



CS411 Software Architecture Design

Project 2 Report

Abdul Razak Daher Khatib – 21801340

Efe Şaman – 21903248

Çağla Ataoğlu – 21902820

Alperen CAN - 21601740

Table of Contents

Table of Contents	2
Problem Definition and Domain	4
Technical Problems	4
Security and privacy	4
Problem	4
Solution	4
Storage	5
Problem	5
Solution	5
Performance	5
Problem	5
Solution	5
Authentication	5
Problem	5
Solution	6
Scalability	6
Problem	6
Solution	6
Receiving messages in order	6
Problem	6
Solution	6
Stakeholders' Contributions and Expectations	7
Owners	7
Maintainer and Assessors	7
System Administrators	7
Communicators	7
Developers	7
Candidate Developers	7
Production Engineers and Testers	8
Users	8
Funders	8
Pros and Cons	8
Meta Whatsapp	8
Telegram	8
Viber	9
Signal	9
Requirements	9
Real-time text transmission	9
Emojis	10
File transfer	10
Voice over IP	11
Video chat	12

High Level Architecture	12
Views	12
Module styles - Layered Style	12
Module styles - Decomposition Style	13
Authentication & Authorization Module	13
Transmitter Module	13
Encryption Module	14
Compressor Module	14
VOIP Module	14
Assembler Module	14
Module styles - Generalization style	14
Encryption Module	14
Component and connector styles - Call-return style	14
C&C styles - Client-Server	15
C&C styles - Publish-Subscribe	15
C&C styles - Pipe-and-filter	15
Allocation styles - Deployment Style	16
Rationale	16
Layered architecture	16
Client-server architecture	16
Decomposition Style	17
Generalization style	17
Client-server style	17
Subscribe-publish style	17
Microservices and Horizontal scaling	17
Pipe-and-filter	17
Patterns	18
Microservices	18
Horizontal Scaling	18
Three-tier architecture	18
Model-View-Controller	18
Implementation	19
Name and logo	19
Features	19
Register and login	19
Create new chat	20
View chats	20
Send and receive messages	20
Comparison with Products	23
References	24

Problem Definition and Domain

It's no secret that instant messaging apps are one of the essential forms of communication in today's world. By 2021, it was estimated that 3.09 billion people used mobile messaging apps worldwide [1]. Some of the most popular messaging apps are Whatsapp, Telegram, Viber and Signal. None of these are perfect, but since messaging has become a non-negotiable part of most technology users' daily life, people simply pick whichever one best suits their needs (or in many cases, whichever one their friends and family use) and go with it. The aforementioned 4 messaging apps were chosen for analysis in this report.

Technical Problems

As mentioned in the previous section, Whatsapp, Telegram, Viber and Signal are very popular instant messaging (IM) apps, and none of them are without problems.

Security and privacy

Problem

Security and privacy are major problems with messaging apps. They're technical problems that a lot of users care deeply about, as seen in the case with Whatsapp [2] where many users switched to other platforms because they found out their data was being collected. In a secure IM app, no one but the sender and receiver should be able to read the messages.

Solution

End-to-end encryption is a popular solution to improve security. End-to-end encryption ensures that messages are in an unreadable format during transmission [3]. Another solution is disappearing or self-destructing messages. These messages are automatically deleted once the receiver reads the message. This way, private messages are not stored permanently. Signal has an interesting security feature called safety numbers, where users can check if their conversation is secure using a number unique to a chat [4].

Storage

Problem

Storage is another common issue, both on the server side and the user's side. It can be a problem for users when their messages are stored locally and take up too much space on their devices. In contrast, when all messages are stored on the servers, these servers need to be highly scalable to adapt to a large number of chats. The problem is that the developers need to decide which method fits their platform's needs.

Solution

Different platforms have different solutions to this. For example, Whatsapp chats are kept on the devices, which takes up a lot of storage space. Users can back up their chats to the cloud if they wish [5]. In contrast, Telegram stores all chats on their servers, and the chats stay there

until they are deleted by the users, except for the secret messages, which are deleted immediately after viewing. This is more efficient for the users' device storage. On the downside, this has raised some security concerns [6].

Performance

Problem

When developing an instant messaging app, performance is critical. An user should immediately receive a message after it's sent to them. The servers should be strong enough to handle performance draining operations such as group chats with a lot of participants, large files or media, photos sent in bulk and such.

Solution

The most common way to ensure that intensive operations do not slow the app down is setting limits. For example, Whatsapp has a limit of 1024 participants per group whereas Telegram supports a whopping 200000 participants per group. This can mean that Whatsapp groups usually have better performance than Telegram groups because the participant limit is much stricter. Similarly, Viber has a limit of 40 participants per video call to prevent performance issues in overcrowded video calls. Additionally, images and videos can be compressed before being sent, while maintaining the highest quality possible, in order to reduce the load on servers.

Authentication

Problem

A major threat to any user for IM applications is authentication. Insecure authentication protocols can lead to bad actors having access to the users messages and data receiving them instantly exactly like the user. The risk lies in the fact that a user performs a single login on his device only, so no more than a single successful attempt to authenticate is needed for the user's data to be exposed. Therefore, since the data is personal and sensitive, any attempt to login from another device should be scrutinized.

Solution

Instant messaging applications must maintain only a single account per user, meaning that each number can have a maximum of one account. This account is opened in one device only, or opened in a browser after using the first to authenticate the user. Similar to WhatsApp web. First the phone number carrier should be notified and asked to approve the authentication attempt, second in case the user approves all old devices should lose access immediately to the account, and then the new device shall have access and get fully authenticated. This prevents any access from old devices and assures the user that no one else has access to their data as long as they did not lose access to their account.

Scalability

Problem

Similar to the performance problem, users should be able to receive and send messages instantly. For performance issues, we can set some limiting factors. However, some elements cannot be limited, like the number of users currently active or media files sent concurrently through the system as a whole.

