CS411 Project 2 Presentation - Chatty

Abdul Razak Daher Khatib
Efe Şaman
Çağla Ataoğlu
Alperen Can

Problem Definition and Domain

- Messaging apps are used by 3.09 billion worldwide
- Users pick ones that fit or needs or ones that their friends and family use

Technical Problems and Solutions

Security and Privacy

Problem

- Case of Whatsapp: many users switched because of data collection
- Only sender & receiver should be able to read messages

Solution

- End-to-end encryption
- Disappearing messages
- Safety numbers (used by Signal)

Storage

Problem

- Where to store messages
- Stored locally: takes up too much device storage
- Stored on cloud: Servers need to be scalable

Solution

- Whatsapp stores locally, back up option available
- Telegram stores on cloud (more storage efficient, security concerns)

Performance

Problem

- Messages should be sent/received immediately
- Servers should be able to handle lots of messages, large group chats, large files, etc.

Solution

- Whatsapp: max 1024 group participants
- Telegram: max 200k group participants
- Viber: max 40 video call participants
- Compressed media

Authentication

Problem

- Auth protocols should be secure
- Bad actors logging in from other devices

Solution

- Only one account per person/phone number
- Whatsapp Web: approve authentication on phone
- Old devices should lose access

Scalability

Problem

- Some factors affecting performance cannot be limited (number of active users etc.)

Solution

- Microservices to figure out higher load functionalities
- Horizontal scaling

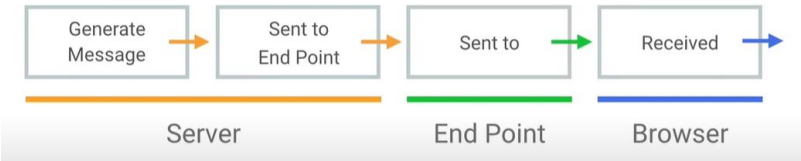
Stakeholders

- Owners
- Funders
- Users
  - Individuals
  - Small Businesses (Whatsapp)
- Employees
  - Developers
  - System Administrators
  - Test Engineers
  - Maintainers & Assessors
  - Contributors (Signal)

Requirements and Quality Attributes

Real-Time Text Transmission

- Performance
  - Transmit instantly
- Reliability
  - Deliver without damage
- Data Privacy
  - About the stored data
- Security
  - Protect the data



Requirements and Quality Attributes

Emojis

- Unicode Encoding
  - UTF-8
- Compatibility

U+1F600 : 😀
U+1F603 : 😃
U+1F604 : 😄

😀 😁

☺ □

Requirements and Quality Attributes

File Transfer

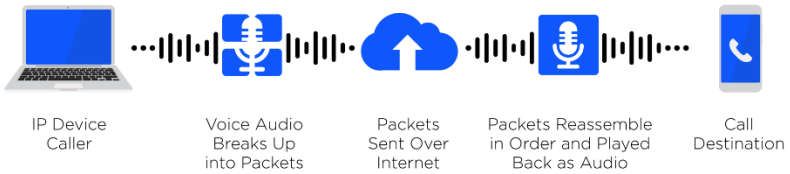
- Documents, multimedia and more
- FTP, TCP, HTTP

- Performance
  - High speed, large files
- Reliability
  - Data should not be lost nor damaged
- Security
  - Interception by third parties

Requirements and Quality Attributes

VoIP

- RTP
  (Real-Time Transfer Protocol)
- Performance
  - High quality audio
  - Synchronization (for calls)
- Security
  - Call should not be intercepted by third parties



