + Add new recipients in the **To** and **CC** fields as required.
  + Attach additional files as needed.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* **Forwarding**:
  + Forward the email, including original attachments.
  + Option to edit the list of attachments (add or remove) before sending.

A screenshot of a computer

Description automatically generated

* **Editing Features**:
  + Modify the **subject line** if required during reply or forward.
  + Add or update recipients in the **To** and **CC** fields.

These features provide users with flexibility and control over email communication, ensuring a smooth and customizable interaction experience.

**4.5. Handling Attachments**

* **Download Attachments**:
  + All attachments in an incoming email can be downloaded directly with a single click.
* **Attachment Management**:
  + While forwarding an email, you can:
    - Include the original attachments.
    - Add new attachments alongside the existing ones.
    - Remove specific attachments if not needed.
  + In **Reply** and **Reply All** options, you can attach files to the email.

**Feedback on Sending**:

* **Success Notification**:
  + If the email is sent successfully, a **pop-up notification** is displayed with a message such as:
    - *"Success: Your email has been sent successfully!"*

A screen shot of a computer

Description automatically generated

* **Error Notification**:
  + If the email fails to send due to any issue (e.g., connectivity or invalid recipient addresses), an **error pop-up notification** is displayed with a message like:
    - *"Error: Failed to send the email. Please try again." Or “blank page”*

**4.6. Incoming and Outgoing Mails Management**

The application provides an intuitive interface to manage **incoming** and **outgoing mails**, ensuring efficient workflow and storage management. Below is a detailed overview of how the application handles these emails:

**Left Container Overview**

* All **incoming** and **outgoing mails** are displayed in the **left container** of the UI.
* Emails are **differentiated by their direction**:
  + **Incoming Mail (↓)**: Emails sent from users to agents.
  + **Outgoing Mail(↑)**:Emails sent from agents to users.
* Mails are stored temporarily in the **cache memory(accepted\_mail key)** for active interactions.

A screenshot of a computer

Description automatically generated

**Agent Interaction Management**

* The **maximum agent interactions** are calculated based on the count of **incoming mails** in the cache memory:
  + If the number of incoming mails in the cache memory is **≥3**, the agent's state changes from **Available** to **Available on Demand**, restricting additional interactions.
* This ensures agents handle their workload efficiently without being overloaded.

A screenshot of a chat

Description automatically generated

**4.7. Wrap-Up Process**

* **Wrap-Up for Incoming Mails**:
  + After replying to a user's email, the agent can **wrap up** the mail.
  + Wrapping up:
    - Permanently **deletes the email from cache memory (accepted\_mail key)**.
    - **Stores the email in PostgreSQL** for record-keeping.
* **Wrap-Up for Outgoing Mails**:
  + Outgoing mails can also be wrapped up by the agent.
  + These mails are also stored in PostgreSQL after being wrapped up.

**History of Wrapped-Up Mails**

* All wrapped-up emails (incoming and outgoing) are stored in PostgreSQL.
* To view wrapped-up emails:
  + Click on the **"History"** section in the UI.
  + The application fetches all wrapped-up emails (incoming and outgoing) from PostgreSQL in the **order of wrapping up**.

A screenshot of a computer

Description automatically generated

This systematic management ensures a clear distinction between active and completed interactions, improving efficiency while maintaining a detailed record of email communications.

**4.8. Rejected or Unaccepted Mail Handling**

The application efficiently manages **rejected** and **unattended emails** to ensure no communication is lost. Below is a detailed explanation of how rejected and unattended emails are handled:

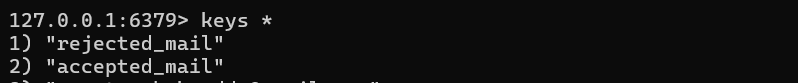
**Rejected Mail Process**

* **Rejection by Agent**:
  + If an agent rejects an incoming email, it is stored under the rejected\_mail key in the **cache memory**.
  + This action ensures the email remains in the system for reassignment.

A screenshot of a computer

Description automatically generated

* **Backend Check**:
  + The backend performs a continuous check on the rejected\_mail key every **15 seconds**:
    - If there are emails in the key:
      * They are reassigned **one by one** to the next **available agent**.
      * The email is removed from the rejected\_mail key after successful reassignment.
    - If the key is empty:
      * The system continues to loop and check for new emails.