Solution

Multiple decisions can be made to account for such issues. For example, by modulating the system into microservices we can observe what functionalities are receiving the highest load and with horizontal scaling we can allocate more nodes for said overwhelmed microservices.

Receiving messages in order

Problem

Instant Messaging applications are used by users to have conversations. Naturally, conversations out of order make little to no sense. Therefore it is crucial to receive messages in order. While this problem looks simple at first glance, in practice it is difficult to ensure order of reception. The problem arises when the transmitting server is changed, either because of change in the server assigned due to a new server being assigned due to heavy load on the current server, or if the server restarts unexpectedly. Once this happens the order can be lost.

Solution

In order to solve this problem, we propose a solution similar to Google Cloud's [7]. The solution is as follows: keep an OrderingKey related to the messages, this key is transmitted with the conversation every time a new server is assigned. OrderingKey keeps track of the messages and their order [8]. OrderingKey can be perceived as an ID for the queue of messages, so whenever a connection is re-established the OrderingKey can be used to find that queue and start appending to it.

Stakeholders' Contributions and Expectations

Owners

The main stakeholders are the owners. They recognize a market opportunity for a service and seize hold of it. Regardless of the company's size, the owner has ultimate authority over it and thus chooses whether or not to assign some crucial executive responsibilities to qualified individuals.

Maintainer and Assessors

The maintainers and assessors are responsible for merging pull requests and closing issues on Github for open-source projects such as Signal and Telegram. They control what is approved to merge into the project, and have a significant role in the development of it.

System Administrators

System administrators support and maintain computer servers and networks. Their responsibility is to make sure that computer systems meet the needs of the organization. Since these platforms have millions of users communicating across worldwide, system administrators are one of the most important stakeholders for these companies.

Communicators

Communicators are the people who work in customer support. Their responsibility is answering the questions and tickets of the users to have a role in improving the existing system based on the users' needs and expectations. Communicators may include volunteers as in Telegram.

Developers

Developers are the stakeholders with the most contribution. They are responsible for fixing the bugs after users report them, restructuring the project if necessary in order to keep it up with new technology, and adding new features to the system. Writing maintainable and readable codes are also a big responsibility of the developers.

Candidate Developers

Candidate developers are the submitters of pull requests having an interest in the development of the open-source projects. Although sometimes their work is not completely merged, they still have an influence on the development of the projects, such as in the Telegram-Web [9]. They expect a well organized architecture with a good readability so that changes can be made easily.

Production Engineers and Testers

Product engineer focuses on the overall product in terms of the innovation, design, test and deployment phases, and test engineers inspect and report on the quality of products through the entire production cycle. They are important stakeholders taking part in the entire development process.

Users

Every user of Telegram is considered as a stakeholder. They are the individuals who use the applications for messaging with others. Their expectations include making video calls, sending voice messages, using graphical smileys and gifs, and transferring files. They also expect a user-friendly interface. Security and privacy are also indispensable for users. In addition, users may include small business accounts as Meta Whatsapp started in 2018. Their expectations include interacting with customers easily via tools enabling them to respond to messages quickly.

Funders

Funders are the individuals or organizations providing financial backing for the projects. They are one of the important stakeholders for some platforms such as Signal, which is funded nearly \$50 million.

Pros and Cons

Meta Whatsapp

Pros

Whatsapp is the most widely used messaging app [10]. It has lots of useful features such as replying to messages, stories, web version, starred messages, broadcasting etc. Chats are end-to-end encrypted for security. It's part of the Facebook ecosystem and it has a business version available.

Cons

It was revealed that Whatsapp collects personal data for advertisements [11], which damaged user trust. Chats are stored on the device so the app consumes a lot of storage. Whatsapp cannot be used without a phone number. Also, the size limit for sending files is relatively low (16MB) [12].

Telegram

Pros

Telegram is a widely used app, especially in some parts of the world. It has a good security system and a secret chats feature with higher security. Messages don't take up space on the device, they're stored in the cloud. The channels feature allows broadcasting, reaching information fast, polls, and livestreams [13].

Cons

Telegram has no indicator of online/offline status of a user. Chats (except secret chats) are not encrypted. Moreover, Telegram doesn't have official technical support, it's done by volunteers.

Viber

Pros

Viber offers high quality video calls and allows users to call internationally for low prices. The ability to create stickers and gifs enhances chats. Viber uses end-to-end encryption.

Cons

Viber doesn't have as many users as other examples. It contains advertisements in the app and the interface has an outdated look.

Signal

Pros

Signal is open source and fully free [14]. It has higher security and encryption than other examples. It has a disappearing messages feature for private chats. It offers other security features such as blocking screenshots and blurring faces in photos.

Cons

Signal has relatively slow performance. It has a smaller number of users compared to other IM apps.

Requirements

Real-time text transmission

Push technology is an internet communication system in which the transaction request is generated by the publisher [15]. Opposite the pull technology, information transmission requests are not made by the clients. Instant messaging, synchronous conferencing and e-mails are the common examples working with push technology. It requires using standard APIs and services integrated with mobile and desktop platforms. For a web based push technology, each browser manages push notifications through its own system called push service. Since security is also a requirement, subscribing users to the push service of the browser is necessary to encrypt the messages when they grant permission for push on the site [16]. Subscription also includes a unique endpoint URL for the push service. Next, push messages should be sent to this URL from the server, so that push server can forward the message to the client's device or browser. Ultimately, the push event has to be picked up by the service worker which gets the data from the message, then a notification is displayed.