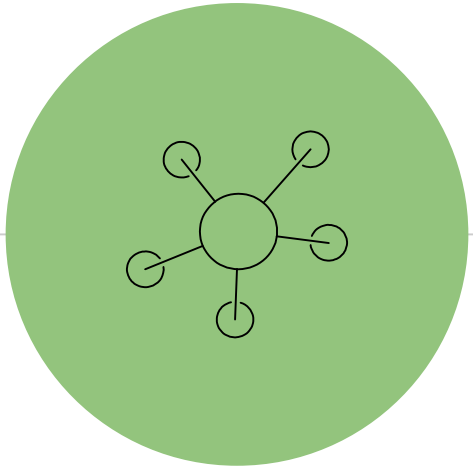
Requirements and Quality Attributes

Video Chat

- Broken into packets,
  Compress,
  Reassemble

- TCP & UDP
  - Higher quality
  - High speed

- Robustness
  - Multiple participants
- Performance
- Security

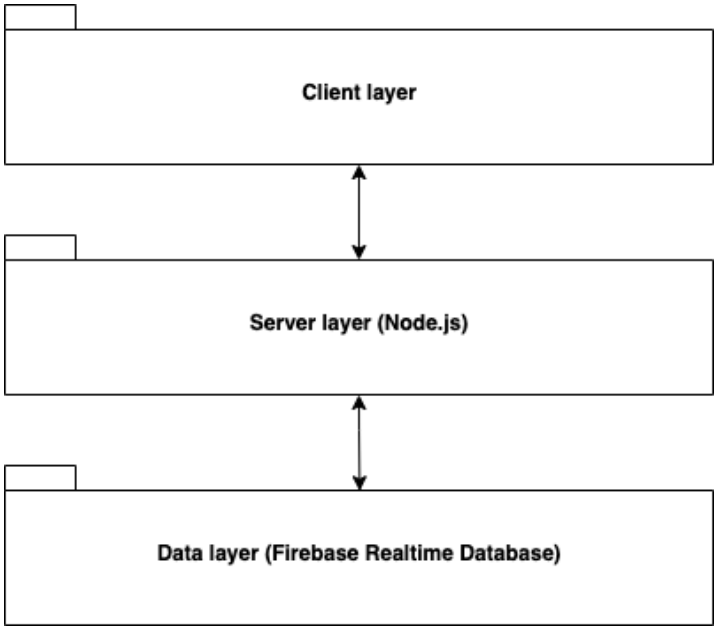


Views and Rationale

Views and Rationale

Module styles - Layered Style

- The client layer. The client performs API calls.
- The server layer. Responsible for communicating with the data layer.
- The database layer. Firebase Realtime Database.



Views and Rationale

Module styles - Decomposition Style

- Design IM system into a modifiable and extendable one.
- Decomposed our system into different parts based on their responsibility,<<is responsible for>>.
- Based on the requirements and the technical problems.

Views and Rationale

Module styles - Authentication & Authorization Module

- Handles the process of any account access attempt.
- Authenticates the device regularly to keep access tokens update-to-date.

Views and Rationale

Module styles - Transmitter Module

- Is responsible for all forms of transmission.
- Includes the transmission of text messages, media, and files.
- Responsible for keeping messages in order.

Views and Rationale

Module styles - Encryption Module

- Key module in the system.
- Encrypts all data that is sent or received.

Views and Rationale

Module styles - Compressor Module

- Ensures that the servers will not be overwhelmed.
- All large files and VoIP will be compressed.

