Deliverable #2

SE 3A04: Software Design II – Large System Design

Tutorial Number: T01
Group Number: G8
Group Members:

- Kyle Hagerman
- Michael Breau
- Zongcheng Li
- Jay Sharma
- Ahmed Al-Hayali

IMPORTANT NOTES

- Please document any non-standard notations that you may have used
  - *Rule of Thumb*: if you feel there is any doubt surrounding the meaning of your notations, document them
- Some diagrams may be difficult to fit into one page
  - Ensure that the text is readable when printed, or when viewed at 100% on a regular laptop-sized screen.
  - If you need to break a diagram onto multiple pages, please adopt a system of doing so and thoroughly explain how it can be reconnected from one page to the next; if you are unsure about this, please ask about it
- Please submit the latest version of Deliverable 1 with Deliverable 2
  - Indicate any changes you made.
- If you do NOT have a Division of Labour sheet, your deliverable will NOT be marked

1 Introduction

1.1 Purpose

This document presents an overview of the Secure Chat (SC) system architecture. It encompasses a high-level evaluation of the system's design considerations and outlines key points for its various subsystems.

As the document builds upon the requirements specification, it is advised that readers refer to Deliverable 1 for context regarding the application. The anticipated audience includes, but is not limited to, product managers, developers, domain experts, and SC team members.

1.2 System Description

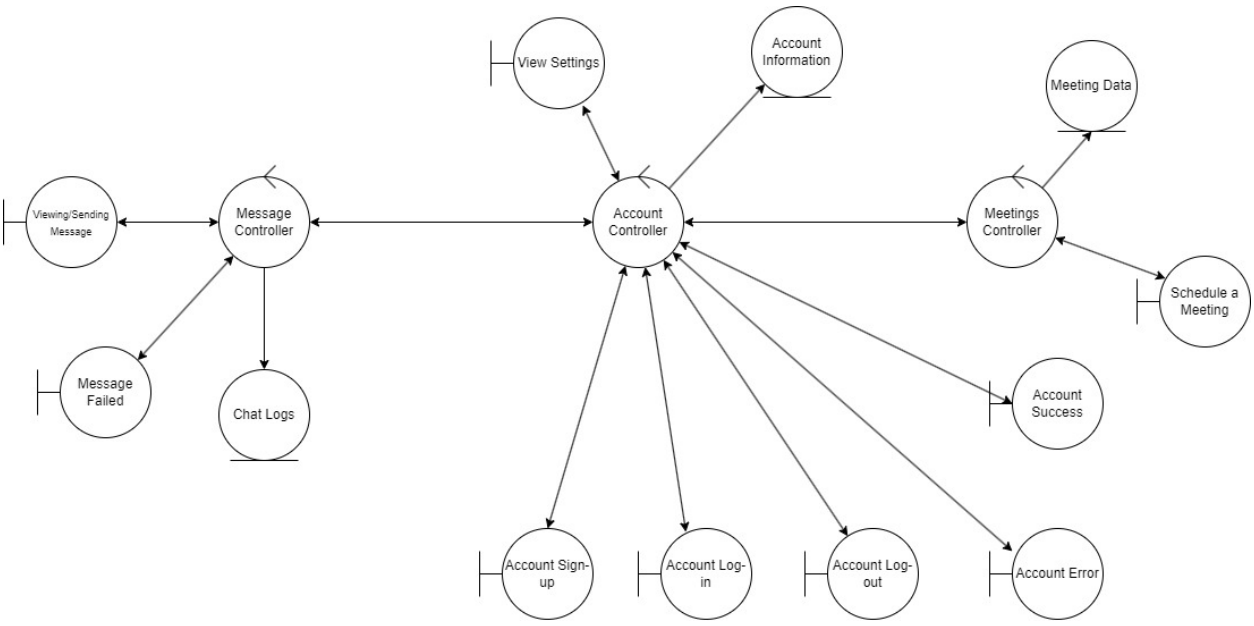
Secure Chat (SC) employs an interaction-oriented architecture, specifically the Presentation-Abstraction-Control (PAC) style, with subsystems leveraging a Repository style architecture.

