Home

Last edited by **Andrew Nehalchuk** 6 months ago

Feather: Chat - Home

A cutting-edge chat application built on Kafka's event-driven architecture, React frontend for seamless user interaction, Redis cache for lightning-fast data retrieval, all orchestrated on Kubernetes for scalability and resilience. Stay connected like never before with Chatify's real-time messaging and smooth user experience. Join the conversation today!

Pages

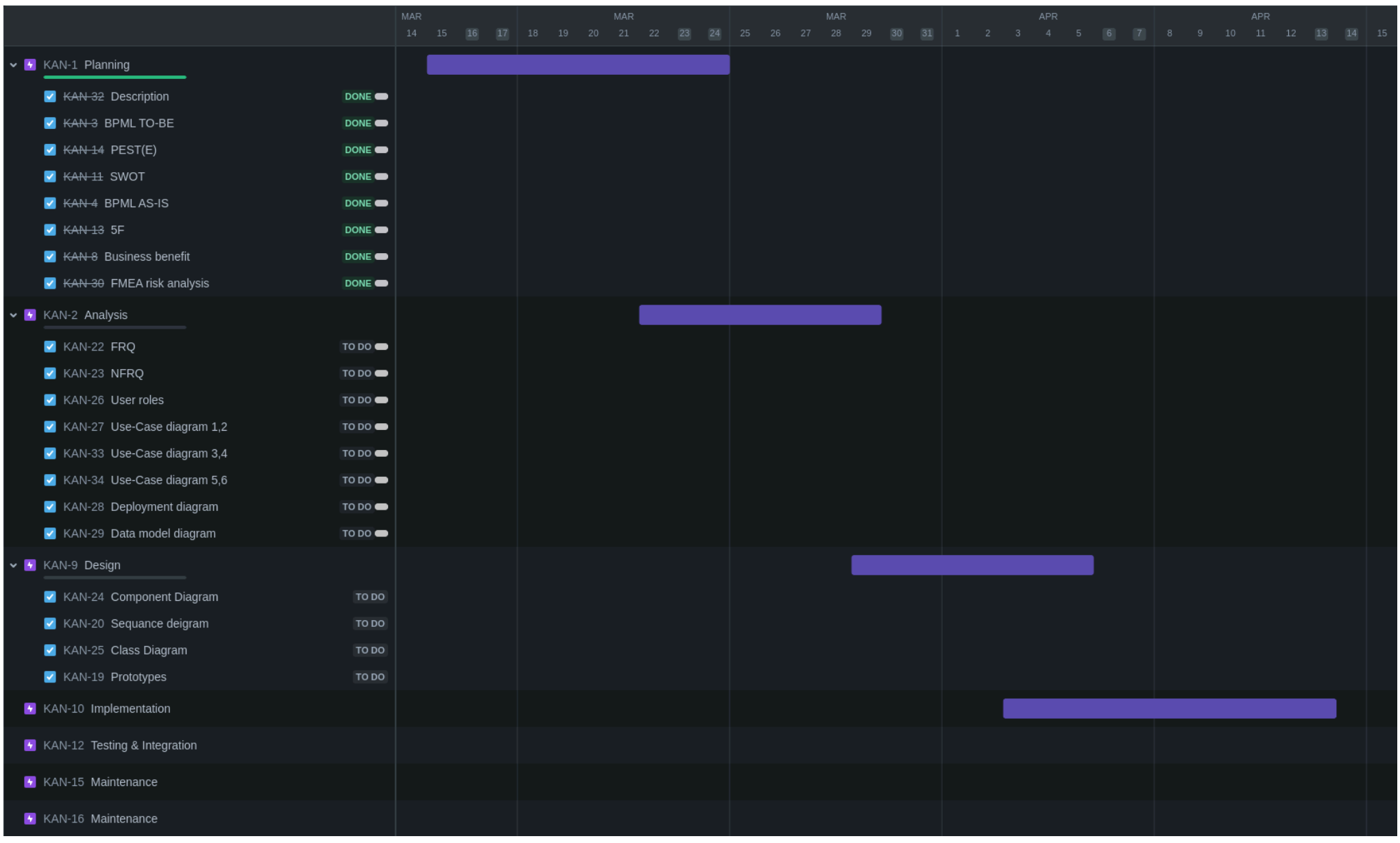
1. Planning
2. Analysis
3. Design
4. Implementation
5. Maintenance

Workflow

Jira RACI matrix

- View PDF Report

Jira timeline



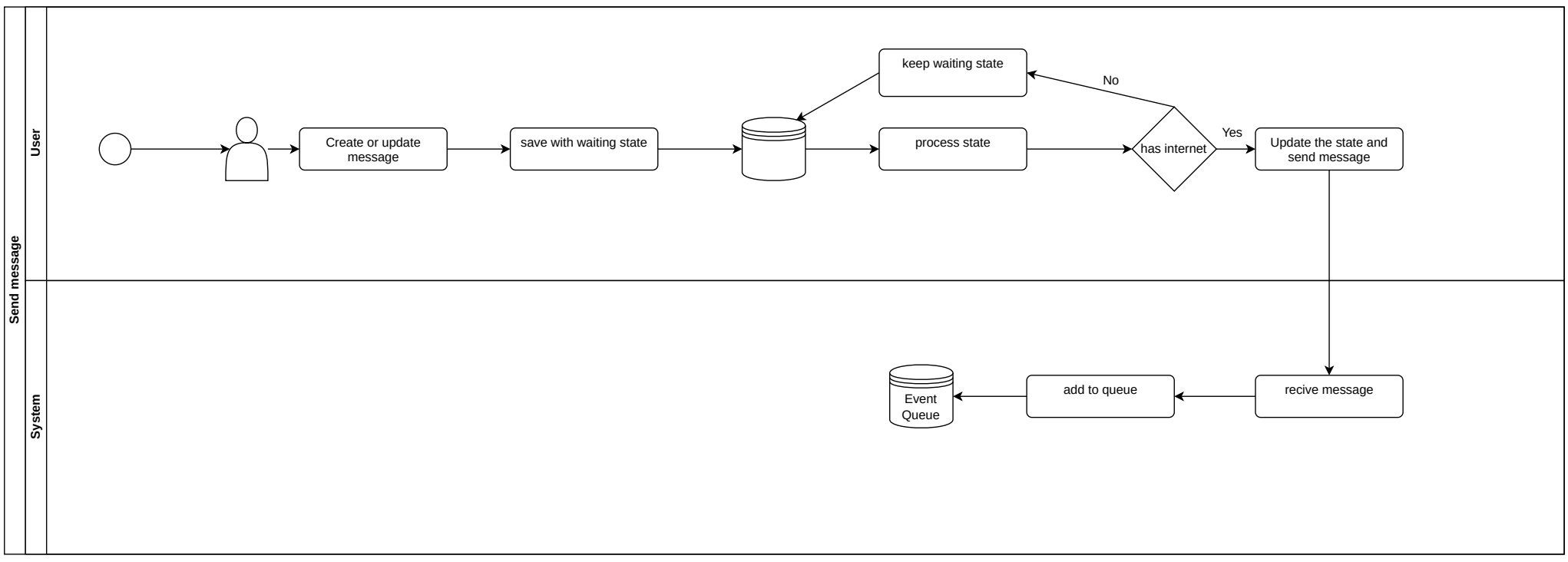
planning

Last edited by **Andrew Nehalchuk** 5 months ago

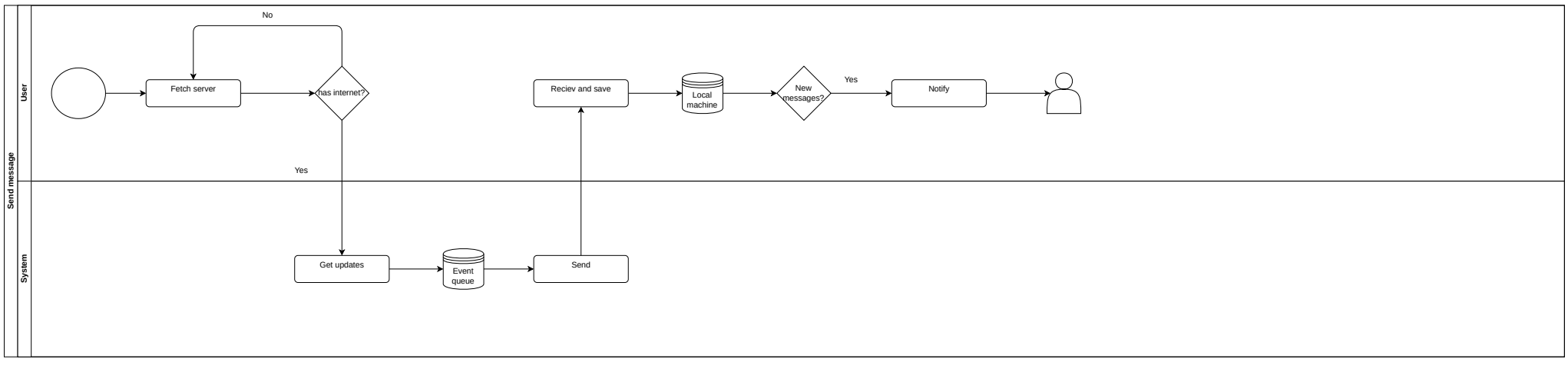
AS-IS

Nowadays, people use popular instant messengers such as WhatsApp and Viber. And as experience shows, these messengers are not famous for their reliability and stability. There are many known cases where user data was leaked. It can also be noted that most instant messengers have a very inconvenient user interface, especially for Messenger

