# *Term Project:* *Instant Messaging App*

Architecture Document

Table of Contents

1 Introduction 2

1.1 Purpose 2

1.2 Target Audience 2

1.3 Terms and Definitions 2

2 Architectural Goals and Constraints 3

2.1 Security 3

2.2 Persistency 4

2.3 Reliability/Availability 4

3 System Overview 5

4 System Architecture 8

4.1 Front-end Architecture 8

4.2 Back-end Microservices Architecture 11

5 Database Design 13

6 Issue and Concerns 15

# Introduction

**Provide an overview of this architecture document and the product it describes.**

First, the key goal of this architecture document is to explain in detail to the reader the overall system design of an instant messaging app and how the frontend and backend of the instant messaging app are designed and implemented.

## Purpose

**Provide a high-level description of the purpose of this document. How will this document help you or others when developing similar product?**

As mentioned above, the core goal of this architecture document is to explain to the reader in detail the overall system design of an instant messaging app as well as how the frontend and backend are designed. Because this architecture document serves to make the instant messaging app developer who is writing this document rethink the overall system design of the application, it can uncover errors in the system design and improve the understanding of the system design. Therefore, this document is of great help to the document writer. Moreover, it can present the overall layout to developers who want to develop similar applications and provide them a smooth start.

## Target Audience

**Describe the target audience for this document.**

The target audience of this architecture document will be development team members and project managers who develop instant messaging app, and development team members and project managers who want to develop similar applications.

## Terms and Definitions

**Define any terms or acronyms you will be using in the remainder of this document.**

# Architectural Goals and Constraints

**Briefly describe the architecture of your product, then give more details respectively in both server side and client side. Tell us the architectural goals on non-functional system properties. (Chapter 4 p6-7) If there are constraints, how they affect or limit the system? (Chapter 4 p15-19)**

This instant messaging app is divided into two parts. It is divided into server and client. The server is the backend of the instant messaging app, and the client is the frontend of the instant messaging app. The server is responsible for connecting to the database, and the client is responsible for various features that users can perform on the interface. For instant messaging app to work normally, server and client must be running at the same time. This app is based on "Stream" which is highly reliable chat infrastructure and feature-rich SDKs. Therefore, the system returns results to users in a reasonable time, the system features behave as expected by both developers and users, the system can deliver its services when requested by users, the system protects itself and users' data from unauthorized attacks and intrusions, system users can access the features that they need and use them quickly and without errors, and the system can be readily updated and new features added without undue costs. However, since this application is created using the chat API called "Stream", the database is very limited, so it cannot handle a large number of users, and whether the system can continue to deliver user services in the event of partial failure or external attack are still unknown.

## Security

**How does the system protect itself and users’ data from unauthorized attacks and intrusions? If it doesn’t, what are the constraints?**

First, to use this application, the user must enter an ID and password and go through the login authorization process to use it. And even the program administrator in the database cannot see the passwords of users and cannot see the chat contents. And since this database is the stream's own database, the stream's own protection exists. However, in the rapidly changing and developing online world, attacks can come in various ways, so I think that this application should have an additional security protection.

## Persistency

**How does the system achieve data persistency?**

First, this instant messaging app is a SaaS Multitenancy application. Multi-tenancy is an architecture in which a single instance of a software application serves multiple customers. Each customer is called a tenant. In this way, a software application in a multi-tenant architecture can share a dedicated instance of configurations, data, user management and other properties. Because each tenant receives data from the reliable and secure database of the chat API platform called "Stream", data persistency is guaranteed.

## Reliability/Availability

**Are there any constraints on the reliability/availability of the system features so that they cannot behave as expected by developers and users?**

**(You can add more topics of your choice.)**

Stream can handle real-time chat as-a-service, with 5 million concurrent connections in a single chat channel. However, it may be limited because it is not a server only for this application, but a server used by several applications in common. And as mentioned above, it is questionable whether it can handle a large number of users because it is given a limited database and is built on that limited database.