Using a PAC architecture supports an agent based hierarchical structure with a clear separation of concerns within the system. On the other hand, a Repository style architecture supports the use of a passive data store with independent subsystems operating on the central data structure.

1.3 Overview

In Section 2, we present an Analysis Class Diagram for the application. Moving forward in Section 3, we discuss the architectural design of the system, and alternatives that were considered. Lastly, Section 4 showcases the Class Responsibility Collaboration (CRC) Cards and goes into detail about classes, responsibilities, and their coupling with other classes.

2 Analysis Class Diagram



3 Architectural Design

3.1 System Architecture

Secure Chat is an interaction-driven application, hence use of the hierarchical presentation-abstraction-control (PAC) architecture, with the integration of data elements using the repository data-centred architecture, is sound. The system's primary functionality can be captured by the subsystems summarized below:

Subsystem	**Purpose**	**Architecture Style**
Account Management	Create, log in, and log out of account	Repository
Message Management	Securely send and receive messages	Repository
Meeting Management	Create, join, modify, or cancel meetings	Repository

Table 1: Subsystems

Without the strict requirement for interaction between the data and display layers, additional separation of concern can be attained by using the PAC architecture, assigning various subsystems to corresponding PAC agents. Provided the live-messaging use-case, it is within reason to expect that presentations' views must strictly exist with a high degree of coupling to corresponding controllers, facilitating agent specificity, i.e., ensuring that changes to a view occur only after a controller event, e.g., typing, is invoked by a user. A favourable consequence of the hierarchical structure of the PAC architecture is its extensibility and forward compatibility, facilitating the introduction of features through new lower-level agents without modification of existing agents [1].