Create or alter message process

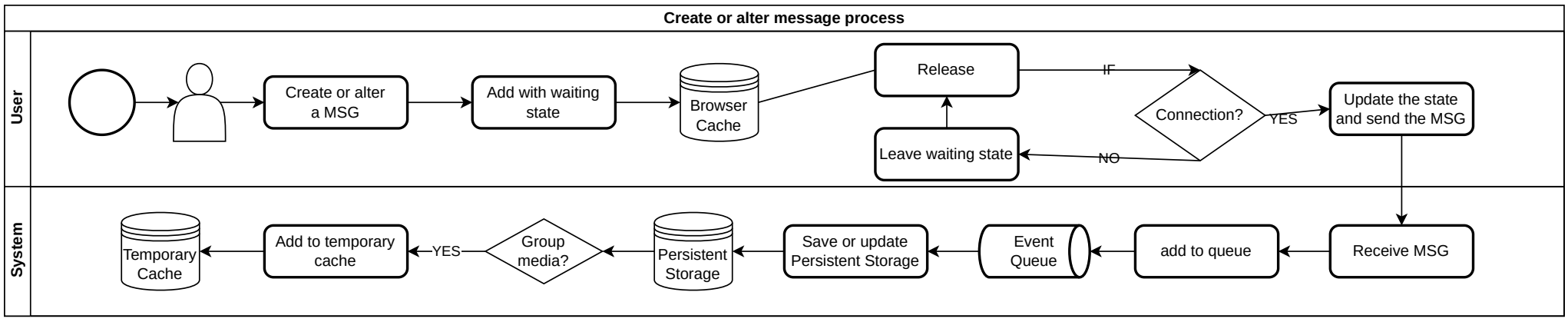


Receive or update message process

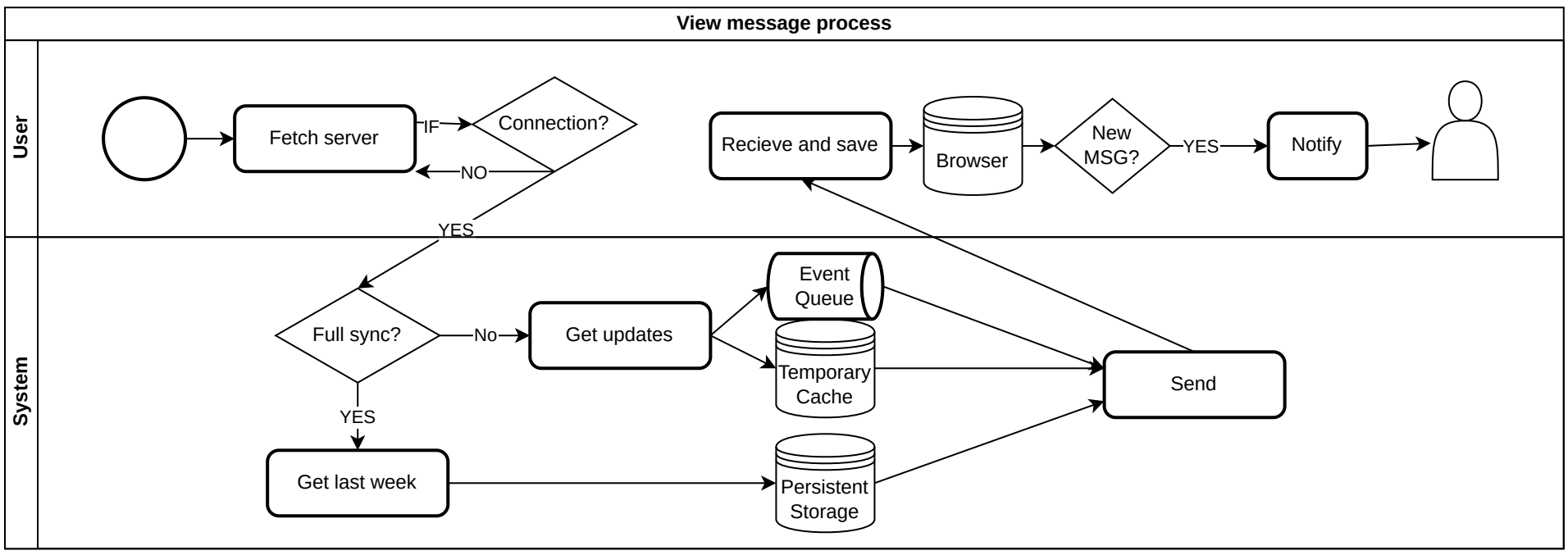


TO-BE

Create or alter message process



Receive or update message process



Business benefit

- **Introducing ChatFlow**: next-generation chat application with event-driven Kafka architecture.
- **Increase efficiency**: Real-time messaging reduces latency.
- **Cost savings**: Centralized communication reduces costs (probably).
- **Resource Automation**: Streamlined processes save time and labor.
- **Integrated Data**: No more data transfer; everything stays in the application.
- **Enhanced control**: Easily track projects and employee performance.

SWOT analysis

Strengths:

- **User friendly** : The user-friendly interface of Father Chat makes it easy for users to navigate the app and engage with its features without encountering complexities. This simplicity enhances user experience and encourages prolonged usage.
- **Available on the Web**: Father Chat's availability on the web extends its accessibility beyond mobile devices, allowing users to seamlessly transition between different platforms while maintaining their conversations. This feature increases convenience and flexibility for users who prefer to use messaging apps on desktops or laptops.
- **Data safety**: Father Chat prioritizes data safety and privacy by implementing robust encryption measures, ensuring that user communications and information remain secure from unauthorized access or interception. This commitment to data safety enhances user trust and confidence in the platform.
- **Group chat functionality**: Father Chat offers robust group chat functionality, enabling users to create and participate in group conversations with multiple participants. This feature facilitates communication and collaboration among teams, communities, or social groups, enhancing the app's utility and versatility.

Weaknesses:

- **Branding recognition**: Father Chat may suffer from a lack of brand recognition compared to well-established messaging platforms, which could hinder its ability to attract new users and compete effectively in the market. Building brand awareness through strategic marketing initiatives becomes crucial to overcoming this weakness.
- **Limited Features**: While Father Chat provides essential messaging features, it may lack some advanced functionalities found in competing apps, potentially limiting its appeal to certain user demographics who seek more comprehensive and innovative features. Continuously updating and expanding the app's feature set can address this weakness and cater to diverse user preferences.
- **Market Competition**: The messaging app market is highly competitive, with numerous established players vying for user attention and market share. Father Chat faces stiff competition from these incumbents as well as new entrants, necessitating strategic differentiation, innovation, and marketing efforts to carve out its niche and gain traction among users.

Opportunities:

- **Niche Targeting**: Father Chat could target niche markets such as families, religious communities, or specific interest groups, capitalizing on its unique branding and features tailored to these demographics.
- **Integration with Services**: By integrating with other services such as payment systems, e-commerce platforms, or productivity tools, Father Chat could expand its functionality and attract new users seeking integrated solutions.
- **Global Expansion**: There may be opportunities for Father Chat to expand its user base globally, particularly in regions where privacy concerns are high or where there's a demand for secure messaging platforms.

Threats

- **Security Concerns**: Any breaches in security or privacy issues could severely damage the reputation of Father Chat and lead to a loss of user trust and confidence.
- **Regulatory Challenges**: Changes in regulations regarding data privacy and encryption could impact the app's ability to operate in certain regions or impose additional compliance costs.
- **Rapid Technological Changes**: The messaging app landscape is constantly evolving, with new technologies and features emerging regularly. Father Chat must adapt quickly to stay competitive and meet evolving user expectations.
- **Competitive Pressure**: Established messaging platforms and new entrants in the market pose a significant threat to Father Chat's growth and market share, requiring continuous innovation and strategic positioning to remain competitive.

5F Analysis - Michael E. Porter's Five Forces Model

Rivalry among existing firms

The market is highly competitive. There are many dominating competitors on a market, such as "Telegram", "Whatsapp", "Viber", "Facebook messenger", "Snapchat".

Potential entrants

The threat of new entrants is low. The market is not that attractive, because of the high rivalry among big players on a market, but on the other hand is very profitable. The new entrants may compete with existing firms, only if it catches audience's attention.

Suppliers

The bargaining power of suppliers is low.

Buyers

The bargaining power of buyers is high. The buyers have a lot of choices, so they can easily switch to another product.

Substitutes

The threat of substitutes is high. There are many substitutes on a market, such as leading social networks.

PEST(E) Analysis

Political

Political factors such as data privacy laws, encryption policies can significantly impact our app. Compliance with government regulations in various countries where the app operates is crucial.

Economic

Economic factors such as economic stability, inflation rates, and currency fluctuations can affect overall profit of the app.

Social

The app's popularity may vary across different demographics, such as age groups or geographic regions, affecting its user base and market penetration.

Technological

Dependence on a internet infrastructure and compatibility with various devices and operating systems are critical for user accessibility and satisfaction.

Environmental

While the app itself doesn't directly impact the environment, the infrastructure supporting it, such as data centers and servers, consume significant amounts of energy.

FMEA Risk Analysis

Mode of Failure	Cause	Effect	Frequency	Severity	Detection	RPN
Unauthorized access to user data, data loss.	Lack of proper access controls, inadequate encryption of user data, or security loopholes in the application's authentication mechanism.	Breach of user privacy.	4	9	9	324
Notification not appearing, notification delay.	Network congestion or server-side processing delays affecting the timely delivery of notifications to users' devices.	Users may miss important messages.	8	10	4	320
Message not delivered, message delay.	Insufficient server capacity or infrastructure failure due to increased user load or technical issues.	Users unable to send or receive messages.	3	10	6	180
Password leakage, authentication failure.	Weak password hashing algorithms or improper handling of user credentials leading to security vulnerabilities.	Compromise of user accounts.	2	9	10	180
App crashes, slow response time.	Software bugs, memory leaks, or compatibility issues with different device configurations leading to instability and crashes.	Negative user experience, potential data loss.	5	9	3	135

analysis

Last edited by **Andrew** 1 month ago

FRQs

For our application, we will need the following functional requirements:

- Registration
- Login
- Managing profile
- Creating the groups
- Sending the messages
- Sending the files
- Editing/deleting the messages

NFRQ

This directory contains non-functional requirements

Backend

- Java 17
- Apache Kafka
- Elasticsearch
- Maven
- Kotlin
- Spring

Frontend

- Support in browsers FireFox, Chrome, Opera, Edge, Safari
- TypeScript
- React
- User-friendly GUI

Database and Infrastructure

- Cache(REDIS)
- DB(PostgreSQL)
- Docker
- K8S

User Roles

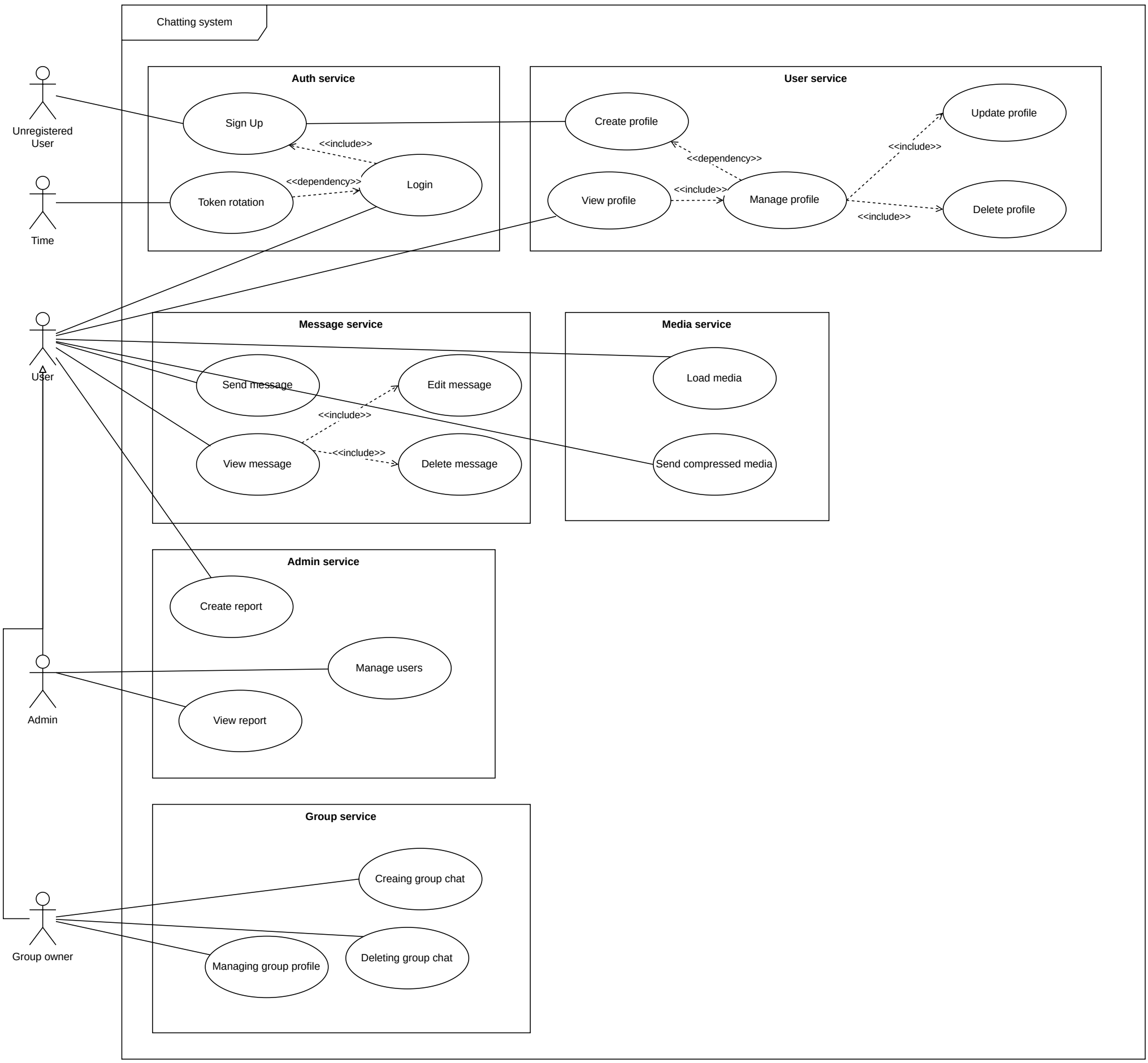
The following roles are available in the system:

- Unregistered user - Can see login & registration pages.
- User - Regular registered user who can send, receive, delete and change messages. They also can manage their profile and settings.
- Group - A set of role applied to group chats:

    - Group owner - Has advanced rights compared to regular users. Can moderate content, delete messages, or block users who violate community rules.
    - Group member - regular member who can send/delete/edit messages and join/leave the group.
- Administrator - Has full access to the system. Can see reports. Can manage users, groups. Can also moderate content, delete messages, or block users who violate community rules.