# System Overview

**A one-page, high-level system functionality and architecture description goes here. It should at least include the system’s user interface, user interface management, application services, shared infrastructure services and database management. (Chapter 4 p22, 35) Then a layered architectural model (Chapter 4 p21-22, 26-35) should be summarized according to your description about the system.**

User Interface

* Web browser: The user interface is basically in the form of a web browser. It works flawlessly in Chrome or Firefox, and the size of the interface is automatically adjusted according to the size of the web browser.

User Interface Management

* Interface creation: In stream chat application, user interface is composed by importing App.css file with UI design already preset. We can choose from several preset App.css, or we can fully customize App.css file.
* Forms management: In the App.css file, we can specify and set values such as primary color, margin, padding, and height.
* Interface delivery: Interface delivery gets determined on the web browser according to the specified value of each variable in the App.css file.

Application Services

* Create channel: Channels are at the core of Stream Chat. Within a channel, user send/receive messages and interact with other users. Once a channel object has been initialized, the Channel component consumes the object and renders user's chat app's functionality. When creating a channel, user can set whether it is a group type or a direct message. If it is a group type, a group channel is created, and if it is a direct message, a direct message chat room is created.
* Edit channel & Delete channel: The instant messaging app provides a feature to edit and delete created channels. Users can edit the channel name by edit channel button, and users can delete the corresponding channel from the channel list by delete channel button.
* Send message: When entering a group channel or direct message chat room, previous conversations in the chat room are retrieved from the database, and the user can enter text to continue the conversation in the chat room.
* React to message: The Stream Chat API basically provides a feature that allows users to react to messages. Reaction has several emoji options, and users can add a reaction by selecting from the options.
* Thread: The Stream Chat API also provides a thread feature. In a group channel, users can create a thread based on a single message, and users can continue a private conversation in it independently of the group channel.

Shared Infrastructure Services

* Authentication: When the instant messaging app is launched, the sign-up page is displayed first. An input box for entering name, user ID, password, password confirmation, phone number, avatar URL, etc. will appear, and enter it into the input fields to proceed the registration process. And at the bottom of the sign-up page, there are a sign-up button and a sign-in button. If the account already exists, user can go to the login page by clicking the sign in button.
* Search: After logging in, channels and direct messages are listed on the left side of the main page. If there are too many channels and direct messages, there is a search input box above to find them at once. Users can search by entering a channel name in the corresponding input box and jump to that channel quickly.
* Storage: The monthly active users (MAUs) are 1.0K/month and the maximum concurrent connection is 200. Also, the message volume limit is 7.0K, and if we want to upgrade these limits, we need to upgrade the stream chat plan.

Database Management

* Stream Chat Dashboard: Database management can be done on the stream chat dashboard, we can check MAUs, concurrent connections, message volume, etc. mentioned above, and we can manage and supervise all users, channels, and chats.

Layered Architectural Model

* User Interface
  + Web Browser
* User Interface Management
  + Interface Creation, Forms Management, Interface Delivery
* Application Services
  + Create Channel, Edit Channel, Delete Channel, Send Message, React to Message, Thread
* Shared Infrastructure Services
  + Authentication, Search, Storage
* Database Management
  + Stream Chat Dashboard

# System Architecture

**Describe the overview of both frontend and backend system architecture of your project.**

The frontend and backend of an instant messaging app are divided into two large and simple folders. In this project repository, there are a client folder and a server folder. The client folder is the frontend, and the server folder is the backend. In the client folder, there are App.css file and assets folder related to the user interface, App.jsx file, the main code file of the program, and the components folder where various features that users can perform on the interface are implemented. In the server folder, there are a controller folder and a routes folder that allow connection to the database and correct data retrieval in relation to authentication.

## Front-end Architecture

**Provide a one-page, detailed description of the frontend architecture of your system. It should include the frontend framework or library, presentation, routing, functionalities, API components and any cloud-based software or platform (Chapter 5) you have used. Specify if you have used any PaaS or SaaS (Chapter 5 p16).**

Overview

* First of all, this instant messaging app is built on top of Stream Chat API. Building on top of the the Stream Chat API, the Stream Chat React component library includes everything we need to build feature-rich and high-functioning chat user experiences out of the box. The library includes an extensive set of performant and customizable React components which allow us to get started quickly. The Chat and Channel components are React context providers that pass a variety of values to their children, including UI components, stateful data, and action handler functions.