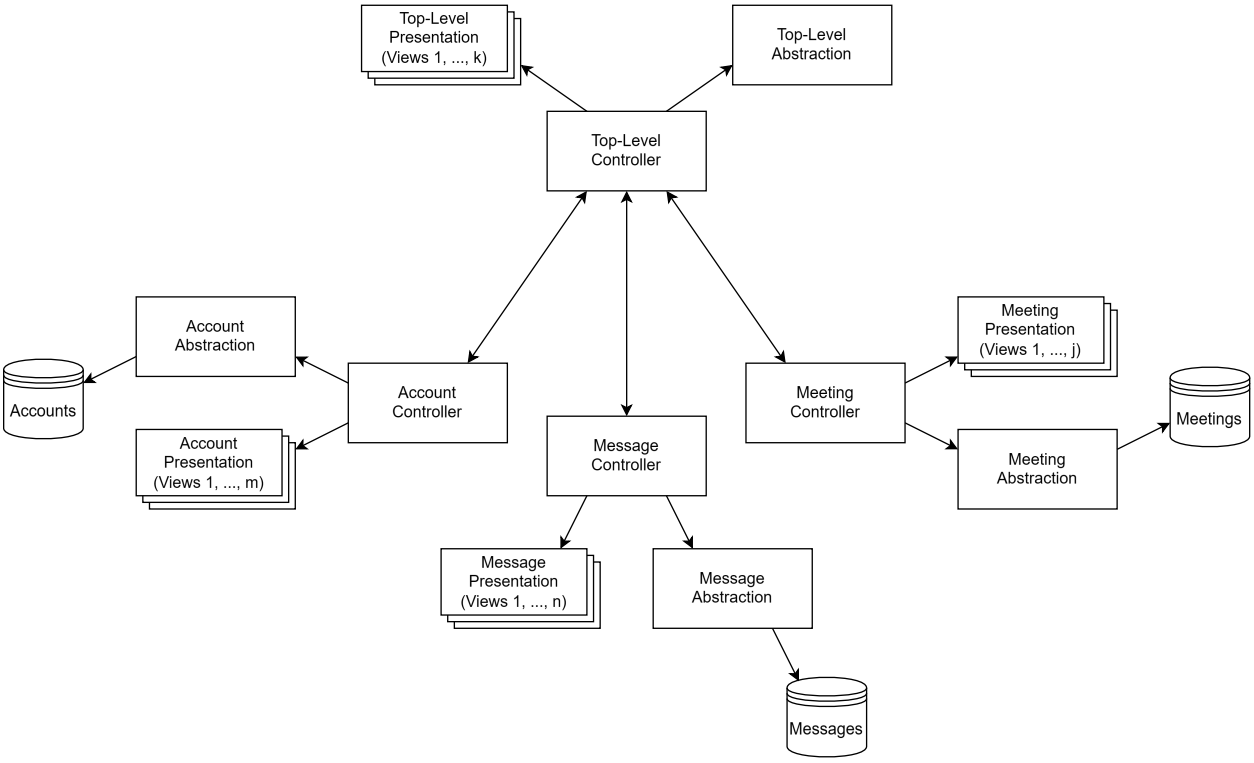


Figure 1: *Secure Chat* Initial System Architecture

The PAC architecture offers multiple views of the same abstraction or model under a presentation layer, as does the model-view-control (MVC) architecture, but it is achieved with communication through the controller instead of between the model and view. The benefit of MVC is synchronization across different

dynamic instances of views through model-view communication, but that is impractical in a live-messaging application - the view would contain authorization and authentication business logic, which is better suited to be used in the controller.

Obviously, passive storage of data, e.g., client accounts, messages, and meetings, necessitates a passive data store, imploring use of the repository architecture. The repository architecture delivers reliable data access to reusable components, i.e., agents given its use with the PAC architecture, with minimal implementation overhead. The necessity of a passive data store in of itself makes considering the blackboard architecture imprudent.

Of the relevant data-flow architectures, process-control acts on analog signals, of which there are none in this system, hence its omission; the pipe-and-filter architecture would be of minimal use as the messaging use-case does not call for any streaming-data throughput constraints whatsoever; batch-sequential data processing can be used to implement a chat filter, but such an architectural modification would further complicate the system, hence the feature is omitted.

3.2 Subsystems

Provide a list of your subsystems, with a brief description of each. Be sure to document its purpose and relationship to other subsystems.

Our application has been divided into 3 subsystems, Account, Meetings Management, and Messages.

- **Account:** This subsystem controls all account settings and logins, including validating user identities for logging into the system.
  - **Purpose:** The account subsystem is required to allow users to log in and out of their accounts as well as change their settings. It is also used to ensure the security that is key to **SC**, using secure communication protocols for chat as well as providing authentication so that all accounts are only logged into by the employee it belongs to.
  - **Relationships:**
    - **Message:** The account system is required to ensure secure messages are sent in **SC**. Due to security being one of the greatest concerns for the overall system, the use of authentication with messaging is of utmost importance.
    - **Meetings Management:** The account system will be used to ensure the authenticity of meeting invites, as you must be logged in and authenticated to send an invite as well as accept one.
- **Meetings Management:** This subsystem stores information about scheduled meetings between users and controls the setup and notification system for scheduling meetings as well as reminding users of their meetings.
  - **Purpose:** The meeting management subsystem is required to facilitate the innovate feature of users scheduling meetings with each other.
- **Message:** This subsystem controls the use of messaging between users and the storing of messages.
  - **Purpose:** The messaging subsystem is required to perform one of the main functionalities of the system: messaging. It includes a large portion of the user interface as well, since users will mainly be interacting with the messaging screen while other subsystems do their work in the background.
  - **Relationships:**
    - **Meetings Management:** The messaging system will add functionality to the meetings management system by allowing meetings to be sent as messages in an open chat, as well as being sent to someone else as a message they can later view.