Use Cases

Complex use-case diagram



1. User Registration

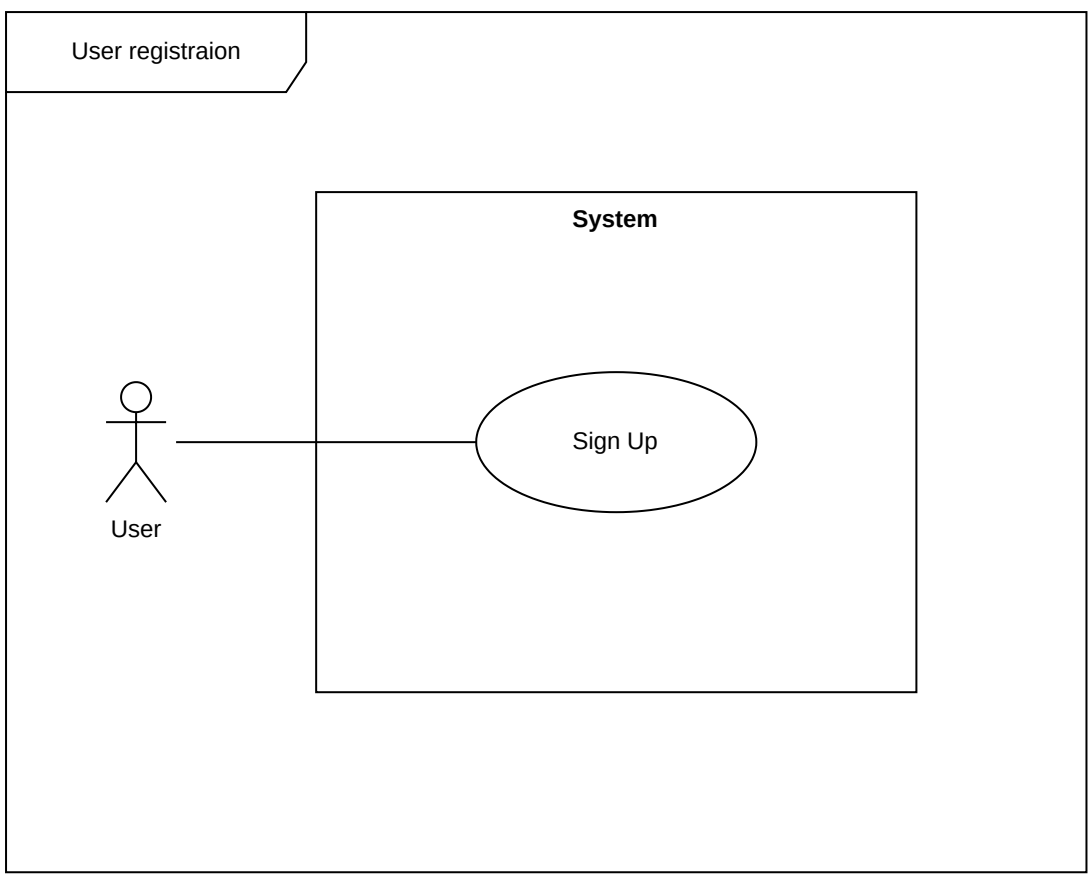
1. The new user launches the chat messaging application.
2. The system presents the user with the option to register for a new account.
3. The user selects the "Sign Up" option.
4. The system prompts the user to provide the some personal information.
5. The user enters the required information into the registration form.
6. The system validates the provided information.

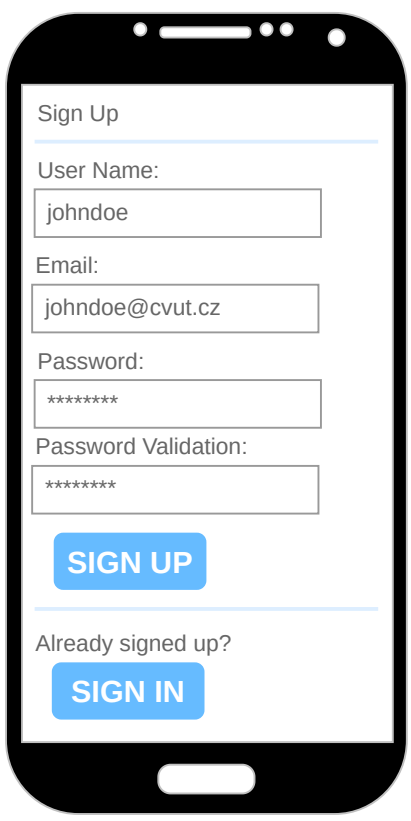
If the provided information passes validation:

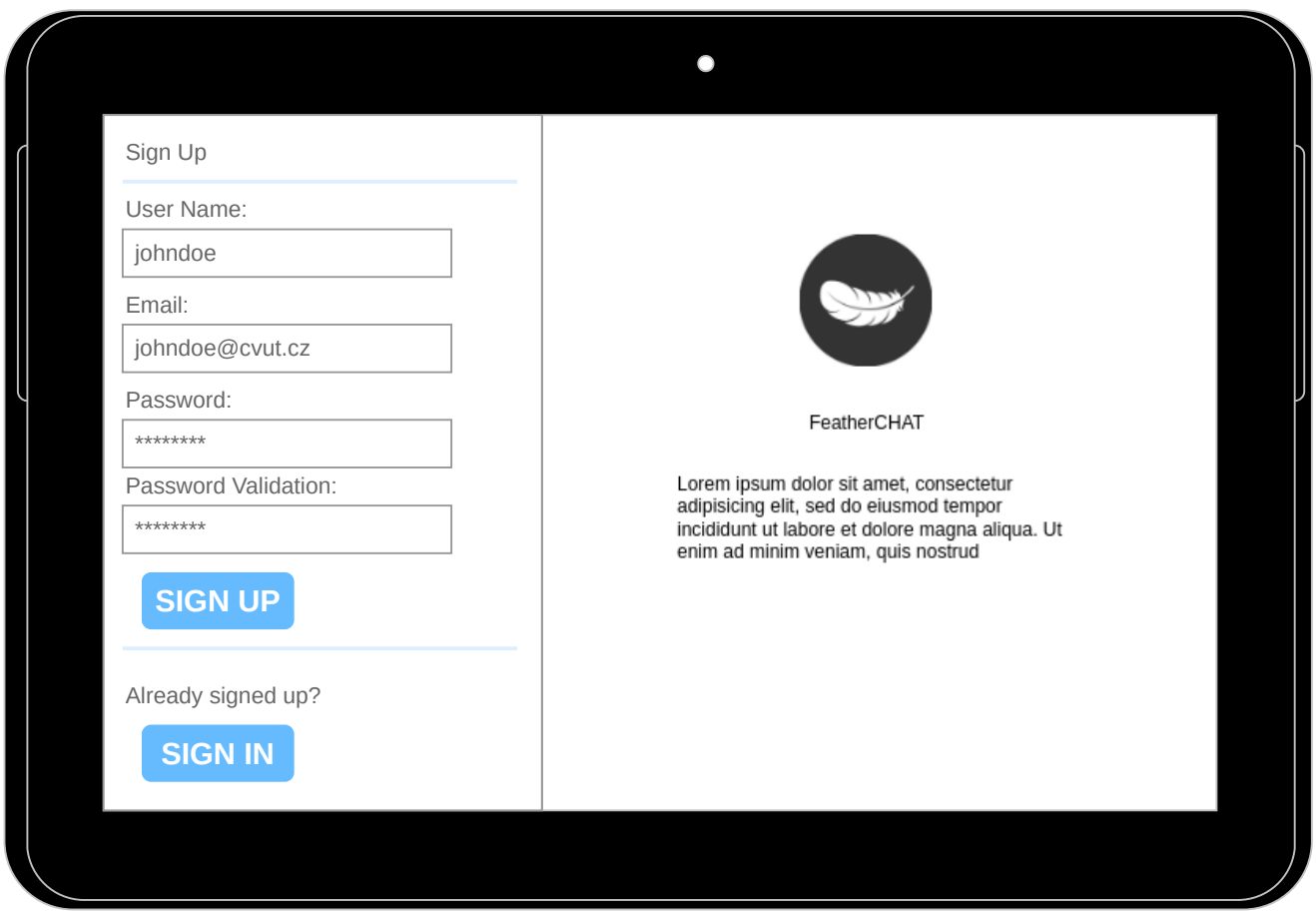
7. The system creates a new account for the user.
8. The system verifies provided information and activates the user's account.
9. The user is redirected to the chat messaging platform's login page.

10. The user logs in using their registered email address/username and password. Upon successful login, the user gains access to the chat messaging platform's features and functionalities.

If the provided information fails validation at any step, the system prompts the user to correct the errors and resubmit the registration form. If the user encounters any issues during the registration process, they can contact customer support for assistance.