Chat client

* To communicate with the Stream Chat API, Stream Chat client instance must be created with our API key and pass via props into the Chat component. We can get the API key as we sign up for a free 30 day trial on their website.

Connecting a user

* Tokens are used to authenticate a user. Typically, we send this token from our backend to our front end when a user logs in. To connect a user, call the connectUser method on our client instance with the user object and userToken provided as arguments. Connect the user directly after instantiating the client to establish a websocket connection with the Stream Chat API. Once the connection has been opened, our client instance will begin receiving events from the API.

Creating a channel

* Channels are at the core of Stream Chat. Within a channel users send/receive messages and interact with other users. Once a channel object has been initialized, the Channel component consumes the object and renders your chat app's functionality.

Chat

* The Chat component is a React Context provider that wraps the entire Stream Chat application. It provides the ChatContext to its children, which includes the StreamChat client instance. All other components within the library must be nested as children of Chat to maintain proper functionality.

Channel

* The Channel component is a React Context provider that wraps all of the logic, functionality, and UI for an individual chat channel.

Channel List

* The ChannelList component renders a list of channels and provides a preview for each. Though the ChannelList is essential in many chat apps, it isn't a required piece of the library. If a ChannelList component is used, a channel object should not be placed as a prop on the Channel component, as the ChannelList handles channel setting internally.

Window

* The Window component handles width changes in the main channel to ensure a seamless user experience when opening and closing a Thread.

Channel Header

* The ChannelHeader displays pertinent information regarding the currently active channel, including image and title.

Message List

* The MessageList component renders a list of messages and consumes the various contexts setup from Channel. This component accepts a wide variety of optional props for customization needs.

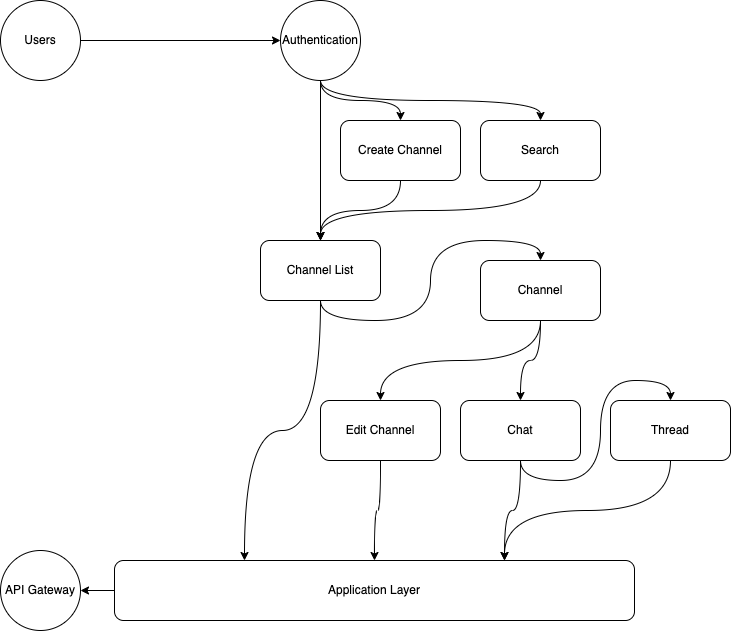
Message Input

* The MessageInput component is a React Context provider that wraps all of the logic, functionality, and UI for the message input displayed in a channel. It provides the MessageInputContext to its children.

Thread

* The Thread component renders a list of replies tied to a single parent message in a channel's main message list. A Thread maintains its own state and renders its own MessageList and MessageInput components.