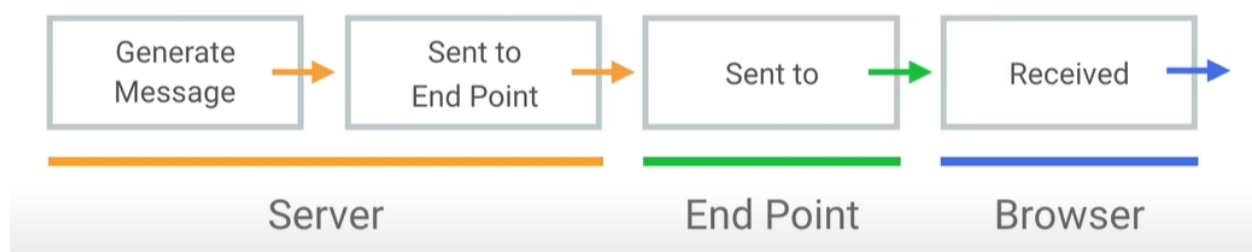


Figure 1: Messaging with Push Technology [16]

The quality attributes for real-time text transmission are below:

- **Performance:**
Messages should be delivered instantly.
- **Reliability:**

Data should be delivered without any damage to the client, even if the receiver's browser is closed.

- **Data Privacy:**

If some data is going to be stored, users have to be informed about it, and they have to be able to choose which information to share.

- **Security:**

Data should be protected against unwanted access.

Emojis

Emojis are the icons to visually represent an emotion or a symbol. Although they have been implemented by using different types of encoding techniques since the 1990s, the Unicode Standard has recently become widespread and dominant. It is an information technology standard for consistent encoding and representation of the symbols, historical scripts and more. Emojis are the characters mostly encoded in UTF format [17]. Therefore, in order to display almost all characters and symbols in the world, it is required for the environment to know which character set is being used, such as specifying UTF-8 or UTF-16 with the necessary syntax based on the language.

The quality attributes for graphical smileys are as follows:

- **Compatibility:**

Emojis should be visible to all users regardless of the operating systems of their devices.

File transfer

File transfer is the process of copying or moving a file from one computer to another over internet connection [18]. Data may include various extensions, such as documents, multimedia graphics and more. In order to set up a file transfer, a communication protocol is necessary to govern the transfer - that is - to define a set of rules determining how the information will be transmitted. The most common ones include File Transfer Protocol (FTP), Transmission Control Protocol (TCP) and HTTP. In case the files contain sensitive information, security is a requirement. Therefore, using encrypted file transfer protocols like FTSP has also become widespread [19]. Moreover, the system also has to be able to support large file transfers, and ensure that data loss is prevented.

The quality attributes for file transfer are below:

- **Reliability:**

The transferred data should not be damaged, lost, nor duplicated.

- **High-Performance:**

Data should be transferred with high speed. In addition, the system has to support concurrent transfer sessions.

- **Security:**

Possible interception to the files by third-parties should be prevented.

- **Compatibility:**

File transfer should support a variety of file formats.

Voice over IP

Voice over Internet Protocol is a digital telephone system enabling users to make or receive phone calls over the Internet, rather than a traditional telephone network. VoIP communication is provided by transmitting voice data packets over the Internet. The voice audio from a phone conversation is broken up into digital data packets. Following, the data packet is sent to the recipient via VoIP technology. In order to travel over the IP network, these packets are compressed, and once they reach the destination they are reassembled. Therefore, a high-speed broadband internet connection is necessary for VoIP. In general, it requires a minimum 90-156 kbps bandwidth for functioning properly [20]. For example, Skype suggests 0.1 Mbps for voice calls [21]. In addition, Voice over IP has been implemented with a variety of communication protocols, such as RTP (Real-time Transport Protocol). Currently, almost all smartphones, computers and tablets are compatible with VoIP, but if a standard desk phone is going to be used, it should be able to connect to an Analog Telephone Adapter.

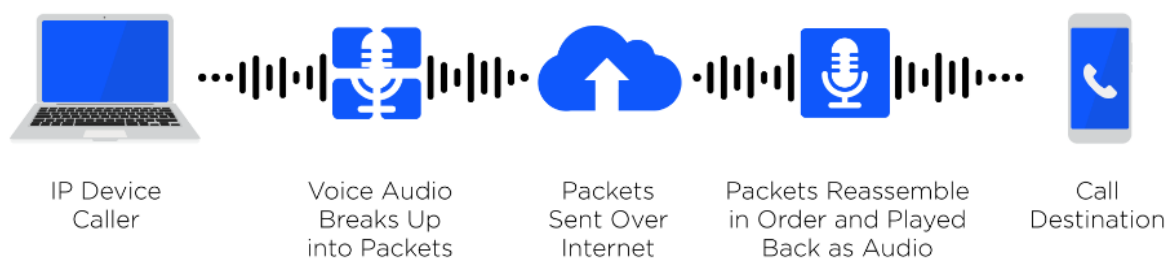


Figure 2: Communication via VoIP [20]

Voice Over IP has the following quality attributes:

- **Performance:**
High quality audio is necessary as well as synchronization.
- **Security:**
The call has to be encrypted and should not be intercepted by third-parties.

Video chat

Video conferencing is the two-way or multipoint reception and transmission of both audio and video signals by people in different locations for simultaneous communication [22]. As in VoIP, the data should be compressed in order to be transferred, by a hardware or software system called codec. It requires the hardware for video input (camera) and video output (monitor), as well as audio input and output devices. In terms of software, both TCP and UDP are being used for video chat [23]. Although UDP is popular for video chats due to providing a high speed of sending data packets, TCP offers a more reliable service with crystal clear HD video, which is often considered as more important than the speed. The bandwidth requirement for 720p

resolution during the video chat is 1 Mbps, whereas 4 Mbps is needed to have 4K resolution[23]. Robust and fast internet connection is also another requirement since download speed determines how well a video connection is received from the participants and upload speed determines how well the video streams will be sent to the others. In addition, internet connection should be stable as latency affects how well the connections synchronize with each other. Therefore, lower latency is required.

The quality attributes for video chat are as follows:

- **Performance:**
High quality video and audio is essential as well as synchronization.
- **Robustness:**
System should be able to serve multiple participants at the same time for group conferences.
- **Security:**
The video call has to be encrypted and should not be intercepted by third-parties.