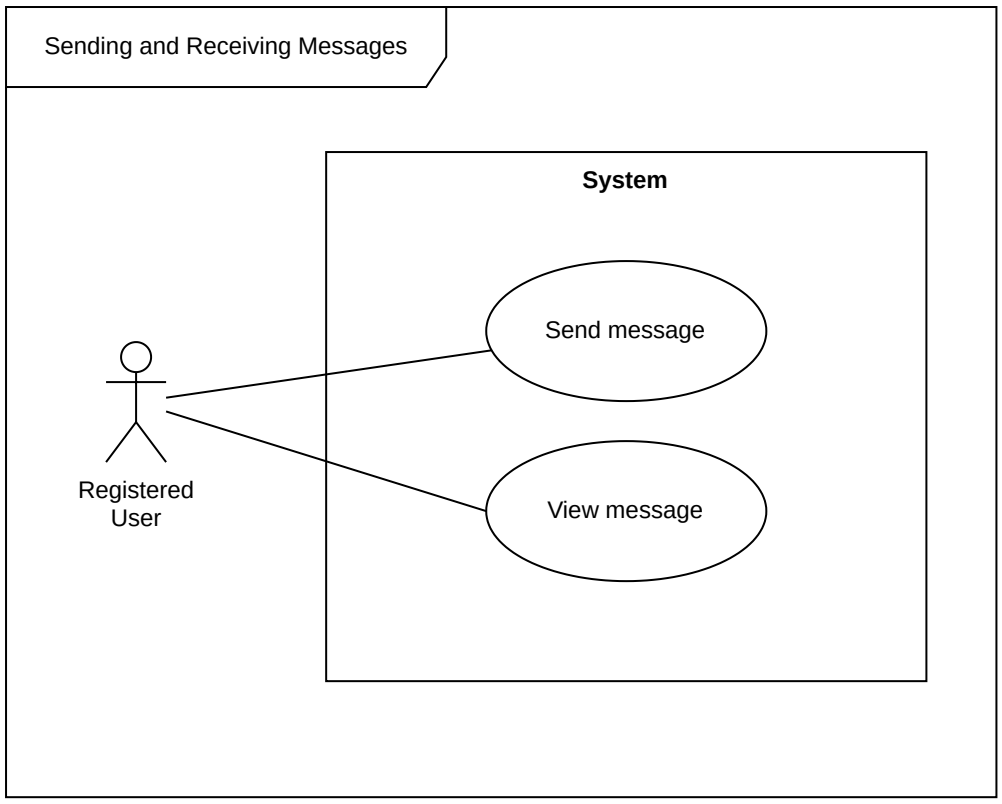


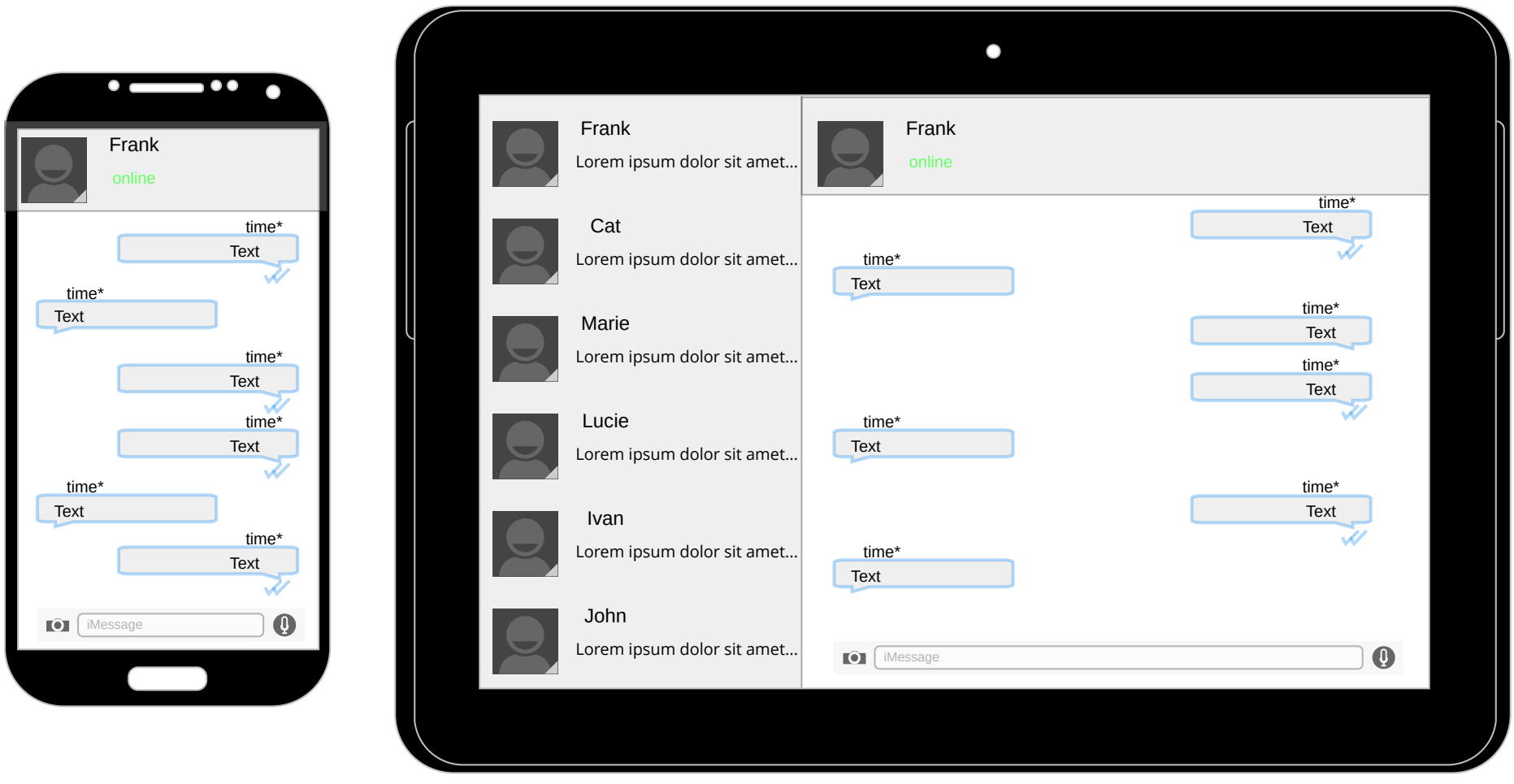




2. Sending and Receiving Messages

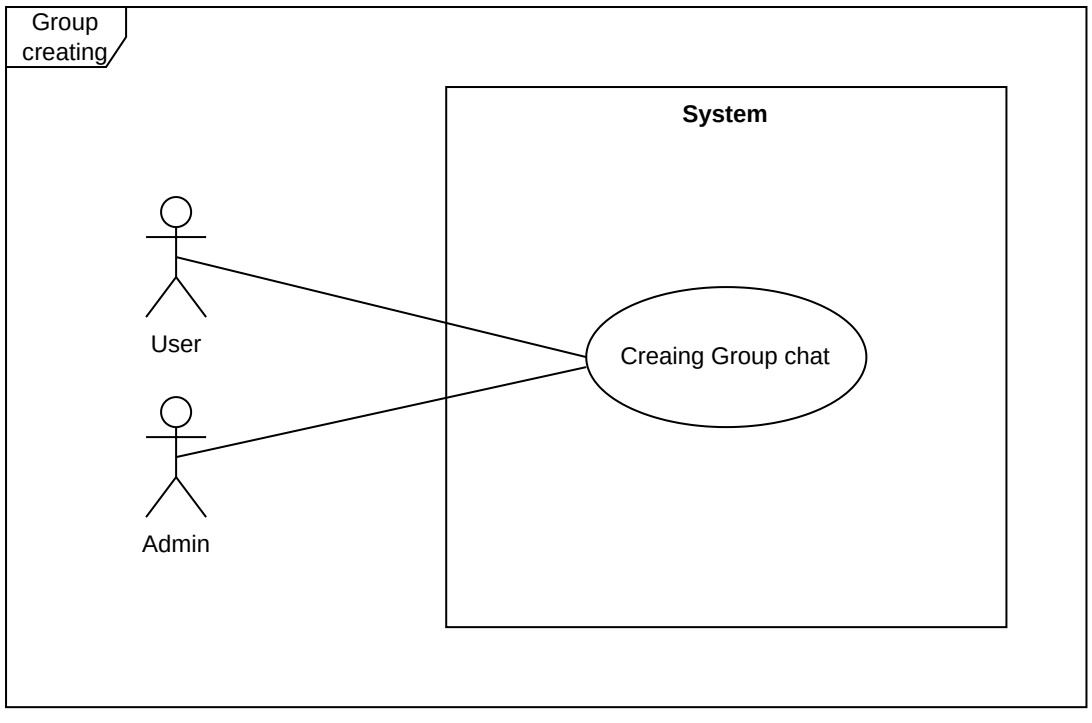
1. The user logs into the chat messaging application using their credentials.
2. The system authenticates the user and displays the main chat interface.
3. The user selects a conversation or starts a new conversation by entering the username.
4. The system loads the conversation history, displaying previous messages exchanged with the selected user.
5. The user types their message in the chat input field and presses the send button.
6. The system sends the message to the recipient's device.
7. The recipient's device receives the message and displays it in the conversation thread.
8. If the recipient is online, they receive a real-time notification of the incoming message.
9. Both users can continue exchanging messages back and forth, with the system ensuring the delivery of each message and maintaining the conversation history.