**Unattended Mail Process**

* **Timeout Handling**:
  + If an incoming email remains **unattended for 15 seconds**, it is automatically marked as rejected.
  + The email is added to the rejected\_mail key in the cache memory.
* **Reassignment Process**:
  + Similar to rejected mails, unattended emails are reassigned to available agents following the steps outlined above.
* **Continuous Monitoring**
* The backend ensures that:
  + The rejected\_mail key is continuously monitored, even if it is empty.
  + No rejected or unattended email remains unresolved.

A black background with white text

Description automatically generated

This workflow guarantees that every user email is eventually attended to, enhancing responsiveness and maintaining service quality.

**4.9. Composing a New Mail**

The application allows users to send new emails directly by following a simple and user-friendly process. Below are the steps to compose and send a new mail:

**Steps to Compose a New Mail**

* **Initiating a New Mail**:
  + Click on the **Compose** button to open a new mail interface.
* **Entering Recipients**:
  + Provide the recipient email addresses in the **To** field.
  + Optionally, add additional recipients in the **CC** field.
* **Composing the Mail**:
  + Write the email content in the body of the mail.
  + Add **attachments** if required (e.g., images, PDFs, text files, Excel sheets).
  + Attachments are optional and can be skipped if not needed.
* **Sending the Mail**:
  + Click the **Send** button to dispatch the email.

A screenshot of a computer

Description automatically generated

**Post-Sending Feedback**

* **Success Notification**:
  + If the email is sent successfully, a **pop-up notification** appears with a message:
    - *"Success: Your email has been sent successfully!"*
* **Failure Notification**:
  + If the email fails to send, a **pop-up notification** appears indicating an blank screen or error:
    - *"Error: Failed to send the email."*

This feature ensures that users can easily initiate and send new emails independently of replies, reply-all, or forwards, providing a versatile and efficient communication experience.

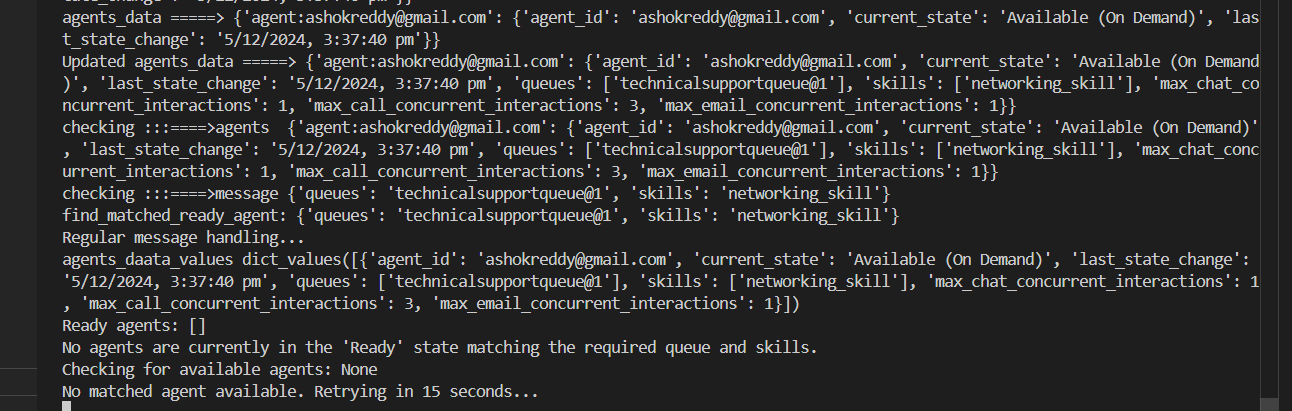
**4.10. Handling New Mails in the SMTP Server When No Agents Are Available**

The backend system is designed to manage scenarios where new mails arrive but no agents are available to handle them. Here is a detailed explanation of how this process works:

**Process Workflow**

* **New Mails Arrival**:
  + When new mails arrive in the **SMTP server**, the backend checks for available agents to assign these mails.
* **No Agents Available**:
  + If no agents are available at the time:
    - The backend **waits for 15 seconds** before re-checking for available agents.
* **Continuous Monitoring**:
  + The backend enters a **loop**, continuously searching for agents every 15 seconds.
  + This ensures no incoming mail is dropped or left unattended.
* **Agent Becomes Available**:
  + As soon as an agent becomes available:
    - The backend resumes the process of assigning mails to agents.
    - The available agent is immediately assigned the next mail from the queue.