High Level Architecture

Views

Module styles - Layered Style

One module style that can be applied to our system is layered architecture style. The three layers include:

- The client layer, which is the layer that users interact with. The client sends requests to the server layer and performs API calls.
- The server layer, which responds to requests coming from the server layer. Modules in this layer are responsible for performing business logic and communicating with the data layer.
- The database layer, which is responsible for storage of data. Our system will use Firebase Realtime Database for this purpose.

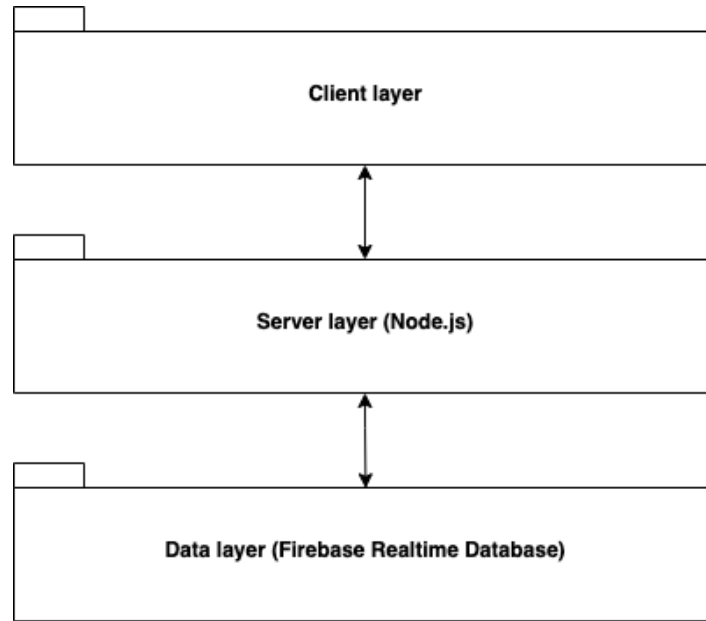


Figure 3: Layered architecture

Module styles - Decomposition Style

In order to analyze and design the IM system in a modifiable and extendable method, we used Module styles and decomposed our system into different parts based on their responsibility, <<is responsible for>>. Basing our analysis on the requirements and the technical problems we observed a number of Modules that our system consists of. The following sections explore and discuss these Modules.

Authentication & Authorization Module

One of the problems was authentication and authorization. In order to ensure that the problem is resolved efficiently and to allow modifications for this process in case of any exploits, we separated this functionality as one key module. This module will handle the process of any account access attempt and will authenticate the device regularly to keep access token update-to-date.

Transmitter Module

Another key requirement is the ability to transmit messages between users, this module will handle all the processes required for all forms of transmission, after the necessary processing is done. This includes the transmission of text messages, media, and files. For example, we will create a thumbnail and send it to be a placeholder while the media file is still being transmitted. While a text will be sent directly. The difference in the transmission here requires different implementation for each type of file despite the fact that the main process is common to all of these files.

Encryption Module

Encryption is an expected standard by users and no IM application can skip implementing such a module. In order to keep all of the data safe and secure even if the system's security was

undermined and in order to protect the data while transmitted, the encryption module will encrypt all data that is sent or received (and consequently stored).

Compressor Module

In order to ensure that the servers will not be overwhelmed by data exchanged by users, all large files and VoIP will be compressed into specific standards, that maintain an acceptable quality but lowers the size, to make the load light on both the servers, application, and networks.

VOIP Module

In this module the process of VoIP will be handled. Our implementation for transmitting messages and VoIP is different for reasons mentioned above. The module will handle the data from the beginning to the end of a VoIP request, this includes compressing, encryption, breaking into packets, receiving, ordering, decrypting and then playing.

Assembler Module

All large data transmitted, like long texts, media, other files, and VoIP, need to be broken down into packets before sent and then to be reassembled in correct order before it can be accessed by the receiver. We use this module to handle this process.

Module styles - Generalization style

The system will handle various types of files, and will transmit data in different ways based on the type of communication. Users can send messages, pictures, audios, videos, emojis, and other files. These are all transmitted in similar ways, in other words they follow the same protocol of compressing, encrypting, breaking down, ordering, etc. However, since the representation of these files is different on the low-level the implementation can differ in details. We use Generalization style to capture the similarities in the process and define a common protocol for each of our main processes that can have different implementations based on the data type. The following sections present these realized generalizations.

Encryption Module

Text, images, videos, and VoIP all use encryption. However, images are represented as 2D arrays and are segmented into different pieces, VoIP uses audio, a simple 1D array, or two arrays in case it is stereo, and videos are a number of images and an audio file. This necessarily means that the encryption process will be implemented differently.

Component and connector styles - Call-return style

Our system uses call-return style architecture, more specifically client-server architecture. In this style, the client communicates with the server via several methods, such as sockets and HTTP calls in our case. We will use the Axios.js framework to facilitate HTTP calls between clients and the server.