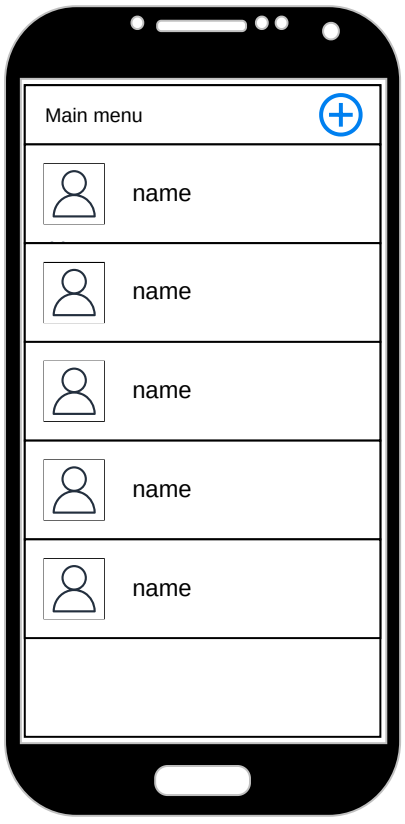


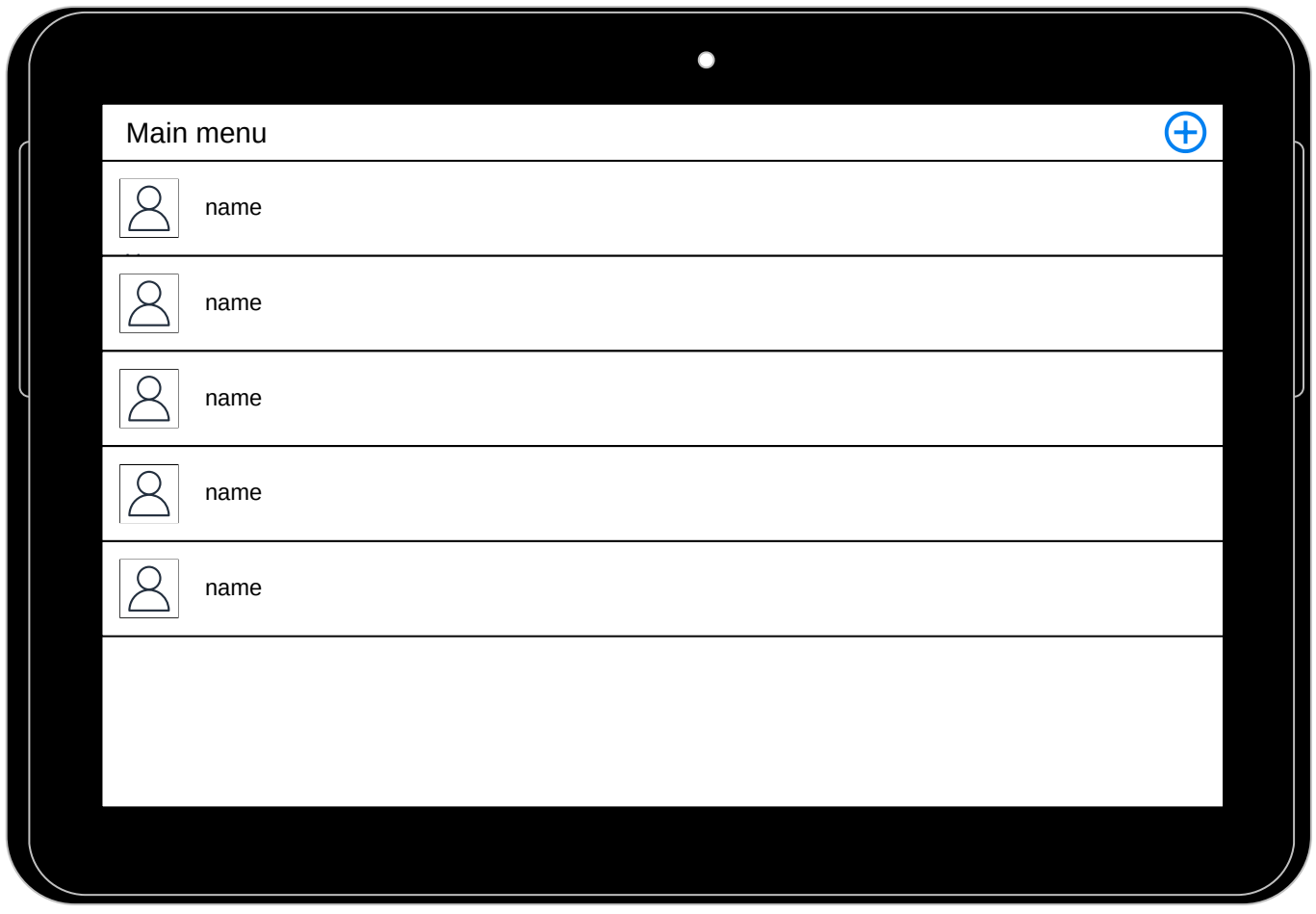


3. Group chat creating

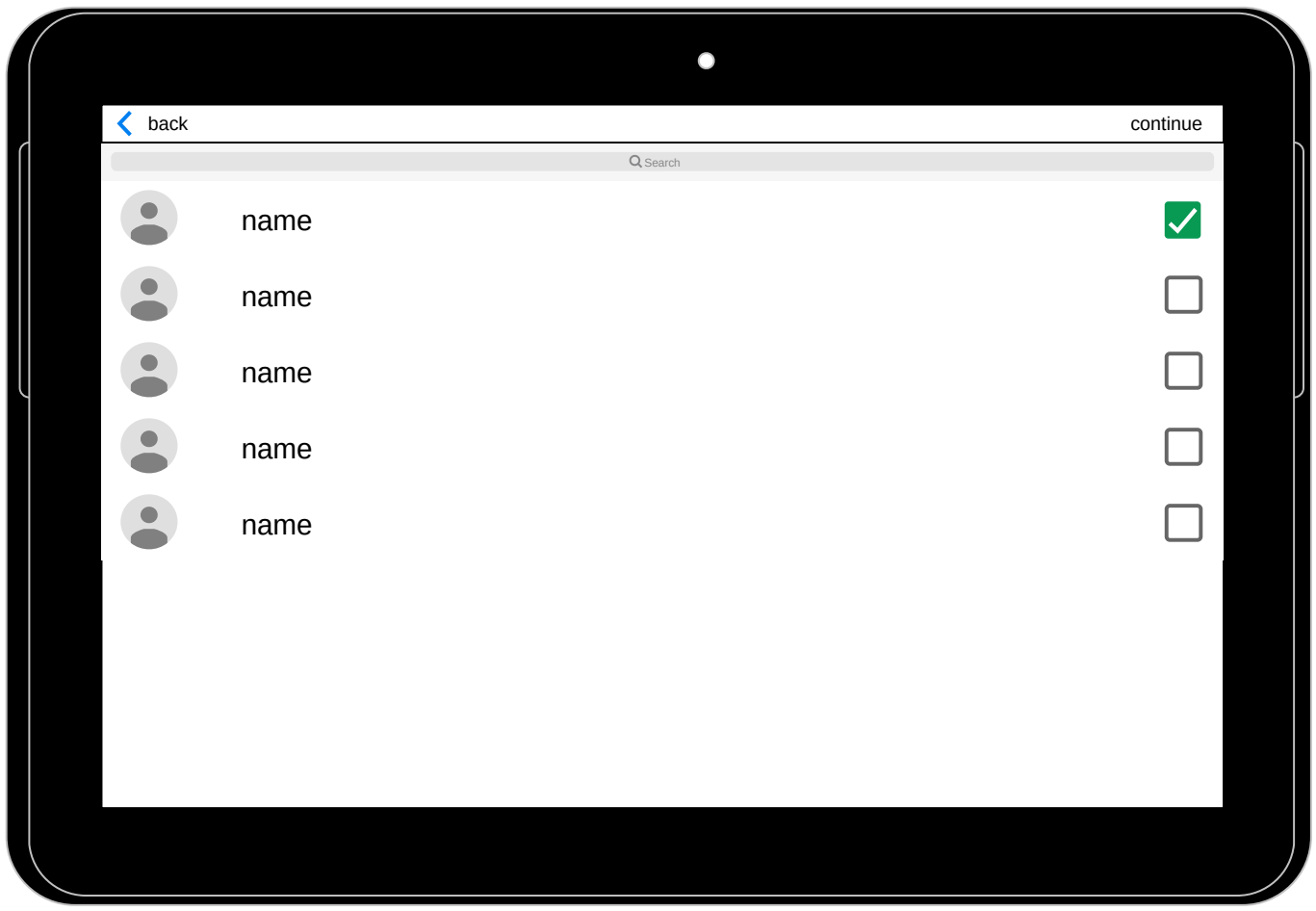
- **User:** From the main page click "creating group chat" button
- **System:** Opens page with list with checkboxes of all friends
  - **User decides to continue:**
    - **User:** Choosing friends and submit
    - **System:** Opens page with group's setting(group photo, name, and button to adding another friends)
      - **User decides to continue:**
        - **User:** Sets up group chat settings and confirm
        - **System:** Opens group chat page
      - **User decides to cancel:**
        - **User:** clicks cancel button
        - **System:** Opens page with list with checkboxes of all friends
  - **User decides to cancel:**
    - **User:** Choose to cancel creating group chat
    - **System:** Redirect to main page