4 Class Responsibility Collaboration (CRC) Cards

Class Name: Message Controller	
Responsibility:	**Collaborators:**
Handle Viewing/Sending Message	Viewing/Sending Message
Notify user the Message Failed	Message Failed
Allow user to get Chat Logs	Chat Logs
Store message data of related accounts	Account Controller

Class Name: Viewing/Sending Message (Boundary)	
Responsibility:	**Collaborators:**
Handle Viewing/Sending Message	Message Controller
Allow users to view/send message	

Class Name: Message Failed (Boundary)	
Responsibility:	**Collaborators:**
Notify user the Message Failed	Message Controller
Handles message failed screen	

Class Name: Chat Logs (Entity)	
Responsibility:	**Collaborators:**
Provide users chat log Store chat logs Store chat log participants	Message Controller

Class Name: Accounts Controller	
Responsibility:	**Collaborators:**
Store Account Information	Account Information
Handle authentication based on Account Information	Account Information
Allow users to change View Settings	View Settings
Allow Account Sign-up	Account Sign-up
Allow Account Log-in	Account Log-in
Allow Account Log-out	Account Log-out
Notify Account Error	Account Error
Notify Account Success	Account Success
Handle Viewing/Sending Message	Message Controller
Inform user of meetings	Meetings Controller

Class Name: Account Information (Entity)	
Responsibility:	**Collaborators:**
Provide Account Information Store employee ID Store username Store corresponding password	Accounts Controller

Class Name: View Settings (Boundary)	
Responsibility:	**Collaborators:**
Notify change of view settings Allow users to change View Settings	Accounts Controller

Class Name: Account Sign-up (Boundary)	
Responsibility:	**Collaborators:**
Notify creation of new account Allows users to sign-up for an account	Accounts Controller

Class Name: Account Log-in (Boundary)	
Responsibility:	**Collaborators:**
Notify account log-in Allows users to log-in to their account	Accounts Controller

Class Name: Account Log-out (Boundary)	
Responsibility:	**Collaborators:**
Notify account log-out Allows users to log-out of their account	Accounts Controller

Class Name: Account Error (Boundary)	
Responsibility:	**Collaborators:**
Notify authentication error or creation error Handles authentication error event Handles creating account error event	Accounts Controller

Class Name: Account Success (Boundary)	
Responsibility:	**Collaborators:**
Notify authentication or creation success Let users view authentication success information Handles creating account success information	Account Controller

Class Name: Meeting Controller	
Responsibility:	**Collaborators:**
Create new meetings Notify user of meetings Add meetings to user account	Meeting Data Account controller Account controller

Class Name: Schedule Meeting (Boundary)	
Responsibility:	**Collaborators:**
Lets users schedule meetings Shows users a confirmation when meetings are accepted	Meeting Controller

Class Name: Meeting Data (Entity)	
Responsibility:	**Collaborators:**
Store meeting time Store meeting participants	Meeting Controller Meeting Controller

A Division of Labour

Include a Division of Labour sheet which indicates the contributions of each team member. This sheet must be signed by all team members.

- **Kyle:**
  - Completed section 3.2 - Subsystems
  - Added items to analysis class diagram brainstorming
  - Create meeting related CRC cards, added authentication to Accounts Controller CRC card
  - Minor spelling/formatting/wording fixes

  KH
- **Michael:**
  - Analysis class diagram brainstorming
  - Completed section 2 - Analysis class diagram
  - Completed CRC Cards for Message Controller as well as the collaborator cards for the controller

  MD
- **Jay:**
  - Completed Introduction section (1.1, 1.2, 1.3)
  - Completed section 2 - Analysis class diagram
  - Completed CRC Cards for Accounts Controller as well as the collaborator cards for the controller

  J.S
- **Ahmed:**
  - Developed, documented, and illustrated architecture – entirety of section 3.1

  A.A.
- **Zongcheng:**
  - Contributed to section 3.1 - System Architecture
  - Contributed to some CRC cards of responsibility, added Account Success CRC card

  Z.L.

References

[1] D. Plakalovic and D. Simic, "Applying mvc and pac patterns in mobile applications," *arXiv preprint arXiv:1001.3489*, 2010.