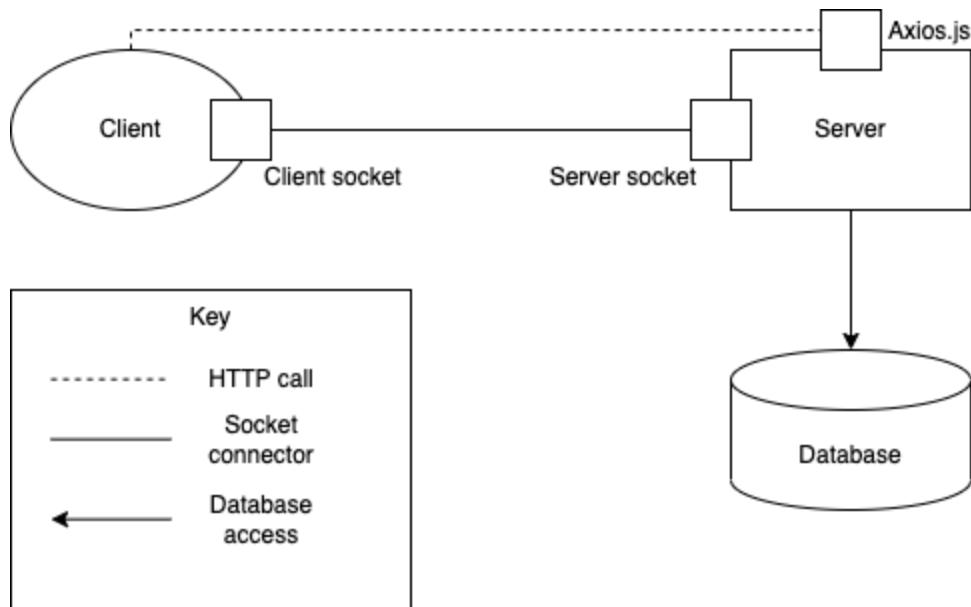


Figure 4: Call-return style

C&C styles - Client-Server

Above we can see the communication between the clients and the server. The clients communicate with the server through either sockets or http requests, to request various tasks. It allows separating work between client-side and server-side, and increases security by handling sensitive work on the server side.

C&C styles - Publish-Subscribe

To allow messages to be exchanged in order and to account for one side, or both, being offline after a message is sent, it is important to implement a publish-subscribe architecture, this will allow for unreceived messages to be kept in the server till the receiver(s) all get a hold of a copy.

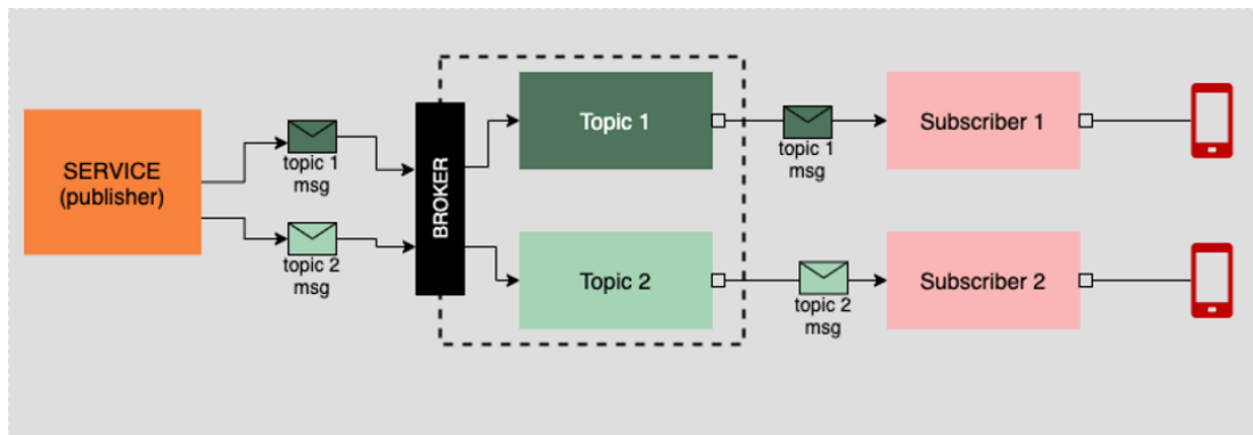


Figure 5: Classic Publish-Subscribe Architecture [24]

C&C styles - Pipe-and-filter

Pipe and Filter is a Data Flow style that is used when data gets transferred and/or transformed. We will use it in this system to deal with media files. Media files can be large in size and slow to transfer. In order to ensure low load on the servers, security and order of messages the system we perform the following processing on each media file: 1) Verify that the file does not exceed

the size limit. 2) Generate a thumbnail and send it as a message, this thumbnail will act as a place-holder for the media file while it is still being transmitted, this way we will ensure the order of messages. 3) Compress the file if needed. 4) Encrypt the file. Afterwards, the file is ready to be sent. Similarly, VOIP can use this style. The processing would be as follows: 1) Compress the audio. 2) Break into packets. 3) Encrypt. And then the receiver side would perform the following: 1) decrypt. 2) Reassemble. 3) Play.

Allocation styles - Deployment Style

We used deployment style to show some details that were not included in the component and connector view. For example, one server is used by multiple clients. These clients can communicate with the server via sockets or the internet, namely HTTP calls. The server is developed in the Node.js environment. It connects to the database via the internet as the database is hosted in the cloud by Firebase.

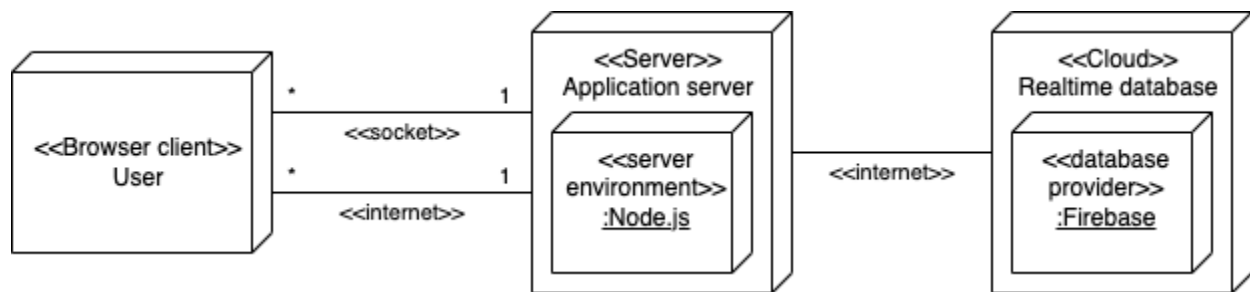


Figure 6: Deployment style

Rationale

Layered architecture

Our rationale for using layered architecture is that it promotes security. Security and privacy are issues that users deeply care about, and layered architecture ensures a more secure application by decoupling the client layer from the data layer. This way, the client is not allowed to communicate with the database, only the server can do that. Another reason to use layered architecture is maintainability. Each layer has a purpose and whenever an error occurs, it's easier to understand where it comes from because the individual steps of data flow are clear. Another advantage of layered architecture is the separation of concerns (SoC). Since different layers are separated the dependencies are separated, minimized and easily managed. This allows for better maintainability and easier testing.