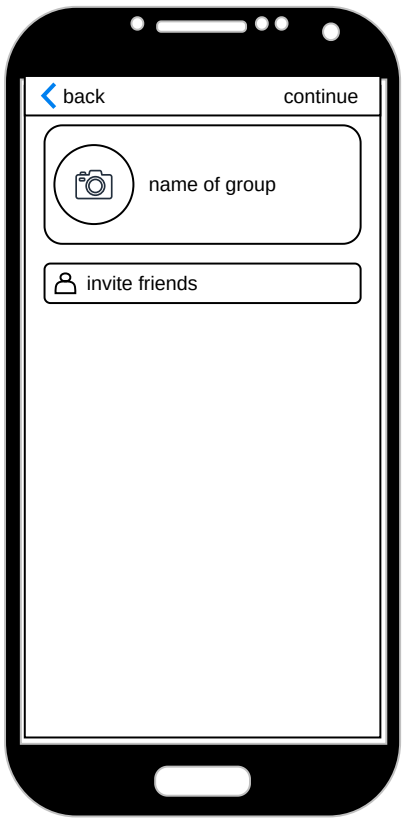

Group
creating
System
User
Admin
Creaing Group chat

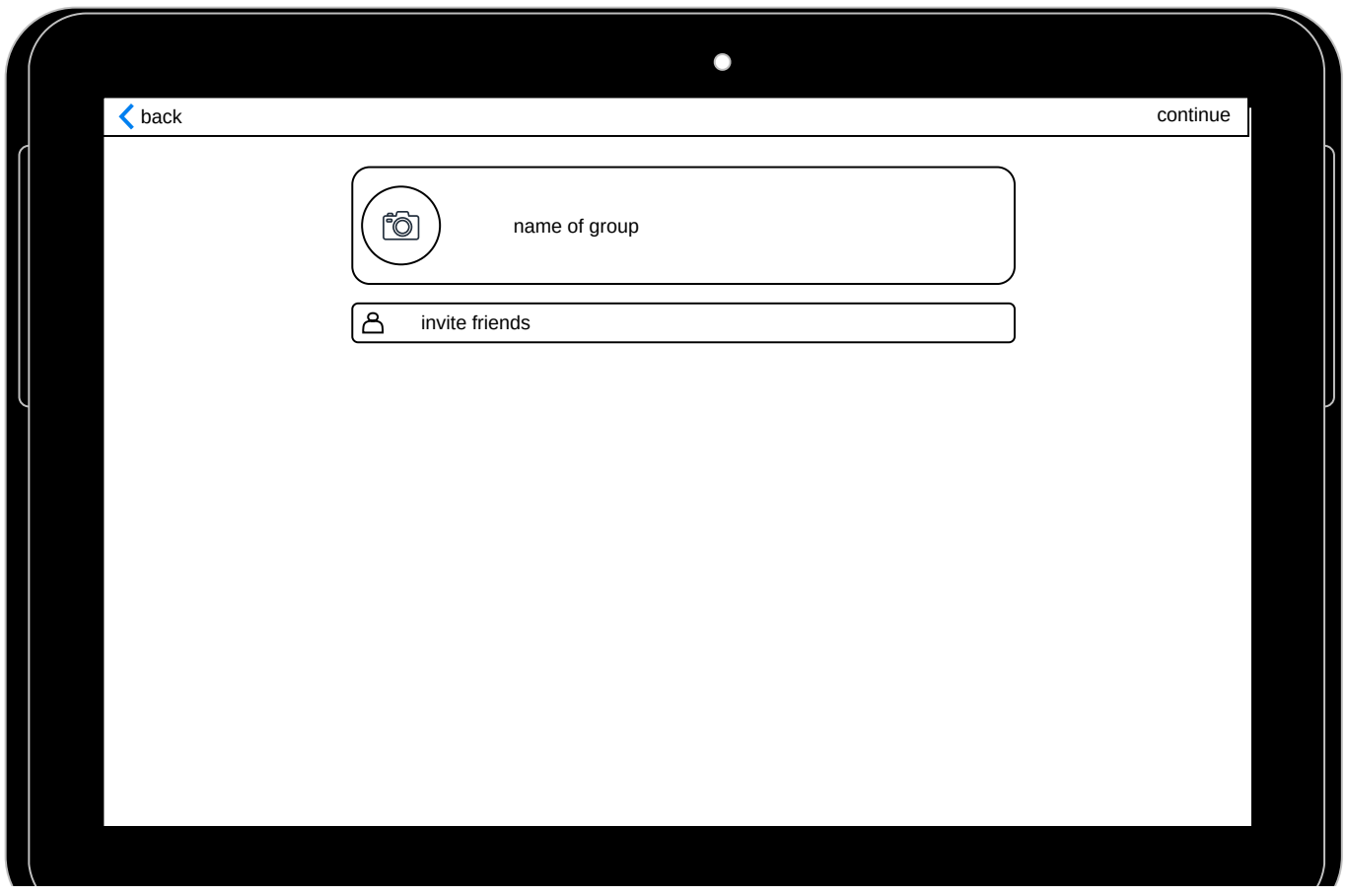

Main menu
name
name
name
name
name


Main menu
name
name
name
name
name


back
continue
Search
name
name
name
name
name

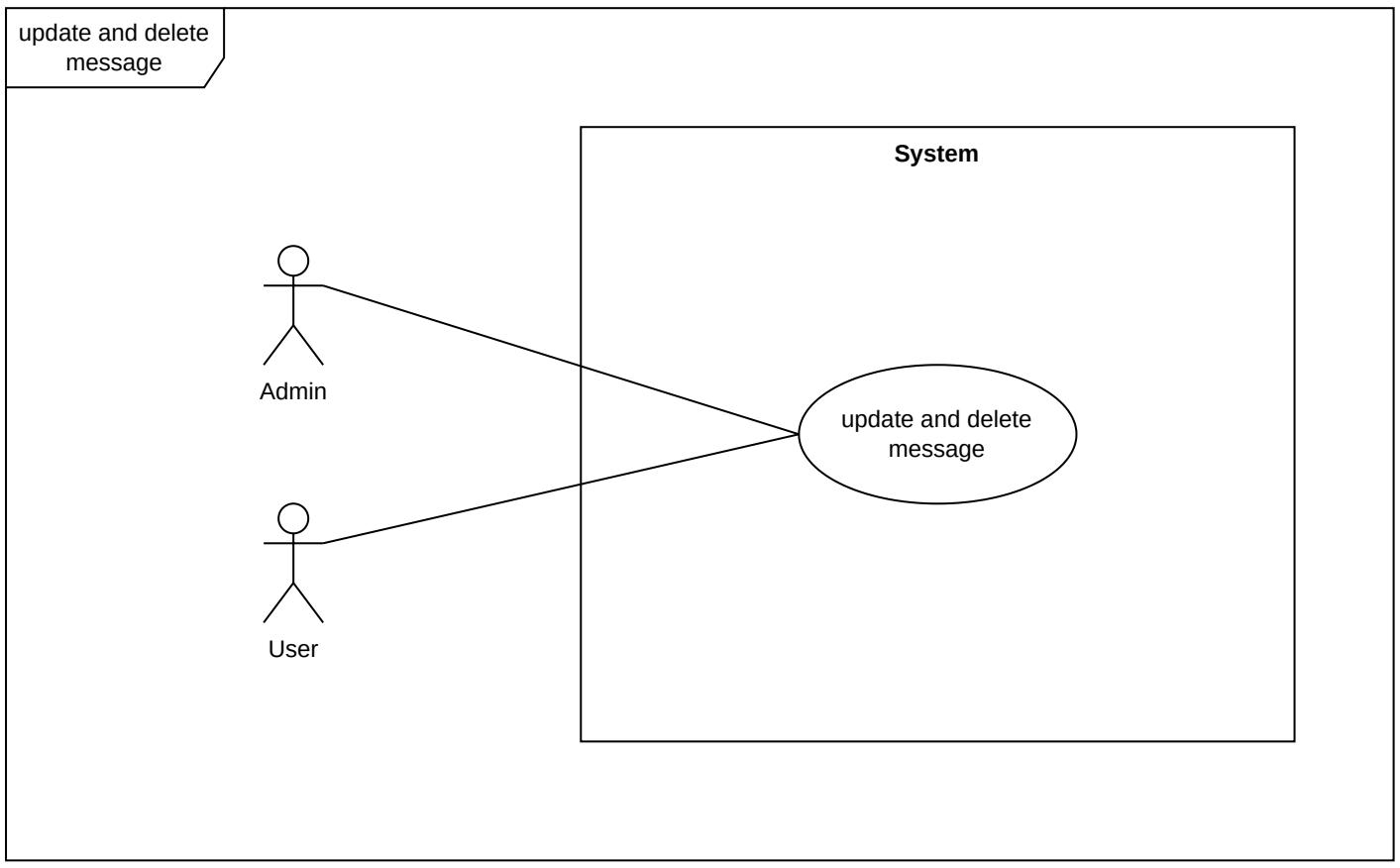

back
continue
Search
name
name
name
name
name

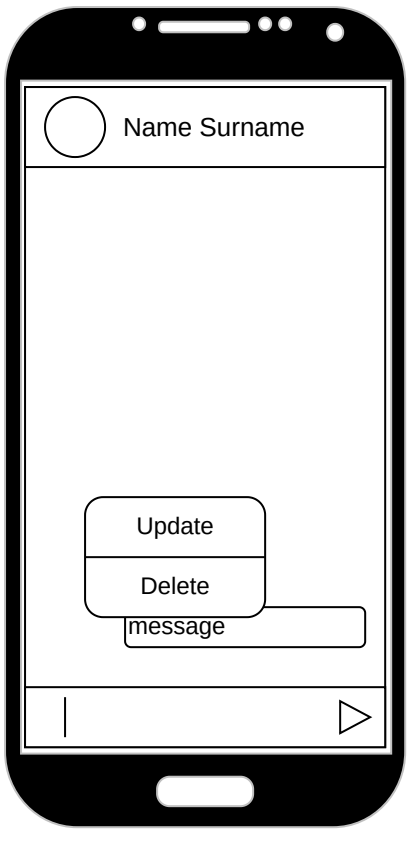

back
continue
name of group
invite friends

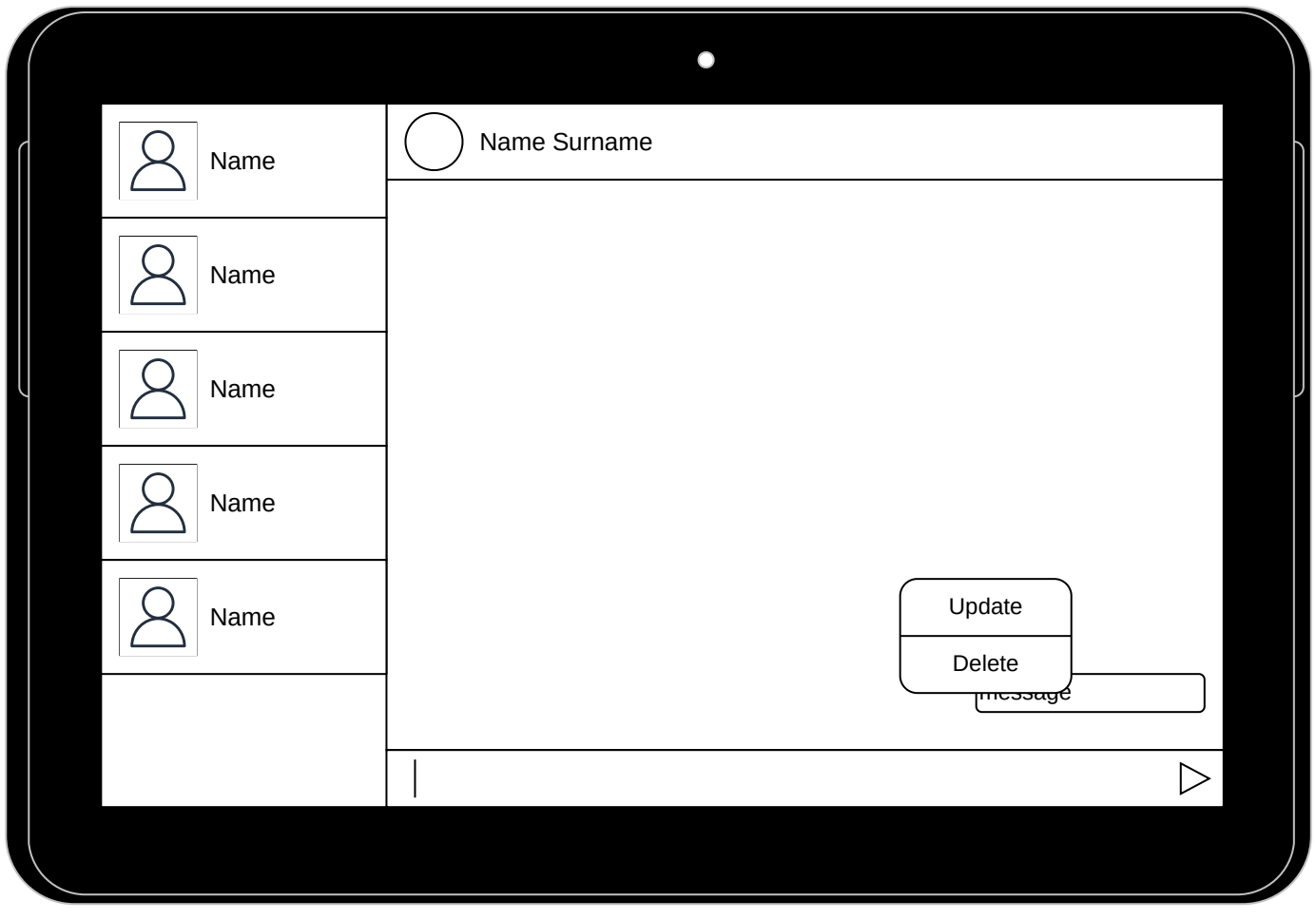

back
continue
name of group
invite friends

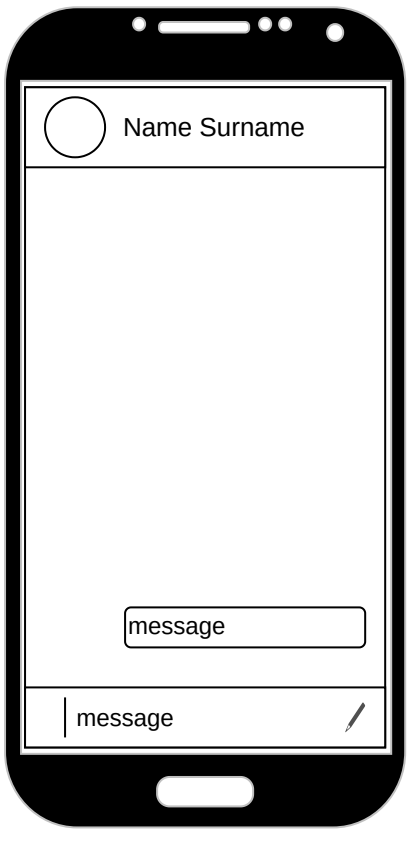
4. Update and Delete message use-case

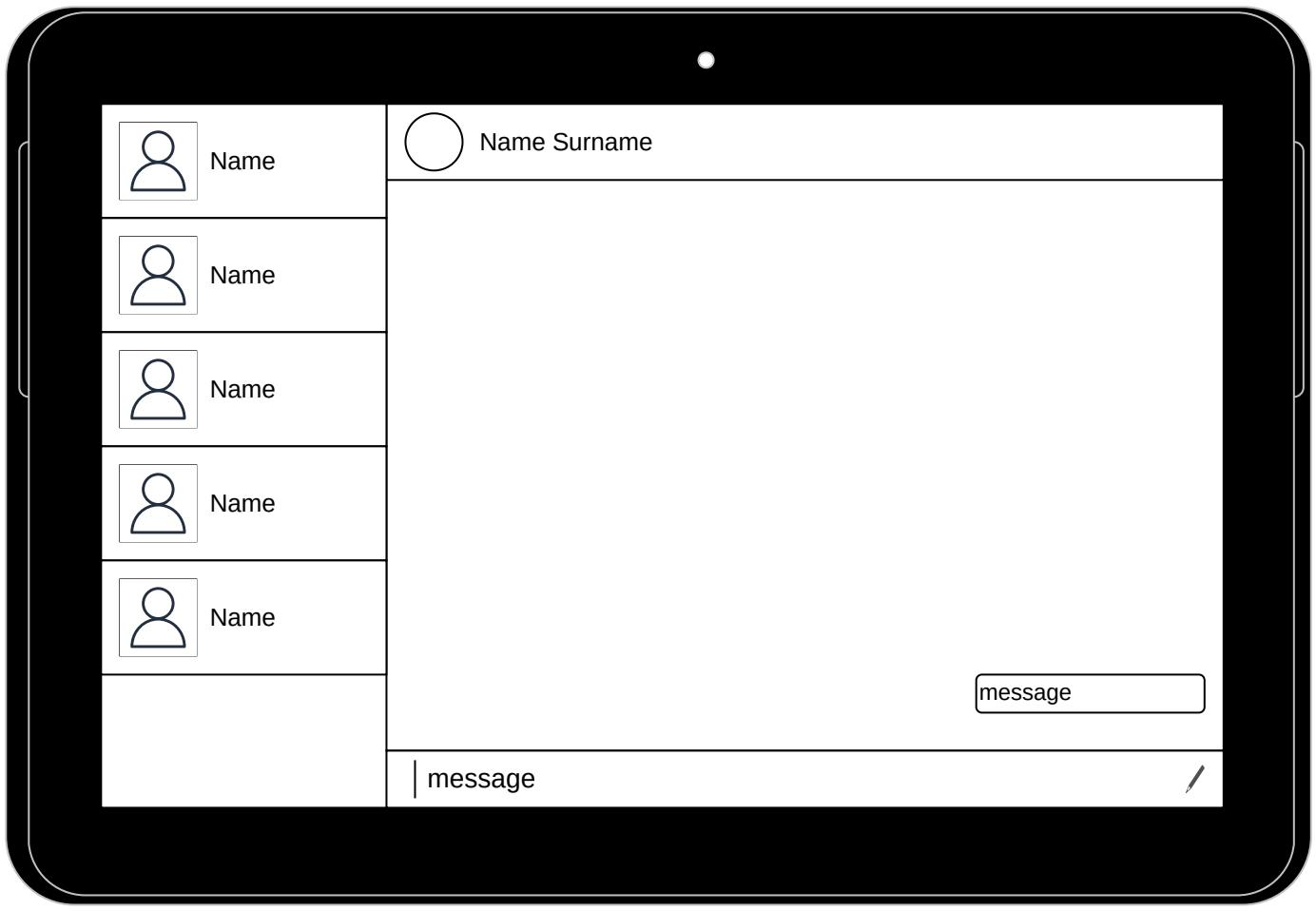
- **User:** Presses the message
- **System:** Displays a pop-up window with the option to edit or delete the message
- **User:** Choose one of the following options
  - **User chooses to delete the message:**
    - **System:** System delete users the message from chat
  - **User chooses to update the message:**
    - **System:** Wraps text from a message into the user's input line
      - **User choose to cancel message editing:**
        - **User:** Click the cancel button
        - **System:** Clear user's input line
      - **User choose to continue editing the message:**
        - **User:** Edit the message in the user's input line and confirm
        - **System:** Updates the message in the chat



