Deliverable #2 Template

SE 3A04: Software Design II – Large System Design

Tutorial Number: T0x
Group Number: Gx
Group Members:

- List all Group Member Names (as listed in Avenue)
- You do not need to use student #s or macid (keep those private).

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 provides a description of the architectural design decisions of the Secure Chat internal messaging application. Included are analysis class diagrams for the messaging and database systems that make up the functionality of the application.

This document is intended for internal stakeholders, including but not limited to, project managers, developers, domain experts, and team members/investors. Deliverable 1 should be read prior, and technical knowledge may be beneficial in better understanding the contents of the document.

1.2 System Description

Secure Chat is an Android-based internal messaging application made for workplace communication on company-issued devices, operates on a client-server architecture for efficient division into client and server components, enabling scalability. Additionally, it adopts a repository-style architecture, ensuring the distinct separation of database logic from the overall application structure.

1.3 Overview

Section 1: Introduction and Context
This section serves as an introductory overview, providing essential context for the subsequent sections of the document.

Section 2: Analysis Class Diagram
In Section 2, an in-depth analysis class diagram is presented, offering a visual representation of the various classes within the application and the intricate relationships existing among them.

Section 3: Architectural Design
Section 3 delves into the architectural design chosen for our system, elucidating the strategic division of subsystems to ensure a cohesive and efficient structure.

Section 4: Class Responsibilities - CRC Cards
Section 4 systematically assigns CRC (Class-Responsibility-Collaboration) cards to each class identified in the analysis class diagram. These cards delineate the responsibilities of each class and highlight their interdependencies with other classes.

Section A: Division of Labour
Concluding the document, Section A provides a comprehensive Division of Labour, outlining the individual contributions of each team member to the creation of this document.

2 Analysis Class Diagram

This section should provide an analysis class diagram for your application.

3 Architectural Design

This section will provide an overview of our architecture and the reason we chose it.

3.1 System Architecture

The SCAA uses a Data Centred Software Architecture, mainly the Repository Architecture style. The Repo is where all of the data is stored and is connected to different management subsystems which are all invoked through the user.

Subsystem	Purpose	Architecture Style
Chat Manager	Create, manage and secure chats	Repository and Pipes and Filters
Account Manager	Create, edit, and delete accounts	Repository
Announcement Manager	Create, secure and manage announcements	Repository

Table 1: Subsystem Architecture Styles

In addition four databases are present in the Repo part of the architecture, a chat database, account database, announcement database, and a KDC database. These are to ensure that everyone's respective information and all the chat information is stored.

For this project, we chose the repository architecture due to its perfect fit for our needs in creating a data-centred, secure chat application. This architecture style is the best option as it supports active user engagement with a more structured and passive data management approach, which is crucial for maintaining the integrity and security of communication. The repository model offers a centralised data management system that facilitates efficient data access, sharing, and updating across the application, ensuring that all interactions are consistently secure and coherent.

Moreover, this architecture allows for a scalable and modular design, enabling easy integration of additional features or updates without disrupting the existing system. It supports concurrent access and modification of data, a critical requirement for a real-time chat application where multiple users interact simultaneously. The centralised control over data also simplifies the implementation of comprehensive security measures, including the management of encryption keys and authentication protocols, further aligning with our project's stringent security requirements.

Additionally, for the specific task of decryption and encryption within our system, we've decided to integrate a pipe and filter architecture into its own subsystem (the Chat Management subsystem). This decision stems from the architecture's proven effectiveness in processing streams of data in real-time, offering a flexible and efficient way to handle the decryption process. This modular approach allows for seamless data flow and transformation, ensuring that decrypted messages are promptly and securely delivered, enhancing the overall performance and reliability of our secure communication solution.

As for pipes and filters we didn't go with this option since there are many limitations to using this for a secure chat app.The pipes and filters architecture is used for processing data through a series of transformations (filters), each performing a specific operation, connected by pipes (data channels). Pipes and Filters focus on data processing rather than centralised data management, potentially complicating the enforcement of security policies across multiple stages. Real-time, stateful interactions required for a chat system are not the primary focus of this architecture, making it less suitable as the core architecture for the application.

As for batch sequential it also doesn't fit the criteria for this project. The way batch sequential works is it takes data in batches through a sequence of steps, each step completing before the next begins. As for this app it requires real-time communication and demands immediate processing, which doesn't align well with the batch processing nature of this architecture. Adapting batch sequential processing to the dynamic, interactive requirements of a secure chat application can be challenging.

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.

Chat Manager: This subsystem is responsible for creating, managing, and securing chats by communicating with the KDC and chat databases. It adheres to the Repository architecture style since it communicates only through the main repository in our system. It also adheres to the Pipes and Filters architecture style since messages need to go through encryption and decryption filters and they can be delivered to other users

through pipes.

Account Manager: This subsystem is responsible for creating, editing, and deleting accounts. It communicates with the account database to edit and delete accounts and account information. The architecture style this subsystem follows is the Repository style because it communicates with the main repository to access the Account database.

Announcement Manager: This subsystem is responsible for creating, securing, and managing announcements. It communicates with the KDC and announcements databases to access user authentication. Since it follows the repository architecture, it communicates to these databases only through the main repository.

4 Class Responsibility Collaboration (CRC) Cards

This section should contain all of your CRC cards.

- Provide a CRC Card for each identified class
- Please use the format outlined in tutorial, i.e.,

Class Name:	
Responsibility:	**Collaborators:**

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.