Client-server architecture

Security is a priority for many instant messaging platforms, which is why we chose to use client-server style architecture. Authentication and authorization can easily be implemented into client-server networks. Moreover, client-server is a reliable style. If one of the clients crashes, it does not affect the server and thus other clients using the server don't experience problems. Additionally, In order to enable instant transmission of data through VoIP a client-server is

crucial since the communication is peer-to-peer in nature and low latency is a key feature in it, we enable each device to act as a server and client in VoIP

Decomposition Style

One of the main concerns of any software system is the maintainability, modifiability, and the extensibility especially with the constant changes in technologies and style. Decomposition style allows for separation of functionality, making the expected changes easy to make and manage. Additionally, the implementation of the style on a hardware level allows for enhanced security since one microservice failing would not compromise the whole system. Finally, by separating the modules we allow for less dependencies and for easier testing.

Generalization style

Since IM deals with sending and receiving messages instantly in various forms, it is natural that the overall process of all transmissions is the same. However, implementations for each type of transmitted data are different. That is why we captured the shared protocols of each functionality we support and then based on these protocols we standardized the processes for these functionalities, then implemented each part according to the nuances that they require. These generalizations make the code easy to read, extend and enable reuse.

Subscribe-publish style

While it is important to send and receive messages instantly, IM applications have to account for more than merely this. If the users are in groups or if the receiver is not online the messages need to be stored until the receivers receive it, otherwise any network failure could cause a loss of the messages making the system unusable in a large number of situations. This is why subscribe-publish was the main choice for all forms of exchanged data; the application shall keep a copy while it is not received by all parties.

Microservices and Horizontal scaling

To enable modularity and scalability and security of the system, microservices were our immediate choice. Microservices will enable separation of functionalities, meaning if any failure occurs in one functionality the system shall not go down. Additionally, along with horizontal scaling, if a microservice is overwhelmed, an underwhelmed node can be quickly repurposed to account for the high load. Moreover, since change is a constant factor, especially in security, the ability to modify small pieces of the system without the need to take down the whole system is of great importance. Horizontal scaling can also support microservices by being used to deploy only the microservices that have high load, reducing the costs of scaling. In short, both of these styles allow for flexibility and scalability.

Pipe-and-filter

Since each message has to go through processing before being sent, pipe-and-filter was used to implement this on the servers.

Patterns

Microservices

In order to comply with the modularity in our design and allow for better scalability, modifiability, and extendability. We will use microservices. Each process, like compression, encryption, breaking down, etc. Will have a dedicated microservices. These microservice either are sequential or paralleled. If one process requires the other to be done, these two processes will be sequential, if they do not require each other, then they are parallel in the microservices architecture.

Horizontal Scaling

In order to allow for high availability the system will use horizontal scaling; by adding more nodes to the system we do not just increase the capabilities of the servers to serve users, but we also allow for higher availability. The reason we chose horizontal scaling as the main scaling strategy instead of vertical scaling is because the needed computing power is not as high as the needed extremely high availability. The existence of parallel nodes means more requests can be received and served concurrently.

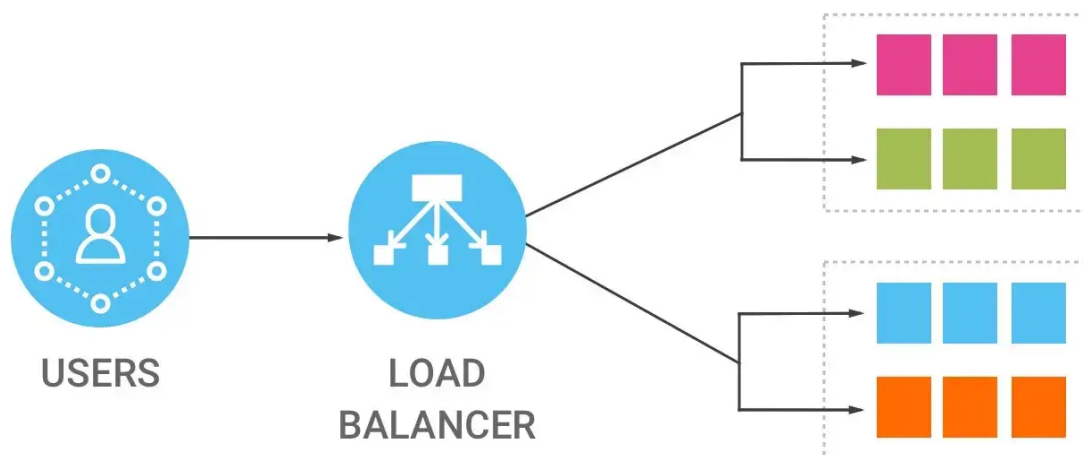


Figure 7: Horizontal Scaling Using Load Balancers [25]

Three-tier architecture

We divided our system into three separate layers, web (also known as presentation) layer, running on the browser of the users. Application layer, which is represented by the server processing the request and managing the data, its transformation and transmission. And finally, the database layer where the data is saved, temporarily and retrieved until all receivers have a copy of the message. In order for this pattern to be implemented properly we implemented an MVC pattern.

Model-View-Controller

In order to accomplish proper separation of responsibilities and modularity on the code level, we separated the system into models, the data containers and processors done in Java, views, the front-end done in JavaScript, and finally, Controller, the bridge between these two elements. We implemented a standard MVC.