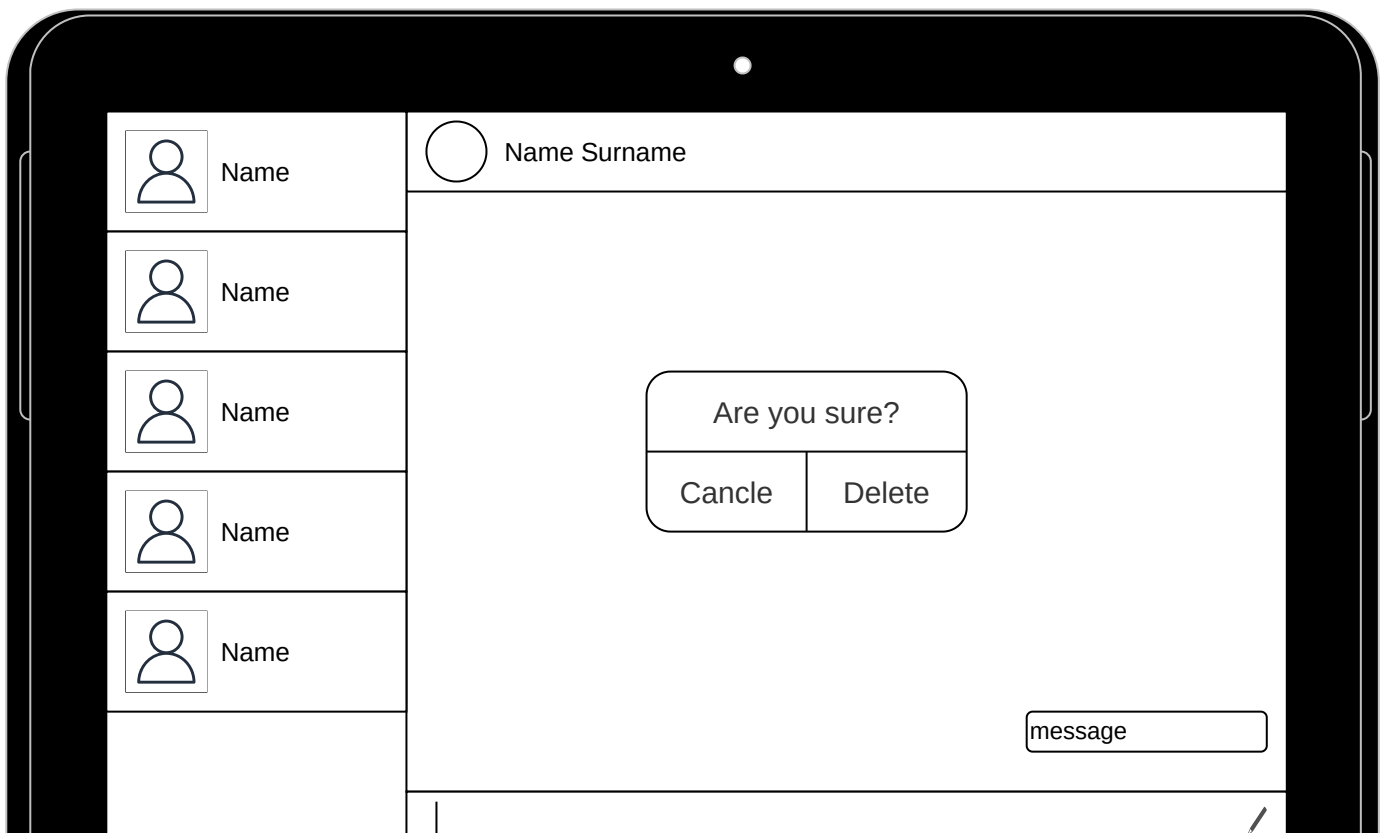

Name Surname
Update
Delete
message


Name
Name
Name
Name
Name
Name Surname
Update
Delete
message


Name Surname
message
message

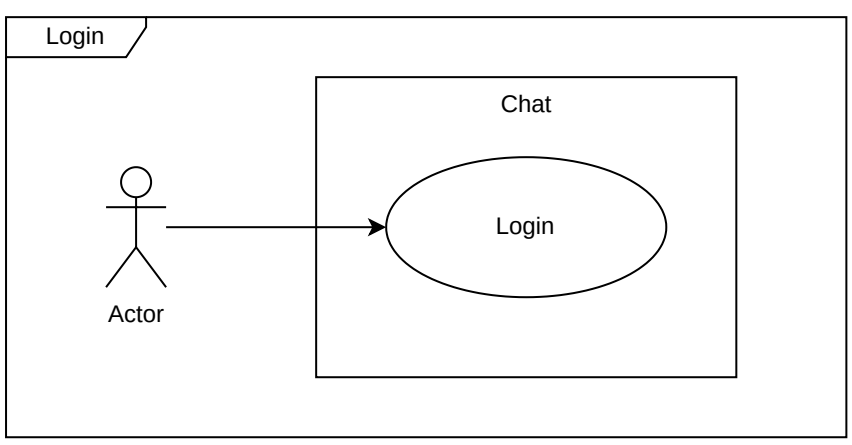

Name
Name
Name
Name
Name
Name Surname
message
message

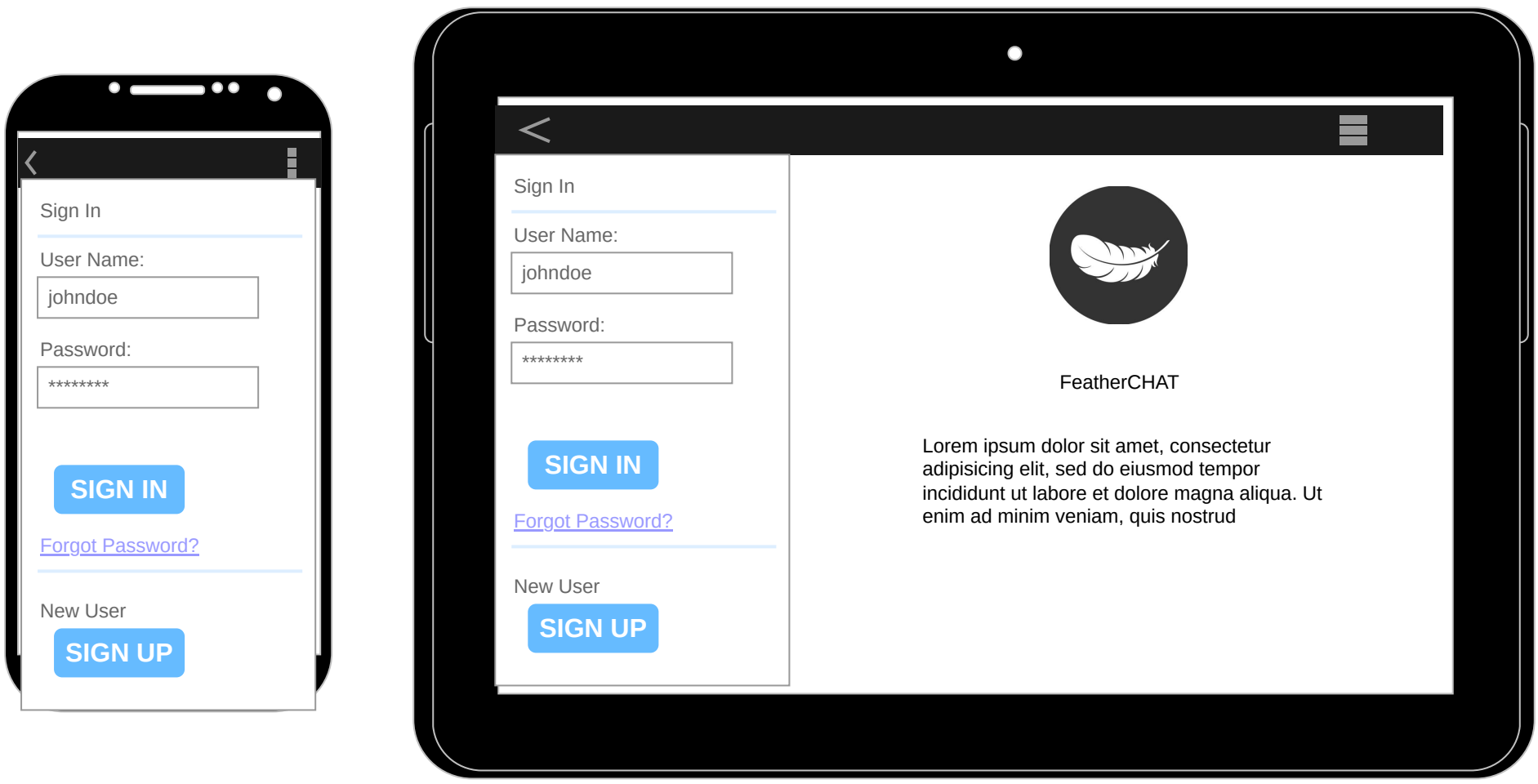

Name Surname
Are you sure?
Cancle
Delete
message
message


Name
Name
Name
Name
Name
Name Surname
Are you sure?
Cancle
Delete
message

5. Login

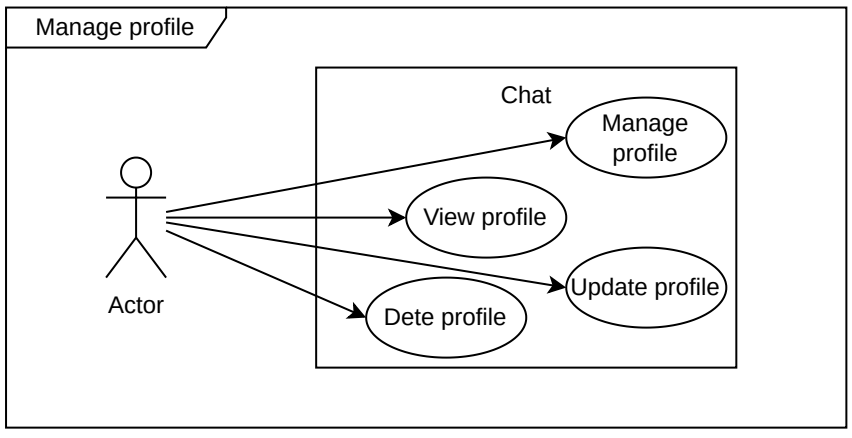
1. User opens the application or website.
2. System presents the login screen.
3. User enters their username or email and password.
4. System verifies the entered credentials.
    - If credentials are correct, proceed to step 5.
    - If credentials are incorrect, prompt the user to re-enter or provide a message indicating the error.
5. System grants access to the user.
6. User gains access to the system and can perform various actions based on their privileges.



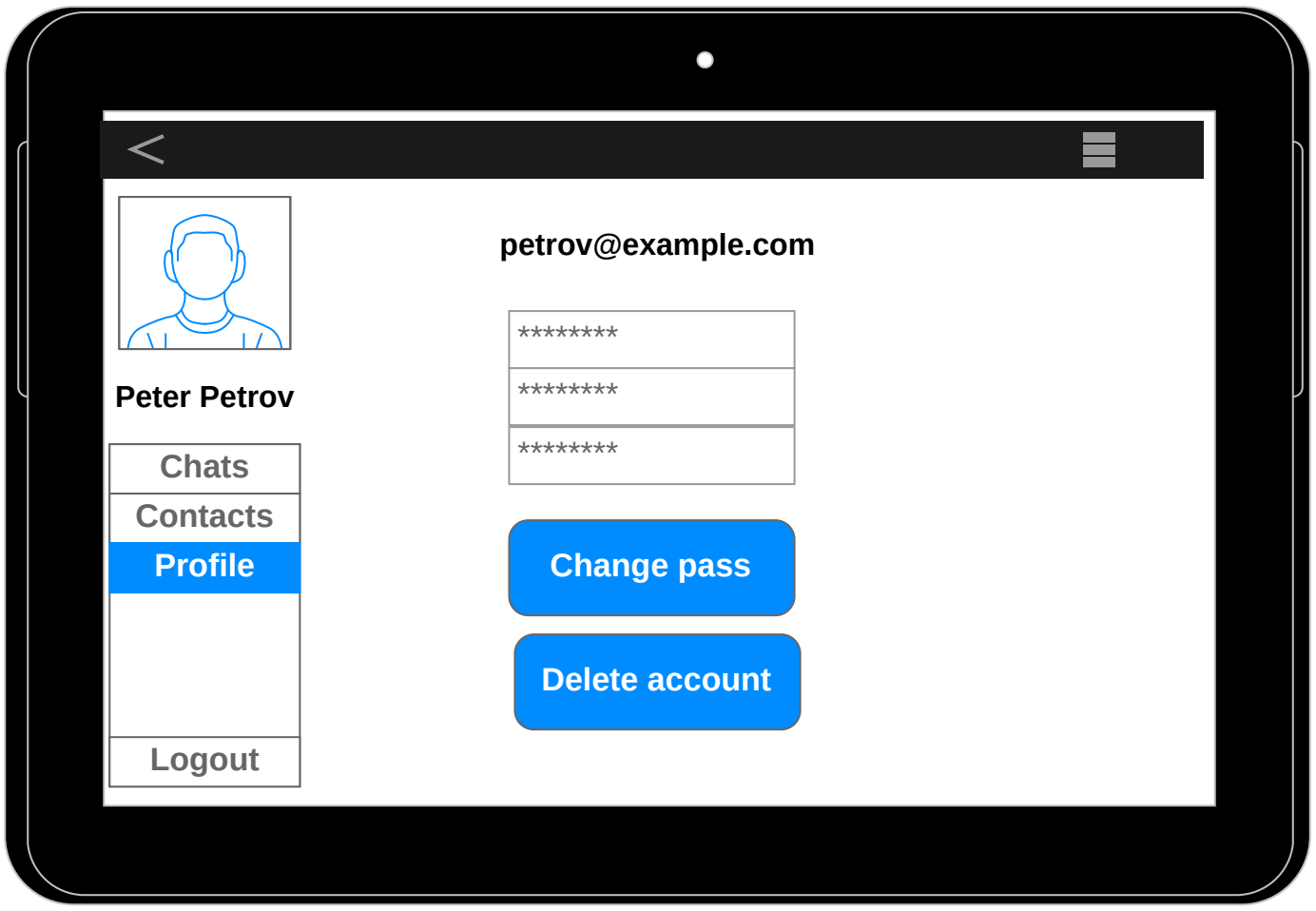


6. Profile

1. If user have login:
    1. See profile photo.
    2. See name and email.
    3. Can change the password.
    4. Can delete profile.