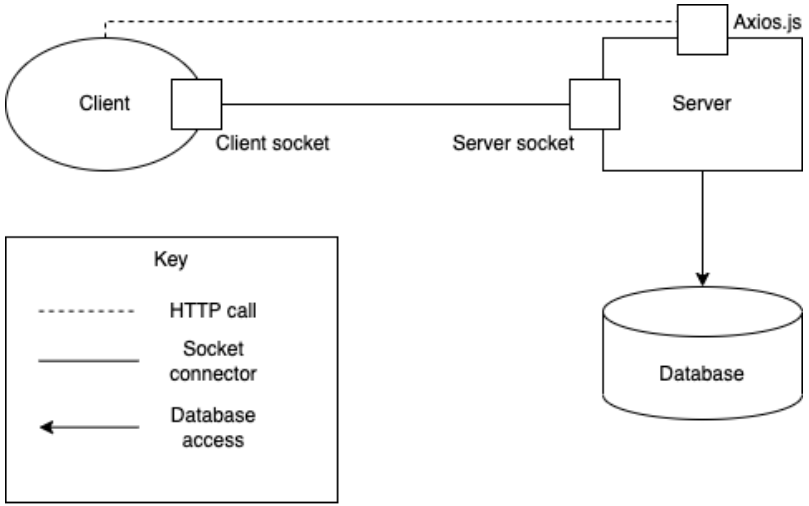
Module styles - VOIP Module

- Transmitting messages and VoIP are different.
- Pub/Sub vs Client-Server(peer-to-peer)
- Handles the data from the beginning to the end of a VoIP.
- Includes compressing, encryption, breaking into packets, receiving, ordering, decrypting and then playing.

Views and Rationale

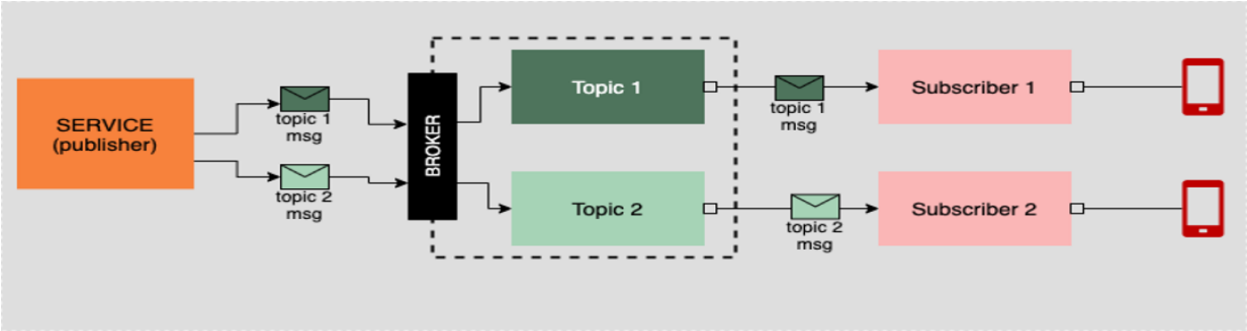
C&C Styles - Call-Return Style

- Client to server via several methods.
- Such as sockets and HTTP calls.
- Uses the Axios.js framework to facilitate HTTP calls.



C&C Styles - Publish-Subscribe Style

- Allows messages to be exchanged in order
- Account for one side, or both, being offline after a message is sent
- Allows for unreceived messages to be kept in the server till the receiver(s) all get a hold of a copy.



Views and Rationale

C&C Styles - Pipe-and-Filter Style

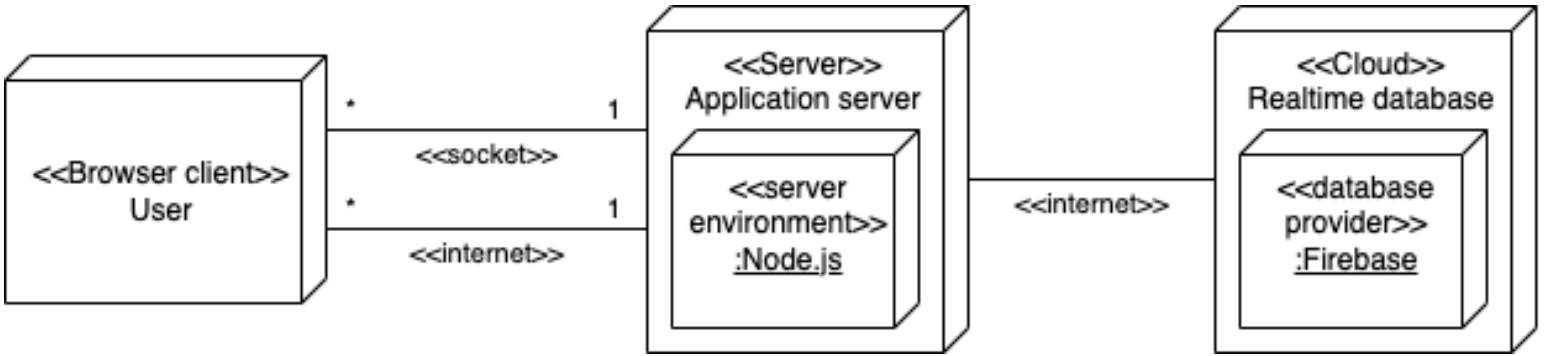
- Used when data gets transferred and/or transformed.
- For media:

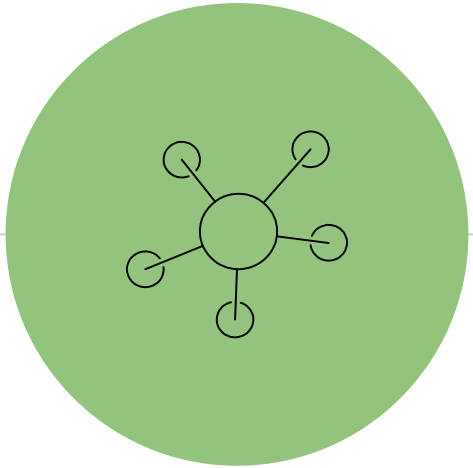
1. Verify the size limit.
2. Generate a thumbnail.
3. Compress the file if needed.
4. Encrypt the file.

The file is ready to be sent.

Allocation - Deployment Style

- One server is used by multiple clients.
- The server is developed in the Node.js.
- Connected to the database in the cloud by Firebase.





Patterns

Patterns

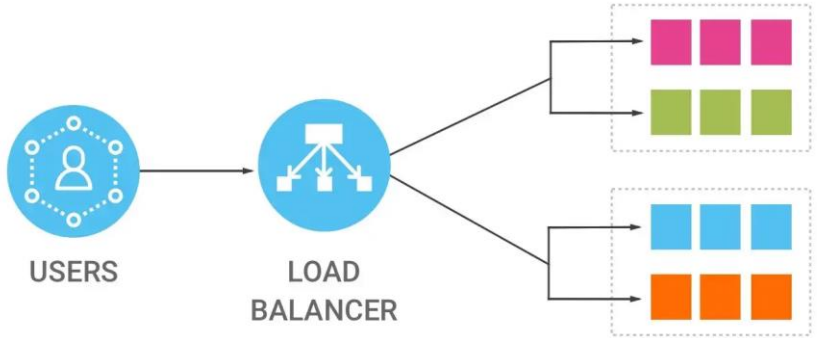
Microservices

- Each process performed on servers. Will have a dedicated microservices.
- These microservice either are sequential or paralleled.

Patterns

Horizontal Scaling

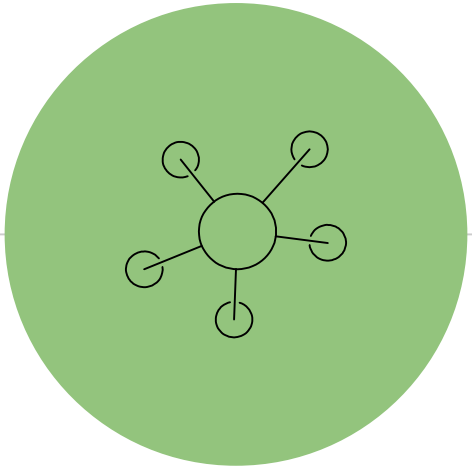
- Allow for high availability.
- Instead of vertical scaling because the needed computing power is not as high as the needed extremely high availability.



Patterns

Three-tier architecture

- System into three separate layers, web (also known as presentation) layer.
- Application layer (the server).
- Database layer where the data is saved.



Implementation

Implementation

Name and logo

- Chatty

Implementation

Features

- Login & Register
- Create new chat
- View chats
- Send and receive messages

Technologies

- Firebase Authentication, Firestore database
- Socket.io for sockets
- Node.io, express.io

Implementation

Subscribe to a chat

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

Sending messages through sockets

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

Implementation

Receiving messages

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

Loading old chat from database

```
// Shortened http endpoints
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) => {})
```

Demo time

Thanks!

Your questions are welcome!