Implementation

Name and logo

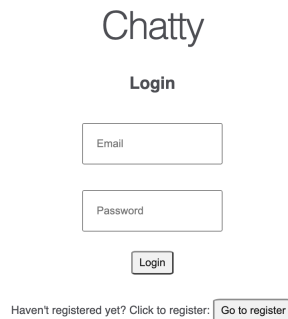


Our application's name is Chatty. The logo is made through an online free logo generator [26]

Features

Register and login

Users can register to Chatty using their email and a password. The email address is automatically checked to ensure it's a valid email, in a format such as 'example@domain.xyz'. Registered users can login, again with their email and password. Users are free to log out from any page. Unlike many IM apps, Chatty can be used without a phone number.

A screenshot of the Chatty login page. At the top, the word 'Chatty' is displayed in a large, grey, sans-serif font. Below it, the word 'Login' is centered in a smaller, bold, black font. There are two input fields: the first is labeled 'Email' and the second is labeled 'Password'. Below these fields is a 'Login' button. At the bottom, there is a link that says 'Haven't registered yet? Click to register:' followed by a 'Go to register' button.

Chatty

Login

Email

Password

Login

Haven't registered yet? Click to register: Go to register

Figure 8. Login page

Chatty

Register

Email

Password

Confirm Password

Register

Already registered? Click to login: [Go to login](#)

Figure 9. Register page

Create new chat

Users can start chatting with another user simply by entering their email and clicking the “Start a new conversation!” button.

Your conversations

john.smith@gmail.com

Start new conversation!

Figure 10. Creating a new chat

View chats

Users can view conversations they’ve created on the main page of the app. From there, they can click a chat to go to its page.

Your conversations

[Continue your conversation with: john.smith@gmail.com](#)

Enter email of user

Start new conversation!

Figure 11. Chats page

Send and receive messages

While a user is on the chat page, they can send messages by typing the message in the box and clicking “send”. They can send and receive messages from other users in real-time, and messages won’t disappear when they exit the app and come back again. All the messages are

kept on the database so they can be accessed from any device. While a user is in the chat page, push technology is active and sockets are used to deliver messages instantly.

Chatty

Chat with john.smith@gmail.com

Log out

janedoe@bilkent.edu.tr: hi!

janedoe@bilkent.edu.tr: how are you?

john.smith@gmail.com: fine, and you?

I'm fine as well :)

Send

Figure 12. Chat page

Our application loads the chat history from the database, however receives new messages through the client socket. The messages are pushed from the server and directly displayed in the client. Below, there are some code snippets to explain how we have implemented such a behavior. The code snippets are heavily shortened to only give the outline of the processes and exclude unnecessary details.

The following code snippet fetches the chat history and displays it. In the code snippet, we fetch the chat from the database, connect our socket to the chat room by emitting `"chat-connected"` to the server socket.

```
const userDocRef = doc(db, "users", user.email)
const userDocSnap = await getDoc(userDocRef)
// Shortened code for the report: get userObj

var chat = userObj.chats[chatInd]
const chatDocRef = doc(db, "chats", chat)
const chatDocSnap = await getDoc(chatDocRef)
// Shortened code for the report: get chatObj

socket.emit("chat-connected", chatObj.id)

async function viewMessages() {
  for (var msg in chatObj.messages) {
```

```

        // Display the msg
    }
}

await viewMessages()

```

The following code block is how the client socket receives messages from the server (which is given as an example in the next code block). The sockets allow us to utilize push technology as we push the messages from the server socket to the client socket.

```

socket.on("chat-message", (messageObj) => {
    var msgText = document.createElement("p")
    msgText.innerText = messageObj.sender + ": " + messageObj.content
    divider.appendChild(msgText)
})

```

The following code block is how we get a message from one client and push it to the proper client (chat room). We also add the message to the database from the user. We first push the message through the sockets, which is very fast; and then add the message to the database to store the message for the future. This improves the performance of the application.

```

socket.on("send-chat-message", (user, chat, msg) => {
    var messageObj = {
        sender: user,
        content: msg,
        timestamp: Date.now()
    }
    io.to(chat).emit("chat-message", messageObj)
    chats.doc(chat).update({
        messages: admin.firestore.FieldValue.arrayUnion(messageObj)
    })
})

```

As mentioned above, the server has two main communication channels, one being the sockets and the other http endpoints (api). The previous code snippet examples were examples of the sockets and how the push technology was implemented. Other processes like creating a new chat thread, or creating a new user are done through http calls.

```

// Initialize firebase database and authentication
admin.initializeApp({
    credential: admin.credential.cert(credentials)
})

const db = admin.firestore()
// Create the server with express.js

```

```

var server = http.createServer(app)
server.listen(3000)
// Import socket.io for sockets
const io = require('socket.io')(server)

// API endpoints
app.get('/', function(req, res) {})
app.get('/main', function(req, res) {})
app.get('/:chat', function(req, res) {})
app.post('/api/users/create', async (req, res) => {})
app.get('/api/user', async (req, res) => {})
app.post('/api/chats/create', async (req, res) => {})

// Server socket
io.on("connection", socket => {
  socket.on("chat-connected", (chat) => {})
  socket.on("send-chat-message", (user, chat, msg) => {})
  socket.on("leave-chat", (chat) => {})
})

```

Comparison with Products

Chatty is similar in a variety of ways to these applications. For example, scalability, authentication, and security were taken into consideration. Additionally, it performs the basic functionalities that all of these applications perform like authenticating users, transmitting messages, etc.

On the other hand, Chatty is quite different from example chat apps that were discussed. First of all, it's really simple. Other chat apps contain a lot of features which can sometimes make them hard to use, or slow the app down. Chatty is quick and user-friendly. One major difference is that Chatty doesn't require a phone number to use, only an email address is sufficient. The email address is also the username, which increases security because it's easier to detect hackers and scammers because they're not able to imitate others. This makes Chatty resilient to phishing and similar security concerns. Another security feature is that Chatty's authentication is handled by Firebase. Since Chatty is a very small application, it makes sense for authentication, authorization and user storage services to be handled by another server. In Chatty, all messages are stored in the cloud, via the Firebase Firestore database. This is different to some apps such as Whatsapp, which consumes a lot of storage on devices. This approach is more similar to Telegram's approach, which also uses cloud-stored chats. Chatty is similar to Signal in its way of development. Chatty is not owned by a big company, it's by a small group of developers.