This mechanism ensures a seamless workflow, prioritizing responsiveness while maintaining efficient mail handling, even during high workloads.



**5.Chat Module Overview**

The chat module operates seamlessly alongside the email module, offering real-time messaging capabilities. Below is an outline of how the chat module works:

**Setup and Initialization**

* **Running the Backend**:
  + In addition to the **email backend**, you need to start the **chat backend files**.
  + Ensure both the **email** and **chat backends** are running simultaneously.
* **Starting the Frontend**:
  + Once the **frontend** application is launched, the **chat component** becomes accessible alongside the email functionality.

**5.1. Chat Component**

* The chat component provides a user interface for real-time communication.
* It integrates with the email module backend, allowing seamless toggling between chat and email interactions.

This modular approach ensures smooth operation and integration of chat and email functionalities, enhancing user experience by providing real-time communication within the same application environment.

A screen shot of a computer

Description automatically generated

**6.Technology Stack**

The following technologies are used in the development of the application:

**6.1. Programming Languages:**

* **Python**:
  + Used for backend development, handling server-side logic, and managing database interactions.
* **React.js**:
  + Used for frontend development to create a dynamic and responsive user interface.

**6.2.Databases:**

* **PostgreSQL**:
  + A robust relational database system used for permanent storage of wrapped-up mails and other application data.
* **Redis**:
  + A high-performance in-memory data store used for temporary storage, such as cache memory for managing active and rejected mails.

This technology stack ensures the application is scalable, efficient, and user-friendly, with a strong emphasis on performance and reliability.

**7.User Registration for Chat**

To enable users to interact with agents via chat, they must first complete a registration process. Below is a detailed explanation:

**User Registration Process**

**Input Required**:

* + Users must provide the following details during registration:

**First Name** ,**Last Name**, **Email Address**, **Phone Number**

**Access to Chat**

* **Custom Chat Page**:
  + After successful registration, users are redirected to a **custom chat page** to communicate directly with an available agent.

This process ensures that all users are properly identified and registered before initiating a chat session, enhancing both user experience and operational efficiency.

A screenshot of a chat box

Description automatically generated A screenshot of a phone

Description automatically generated

**8.Handling User Messages in Specific Queues**

When a user sends a message associated with a specific queue, the backend system ensures the message is routed to an available agent within the same queue.

**Message Routing**:

* + When a message is sent from a user:
    - The backend identifies the **specific queue**.
    - It searches for an **available agent** assigned to the same queue.
  + If an agent is available, the message is delivered, and the agent receives a **pop-up notification**.
  + The notification includes essential details such as:
    - **First Name**: Identifies the sender.
    - **Phone Number**: Provides the phone number of user.
    - **Action Buttons**: Options to **Accept** or **Reject** the email.

**Agent State Management**:

* + Upon receiving the incoming email notification, the agent's state changes from **Available** to **Available on Demand**.
  + In this state, the agent is prevented from receiving other chats, calls, or emails, ensuring undivided attention to the current interaction.
  + This restriction guarantees that the agent handles only **one incoming item** at a time.

A screenshot of a computer

Description automatically generated

* **Chat Handling**:
  + Once the agent **accepts** the chat:
    - The chat is listed in the **left panel** of the interface.
    - Clicking on the chat in the left panel opens its details in the **right panel**, allowing the agent to view and respond.

A white rectangular object with a white border

Description automatically generated

* **Returning to Availability**:
  + After addressing the interaction, the agent's availability is reassessed.
  + The agent becomes **Available** again if their total interactions (including chats, calls, and emails) are fewer than **3**.

This structured approach ensures that agents manage their workload effectively, providing focused attention to each user while maintaining optimal productivity and service quality.

A screenshot of a computer

Description automatically generated

**9.Chat Handling Workflow for Agents:**

**9.1. Message Handling**

**Message Acceptance**:

* + When an agent accepts a user's message:
    - The message is temporarily stored in the **conversation** key in the cache memory for easy retrieval and tracking.
  + The agent can then reply to the message and continue the conversation.

**Continuous Message Flow**:

* + Once the agent accepts a chat, they can continuously receive messages from the user in real-time.
  + All messages exchanged are appended to the ongoing conversation stored in the cache.

