Juhwan Lee
CS300: Elements of Software Engineering
Professor Fei Xie

Instant Messaging App Final Report

Introduction

The term project of the Elements of Software Engineering class was to build and deploy an Instant Messaging App using JavaScript, VScode, and GitHub. As someone who has only taken lower division computer science classes so far, software product development has come to me unfamiliar. Despite learning about software product development in the Elements of Software Engineering class, I lacked basic knowledge about frontend, backend, database, JavaScript, and Github, and I was at a loss as to how to start a term project. In the meantime, I found a full-stack chat application development video uploaded by a user called JavaScript Mastery on YouTube, and through this video, I was able to study how to develop a chat application using React and Stream Chat API and therefore I was able to successfully complete the term project.

React

React is a front-end library provided by Facebook. It can be used as a basis for developing single page applications or mobile applications. In other words, it can be seen as a popular library that allows us to develop a web/app view, which is currently being used a lot. Even if we don't use react, we can make web pages using html, css, and javascript, but it seems to be used a lot because we can easily create a dynamic UI that can interact with users using react. What I felt while building an Instant Messaging App with React is that React has many advantages and is particularly useful for developing SaaS type software such as an Instant Messaging App. The advantages of React are:

1. Data Flow
2. Component-based structure
3. Virtual DOM
4. Props and State
5. JSX

Data Flow

React has a one-way data flow, where the flow of data only flows in one direction. Two-way data bindings such as Augular.js tend to become more complex and difficult to track the flow of data as they get larger (for large applications), so even in complex apps, it is possible to create a one-way flow to make changes in the data flow more predictable.

Component-based structure

Component is an independent unit of software module. In other words, it can be seen as a way to make software into one independent part. React creates a UI (View) by splitting it into several components. Even within one page, each part is made into an independent component, and the screen is composed by assembling these components. Since it is broken down into components, it is relatively easy to understand the entire code. Because code is managed by encapsulating it in functional units and UI units, reusability is high. Therefore, there is no need to input the code repeatedly, and there is the convenience of

importing and using only the components, and it has the advantage that the maintenance and management of the code becomes easy even if the application becomes complex.

Virtual DOM
First, DOM stands for Document Object Model. The DOM recognizes html, xml, and CSS as a tree structure, and treats and manages data as objects. React has a structure like this DOM Tree structure as a Virtual DOM. Virtual DOM refers to a virtual Document Object Model. It is said that the efficiency and speed of the app can be improved by creating a virtual DOM every time an event occurs, comparing it with the real DOM every time it is redrawn, and comparing the before and after states, and reflects only the minimal changes that require changes to the real DOM.

Props and State
Props refers to data passed from parent component to child component. We can easily think of it as read-only data. The props received from the child component cannot be changed, and only the top-level parent component that passed the props can change the props. State is declared inside the component and its value can be changed inside. State is used when dealing with dynamic data, and is used to dynamically change data through user interaction. It can only be used in class-type components, and each state is independent.

JSX
JSX is a grammar that extends Javascript. React accepts the fact that rendering logic is intrinsically linked to other UI logic: how events are handled, how state changes over time, and how data is prepared for display on the screen. Instead of artificially separating technology by putting markup and logic in separate files, React separates concerns into loosely coupled units called “components” that contain both. React does not require the use of JSX, but most people find it visually more helpful when working with UI-related things inside JavaScript code. It also allows React to display more helpful error and warning messages.

Stream Chat API
What is API?
API (Application Programming Interface) refers to an interface created to control functions provided by an operating system or programming language so that it can be used in an application program.

What does API do?
First, API acts as gateways to servers and databases. Valuable information is stored in the database. Not everyone should have access to this database. To prevent this, the API acts as a gateway to our servers and databases, granting access only to allowed people. Second, API allows applications and devices to communicate seamlessly. Here, the application refers to a smartphone application or program that we commonly know. API is responsible for helping applications and devices send and receive data seamlessly. Third, API standardizes all connections. The API standardizes all access, so anyone can get the same access regardless of machine/operating system, etc. Simply put, APIs can be thought of as acting like universal plugs.

Stream Chat

Stream Chat is a tool that provides chat API and SDKs for custom messaging applications, thus enabling the user to build real-time chat quickly. Using Stream Chat's react components, SDKs, and UI kits, users can create a powerful web-chat experience or even a mobile application.

Advantages of using Stream Chat
Building on top of 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 library supports:

1. Rich media messages
2. Reactions
3. Threads and quoted replies
4. Text input commands (ex: Giphy and @mentions)
5. Image and file uploads
6. Video playback
7. Read state and typing indicators
8. Channel and message lists

Since these various functions have already been built for developers, and developers can import and use these component libraries, using Stream Chat, developers can easily develop high-quality chat applications.