References

- [1] Published by L. Ceci and 22, S. (2022) *Mobile messaging users worldwide 2025*, Statista. Available at: <https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/> (Accessed: December 6, 2022).
- [2] Al Jazeera (2021) *WhatsApp fined \$266M by EU privacy watchdog over Data Breach*, Privacy News | Al Jazeera. Al Jazeera. Available at: <https://www.aljazeera.com/economy/2021/9/2/whatsapp-fined-266m-by-eu-privacy-watchdog-over-data-breach> (Accessed: December 6, 2022).
- [3] *What is end-to-end encryption?* (no date) IBM. Available at: <https://www.ibm.com/topics/end-to-end-encryption> (Accessed: December 6, 2022).
- [4] *Signal support* (no date). Available at: <https://support.signal.org/hc/en-us> (Accessed: December 6, 2022).
- [5] *How to save your chat history: Whatsapp help center* (no date) *How to save your chat history | WhatsApp Help Center*. Available at: https://faq.whatsapp.com/1180414079177245/?cms_platform=android (Accessed: December 6, 2022).
- [6] Pichsenmeister, D. (2016) *A glimpse into Telegram's security*, Medium. Medium. Available at: <https://medium.com/@pichsenmeister/a-glimpse-into-telegram-s-security-bbf3eaa58aab> (Accessed: December 6, 2022).
- [7] *Ordering messages | cloud pub/sub documentation | google cloud* (no date) Google. Google. Available at: <https://cloud.google.com/pubsub/docs/ordering> (Accessed: December 6, 2022).
- [8] *Publish messages to topics | cloud pub/sub | google cloud* (no date) Google. Google. Available at: <https://cloud.google.com/pubsub/docs/publisher#using-ordering-keys> (Accessed: December 6, 2022).
- [9] Delftswa (no date) *Telegram web*, *Telegram Web · Delft Students on Software Architecture: DESOSA 2017*. Available at: <https://delftswa.gitbooks.io/desosa-2017/content/telegram-web/chapter.html> (Accessed: December 6, 2022).
- [10] Mehner, M. (2022) *WhatsApp, WeChat and Meta Messenger apps - global usage of messaging apps, penetration and statistics*, *MessengerPeople by Sinch*. MessengerPeople by Sinch. Available at: <https://www.messengerpeople.com/global-messenger-usage-statistics/> (Accessed: December 6, 2022).
- [11] Lomas, N. (2016) *WhatsApp to share user data with Facebook for AD targeting - here's how to opt out*, *TechCrunch*. Available at: <https://techcrunch.com/2016/08/25/whatsapp-to-share-user-data-with-facebook-for-ad-targeting-heres-how-to-opt-out/> (Accessed: December 6, 2022).
- [12] *I get a message that my video is too long and it won't send: Whatsapp help center* (no date) *I get a message that my video is too long and it won't send | WhatsApp Help Center*. Available at: https://faq.whatsapp.com/1119635945362627/?locale=en_US (Accessed: December 6, 2022).
- [13] Telegram Channels (2018) *Telegram*. Available at: <https://telegram.org/tour/channels/tr?ln=a> (Accessed: December 6, 2022).
- [14] *Signal* (no date) *GitHub*. Available at: <https://github.com/signalapp> (Accessed: December 6, 2022).
- [15] *What is push technology? - definition from Techopedia* (no date) *Techopedia.com*. Available at: <https://www.techopedia.com/definition/5732/push-technology> (Accessed: December 6, 2022).
- [16] Google Chrome Developers (2019) *Send and receive push messages - progressive web app training*, *YouTube*. Available at: <https://youtu.be/N9zpRvFRmj8> (Accessed: December 6, 2022).
- [17] *The unicode® standard: A technical introduction* (no date) *[Unicode]*. Available at: <https://www.unicode.org/standard/principles.html> (Accessed: December 6, 2022).

- [18] *What is a file transfer and how does it work?* (no date) IBM. Available at: <https://www.ibm.com/topics/file-transfer> (Accessed: December 6, 2022).
- [19] Glass, V. (no date) *Understanding key differences between FTP, FTPS and SFTP*, JSCAPE. Available at: <https://www.jscape.com/blog/understanding-key-differences-between-ftp-ftp-and-sftp> (Accessed: December 6, 2022).
- [20] *What is voice over ip? complete guide to VoIP* (2022) *What is Voice Over IP? Complete Guide to VoIP*. Available at: <https://getvoip.com/library/what-is-voip/> (Accessed: December 6, 2022).
- [21] *Amount of data and bandwidth needed for VoIP: Gobrolly internet* (2022) *GoBrolly Internet - "It Just Works"*. Available at: <https://gobrolly.com/data-bandwidth-voice-internet-protocol-voip/> (Accessed: December 6, 2022).
- [22] *Videotelephony* (no date) *The Free Dictionary*. Farlex. Available at: <https://encyclopedia2.thefreedictionary.com/Videotelephony> (Accessed: December 6, 2022).
- [23] Writer, S. (2021) *Video conferencing setup requirements: Your checklist for hardware and software*, Lifesize. Available at: <https://www.lifesize.com/blog/video-conferencing-requirements/> (Accessed: December 6, 2022).
- [24] Rreselma (2021) *Architectural messaging patterns: An illustrated guide*, Enable Architect. Red Hat, Inc. Available at: <https://www.redhat.com/architect/architectural-messaging-patterns#message-exchange-architectures> (Accessed: December 6, 2022).
- [25] Yang, L. | P. (2022) *System Design - Load Balancing*, Medium. Computer Science Fundamentals. Available at: <https://medium.com/must-know-computer-science/system-design-load-balancing-1c2e7675fc27> (Accessed: December 6, 2022).
- [26] freelogodesign.org