**Then summarize your description into an architecture diagram.**



## Back-end Microservices Architecture

**Provide a one-page microservices (Chapter 6 p4-23) architecture design of your backend. You should specify how you break down coarse-grain features into microservices, and how microservices communicate with each other. If they all share one centralized database, how are you going to propagate data changes to services that have overlap in data usage? Please always balance fine-grain functionality and system performance. Besides, the description should also include API gateway, and any cloud-based software or platform (Chapter 5) and IaaS, PaaS, SaaS (Chapter 5 p16) you have used.**

Overview

* For the average Stream integration, the development work focuses on code that executes in the client. The React, React Native, Swift, Kotlin or Flutter SDKs connect to the chat API directly from the client. However, some tasks must be executed from the server for safety and those are generating user tokens and syncing users.

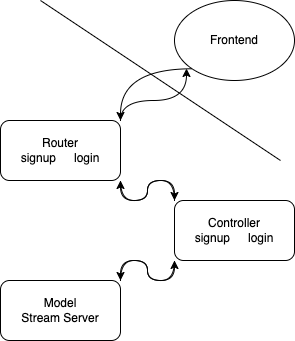
Generating user tokens

* The backend creates a token for a user. It hands that token to the client side during login or registration. This token allows the client side to connect to the chat API for that user. Stream's permission system does the heavy work of determining which actions are valid for the user, so the backend just needs enough logic to provide a token to give the client-side access to a specific user.

Syncing users

* When a user starts a chat conversation with another user both users need to be present in Stream's user storage. So, we need to make sure that users are synced in advance. The update user’s endpoint allows us to update 100 users at once. The thing is that user roles can only be changed server side. The role we assign to a user impacts their permissions and which actions they can take on a channel.

**Then summarize your description into an architecture diagram.**



# Database Design

**This section should describe the design of the database or data hosting environment. You should include which database you have chosen and what is the advantage of using it. Describe your database design and tell us how the design achieves interoperability between the user interface and the background data. The database design should include all the data utilized by your product.**

Data hosting environment

* This Stream Chat API-based instant messaging app uses Stream's own database as well. Database information can be checked on the Stream Dashboard, and all data except private data can be added, modified, or deleted through the Stream Dashboard. The stream's database is a shared database, which means that all users have access to data, and only the data that pertains to them is retrieved. Because it allows for immediate updates and can be used to assert a common data schema, a shared database is a more convenient way of exchanging data than file transfer. Although remote procedure invocation makes it easier to isolate and share functionality, it does not completely eliminate system coupling.

Data design

* The data stored in the database is largely divided into channels and users. Channel-related data exists as sub data in channels data, and user-related data exists as sub data in users data.

Channels

* In the channels data object, there are two data objects, members and messages, and there are a total of 14 channel sub data. Channel sub data are id, type, cid, last\_message\_at, created\_at, updated\_at, created\_by, frozen, disabled, member\_count, config, own\_capabilities, hidden, and name. And the sub data of the members data object are id, role, created\_at, updated\_at, last\_active, banned, online, name, image, fullName, phoneNumber, and hashedPassword. And the sub data of the messages data object are id, text, html, type, user, attachments, latest\_reactions, own\_reactions, reaction\_counts, reaction\_scores, reply\_count, cid, created\_at, updated\_at, shadowed, mentioned\_users, silent, pinned, pinned\_at, pinned\_by, and pin\_expires.

Users

* The sub data of the users data object are id, role, created\_at, updated\_at, last\_active, banned, online, language, shadow\_banned, email, last\_name, first\_name, staff\_user, and dashboard\_user.

# Issue and Concerns

**Describe the issue and concerns you met with during product development.**

Lack of project experience

* This instant messaging app was a difficult project for me to complete. First and foremost, it was challenging at first because I had no prior experience developing applications with this many requirements and functions, nor did I have this level of software development experience. Furthermore, because it was a solo project, it was difficult to find a solution when I ran into problems, and because I was unfamiliar with the JavaScript language, it took a long time because I was coding while searching the web for the code.

Heavily rely on Stream

* Also, one concern is that this instant messaging app is heavily relying on Stream. It is true that high-quality features are implemented in a short time by building components using Stream's Chat API, but this application is not driven entirely by the developer but is driven by the Stream's database with the Stream's Chat API. And the database provided by Stream has limited capacity and concurrent connections. So, if we want to handle a large number of users, we need to upgrade Stream's monthly plan.