Front end design
App.jsx is the main execution, and various app components are built in the form of jsx files in the components folder, and these components are used by App.jsx. And the App.css file is the file that determines the style and layout of the user interface, and the App.css file of Stream Chat where the design of the user interface has already been completed was used. The Assets folder contains any file that lives alongside the source code of the app that the app needs at runtime. In the components folder, each component is built as a jsx file, and the list is as follows:

1. Auth.jsx
2. ChannelContainer.jsx
3. ChannelInner.jsx
4. ChannelListContainer.jsx
5. ChannelSearch.jsx
6. CreateChannel.jsx
7. EditChannel.jsx
8. ResultsDropdown.jsx
9. TeamChannelList.jst
10. TeamChannelPreview.jsx
11. UserList.jsx

Auth.jsx

The Auth.jsx file is a component responsible for authentication of Instant Messaging App, and codes related to authentication of all users, that is, account creation and login are implemented here.

ChannelContainer.jsx
The ChannelContainer.jsx file is the largest component that covers the entire channel part of the user interface and is the outermost component including ChannelInner component.

ChannelInner.jsx
The ChannelInner.jsx file is a component belonging to the ChannelContainer.jsx component, and it is a component that contains and uses the main components of the channel including the MessageList, MessageInput, Thread, Window, ChannelInfo, and CloseCreateChannel.

ChannelListContainer.jsx
The ChannelListContainer.jsx component is the largest component that covers the entire left sidebar with the logout button, and is the outermost component including ChannelSearch, TeamChannelList, and TeamChannelPreview components.

ChannelSearch.jsx
The ChannelSearch.jsx file is a component that includes and uses the ResultsDropdown component, and is the component responsible for the search bar on the left sidebar. Users can enter the name of the channel that the user wants to move into the channel search box, and the channel will be displayed through the ResultsDropdown component.

CreateChannel.jsx
CreateChannel.jsx is a component that includes UserList and CloseCreateChannel components, and is responsible for creating a channel when the Create Channel button on the left sidebar is pressed. If the user clicks Create Channel, the channel name input box and the user list are displayed so that the user can set the channel name and select users to invite to the channel. If the user clicks Create Direct Message, only the user list is displayed.

EditChannel.jsx
EditChannel.jsx component also includes UserList and CloseCreateChannel components and provides different outputs according to Channel Type. If Channel Type is Team, the channel name input box, user list, save changes button, and delete channel button are displayed. If Channel Type is Message, only the delete channel button is displayed.

ResultsDropdown.jsx
The ResultsDropdown.jsx component is used by the ChannelSearch component, and when a user searches through the channel search input box, it is in charge of displaying the search results by dividing it into two sections: channel and direct message.

TeamChannelList.jsx
TeamChannelList.jsx is a component included in ChannelListContainer component. It divides the channel list into two sections, the upper section is Channel and the lower section is Direct Message, and the Create Channel button is placed next to each.

TeamChannelPreview.jsx

TeamChannelPreview.jsx is also a component included in ChannelListContainer, and for a channel whose Channel Type is Team, a preview is provided in the Channel section in the form of "# (channel name)", and for a channel whose Channel Type is Message, a preview is provided in the form of "(Avatar) (full name)" in the Direct Message section. And when the user clicks the preview, the corresponding channel is displayed through ChannelContainer.

UserList.jsx
UserList.jsx is a component used in several components and handles the function of displaying all available users of Instant Messaging App as a list.

Back end design
For the average Stream integration, the development work focuses on code that executes in the client. The React connects to the chat API directly from the client. However, some tasks must be executed from the server for safety.

Data hosting environment
This Stream Chat API-based instant messaging app uses Stream's own database. 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.

Data design
The data stored in the database is largely divided into Channels and Users. Channel-related data will be stored in Channels, and user-related data will be stored in Users.

Channels
Channels data object has two data objects, Members and Messages, and there are 14 Channels sub datas. Channels sub datas are id, type, cid, last_message_at, created_at, updated_at, created_by, frozen, disabled, member_count, config, own_capabilities, hidden, and name. Members data object consists of id, role, created_at, updated_at, last_active, banned, online, name, image, fullName, phoneNumber, and hashedPassword. Messages data object consists of 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
Users data object consists of id, role, created_at, updated_at, last_active, banned, online, language, shadow_banned, email, last_name, first_name, staff_user, and dashboard_user.

Conclusion
Due to lack of basic knowledge related to JavaScript, Github, VScode, database, front-end, back-end, and software development, I encountered difficulties in carrying out the project. However, based on various open sources, YouTube tutorial videos, and class lectures, I was able to proceed with the project while accumulating the basic knowledge I lacked, and I was able to commit and push the final result to Github. It was a very difficult project, but it seems a lot clearer that this project brought me one step closer to being a software developer.