**9.2. Attachment Support**

* Attachments can be shared seamlessly in the chat, both by the user and the agent.
* Supported file types include:
  + **Images**: .jpg, .jpeg, .png
  + **Text Files**: .txt
  + **PDFs**: .pdf
  + **Excel Sheets**: .xlsx
* Attachments are handled as part of the conversation and are displayed in the chat interface.

A screenshot of a computer

Description automatically generated

**9.3. Agent Capacity**

* Agents are allowed to handle **multiple chats** simultaneously, subject to a maximum limit of **3 interactions**. These interactions include:
  + Chats
  + Incoming emails
* When the maximum interaction limit of 3 is reached:
  + The agent's status changes to **available on demand**, ensuring they are not overloaded with additional tasks.

**Summary**

This workflow ensures efficient management of real-time user-agent interactions by:

* Temporarily storing conversations in cache memory.
* Supporting continuous messaging and attachments.
* Balancing agent workload by limiting interactions to a maximum of 3.

**9.4. Chat Session Management and Interaction Count**

**Interaction Count Management**

* **Chat Session Count**:
  + The **count of interactions** for chats is based on the **number of active chat sessions** stored in the cache memory.
  + Each active session contributes to the total interaction count of an agent.
* **Reducing Interaction Count**:
  + When a chat session ends, the corresponding interaction is removed from the agent's total count, ensuring accurate workload tracking.

**Ending a Chat Session**

* **Closing the Chat**:
  + Users can close the chat session when they feel the conversation has ended.
  + To close a chat:
    - Hover over the **three dots** at the top of the chat window.
    - A **pop-up notification** will appear, prompting the user to confirm closing the chat.
    - Upon confirmation, the chat session will be marked as closed.

A screenshot of a chat

Description automatically generated

* **Data Handling Upon Closure**:
  + The closed chat session is:
    - **Deleted** from the cache memory.
    - **Persisted** in the PostgreSQL database for long-term storage.
  + This ensures that temporary data does not overload the cache while retaining historical records.

A computer screen with text

Description automatically generated

**Effect on Interaction Count**

* When a chat session is closed:
  + The **interaction count** for the agent is reduced by one.
  + This allows the agent to take on a new interaction (chat, email, or call) if they are under the maximum limit of **3 interactions**.

**Summary**

This system provides efficient chat session handling by:

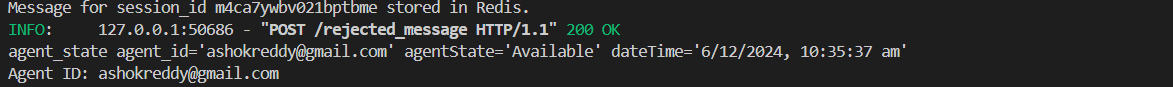
* Accurately tracking interactions based on active chat sessions in cache memory.
* Allowing users to close chats intuitively and ensuring proper data storage in PostgreSQL.
* Dynamically adjusting the agent's interaction count to balance workload.

**9.5. Rejected or Unaccepted Chat Handling**

The application efficiently manages **rejected** and **unattended chats** to ensure no communication is lost. Below is a detailed explanation of how rejected and unattended chats are handled:

**Rejected chats** **Process**

* **Rejection by Agent**:
  + If an agent rejects an incoming chat, it is stored under the agent\_rejected\_message key in the **cache memory**.
  + This action ensures the chat remains in the system for reassignment.



* **Backend Check**:
  + The backend performs a continuous check on the agent\_rejected\_message key
    - If there are chats in the key:
      * They are reassigned **one by one** to the next **available agent**.
      * The chat is removed from the agent\_rejected\_message key after successful reassignment.
    - If the key is empty:
      * The system continues to loop and check for new chats.

**Unattended Mail Process**

* **Timeout Handling**:
  + If an incoming chat remains **unattended for 30 seconds**, it is automatically marked as rejected.
  + The chat is added to the agent\_rejected\_message key in the cache memory.
* **Reassignment Process**:
  + Similar to rejected chats, unattended chats are reassigned to available agents following the steps outlined above.

**Continuous Monitoring**

* The backend ensures that:
  + The agent\_rejected\_message key is continuously monitored, even if it is empty.
  + No rejected or unattended chat remains unresolved.

This workflow guarantees that every user chat is eventually attended to, enhancing responsiveness and maintaining service quality.