Peter Petrov
petrov@example.com
****
****
****
Change pass
Delete account


Peter Petrov
Chats
Contacts
Profile
Logout
petrov@example.com
****
****
****
Change pass
Delete account

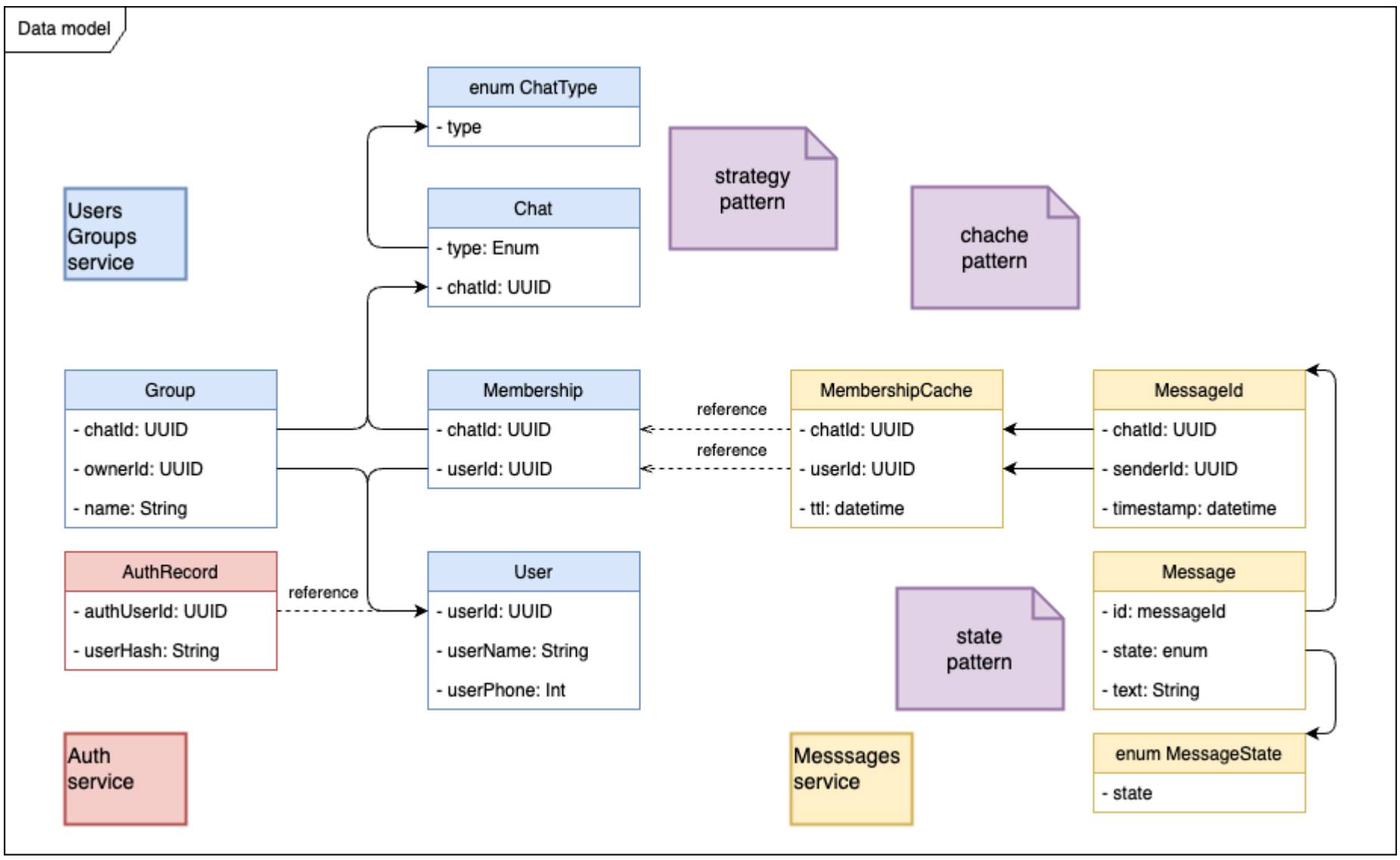
design

Last edited by **andrew** 2 weeks ago

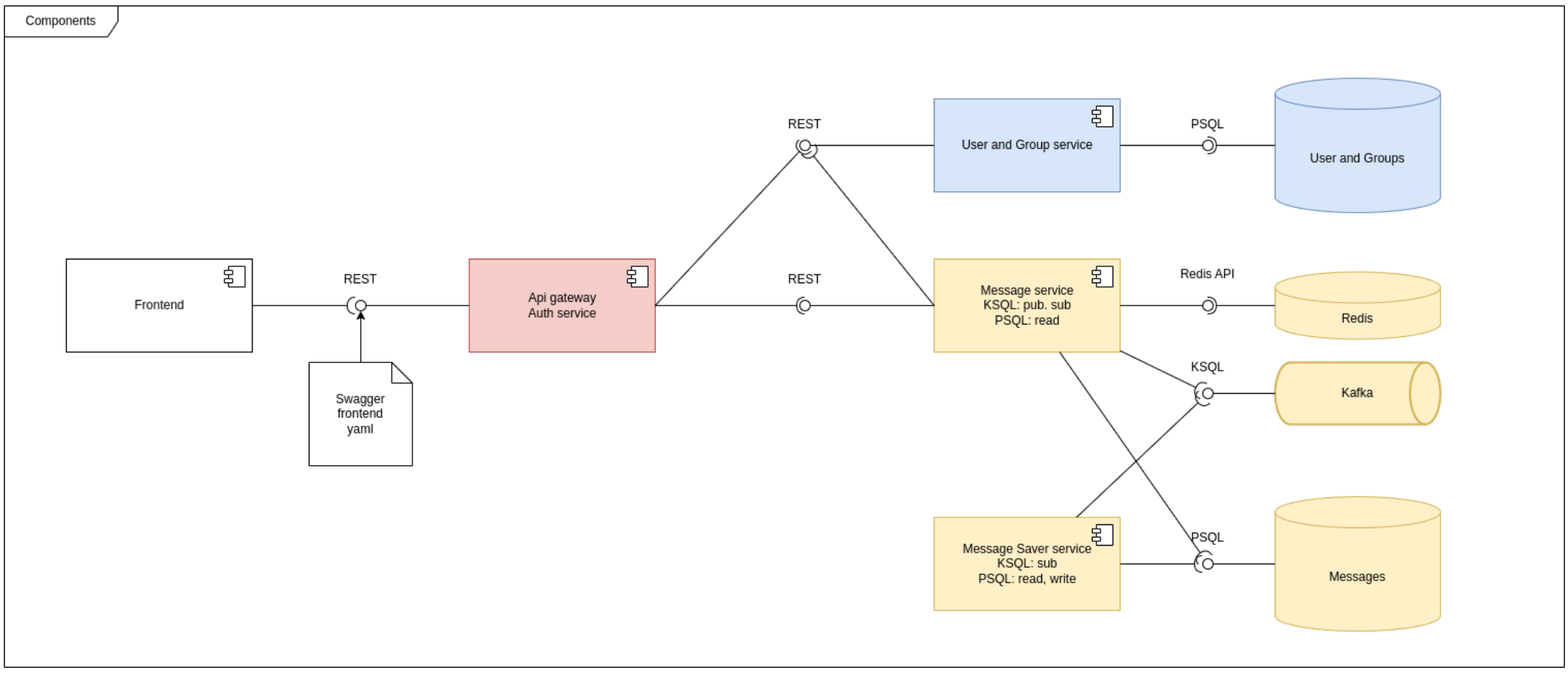
API logic

API logic

Data model

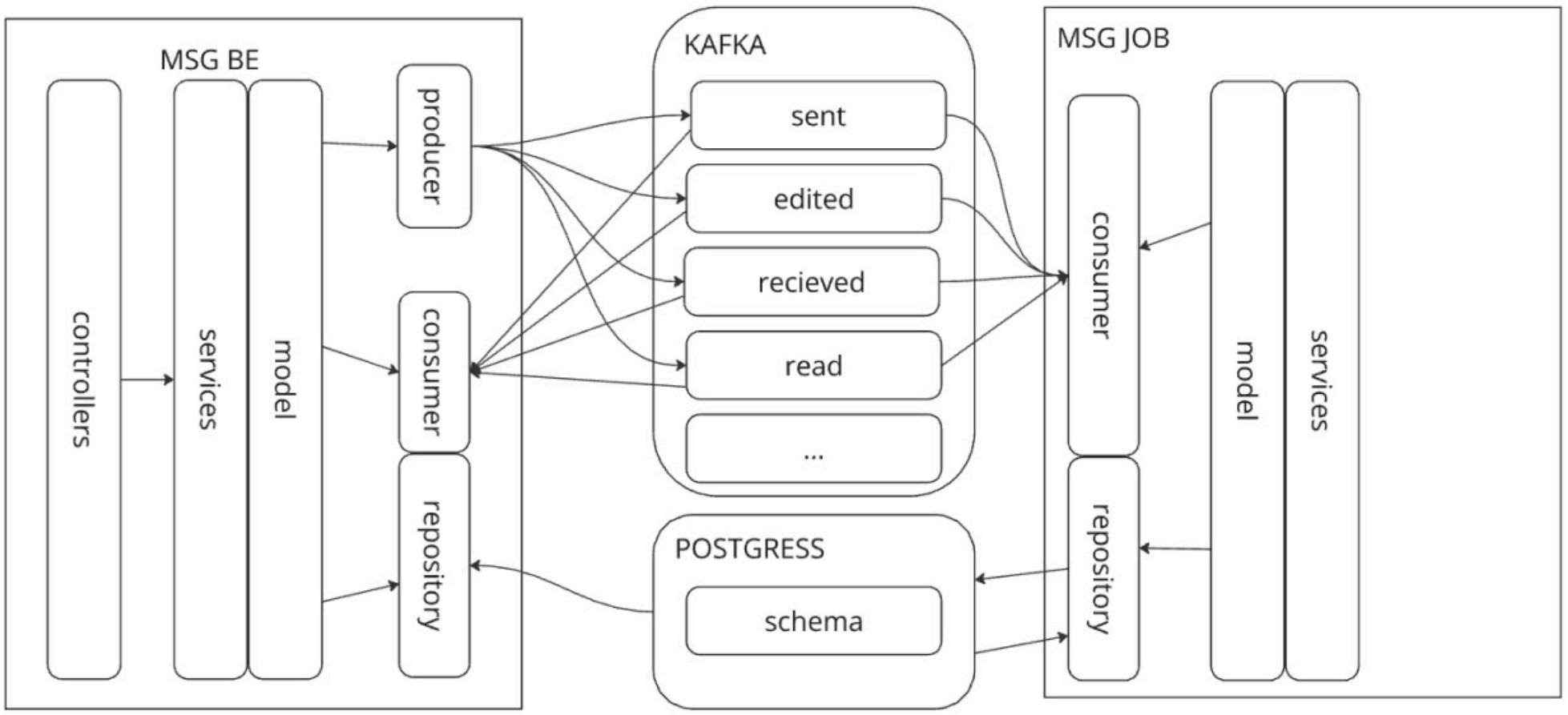


Component diagram:

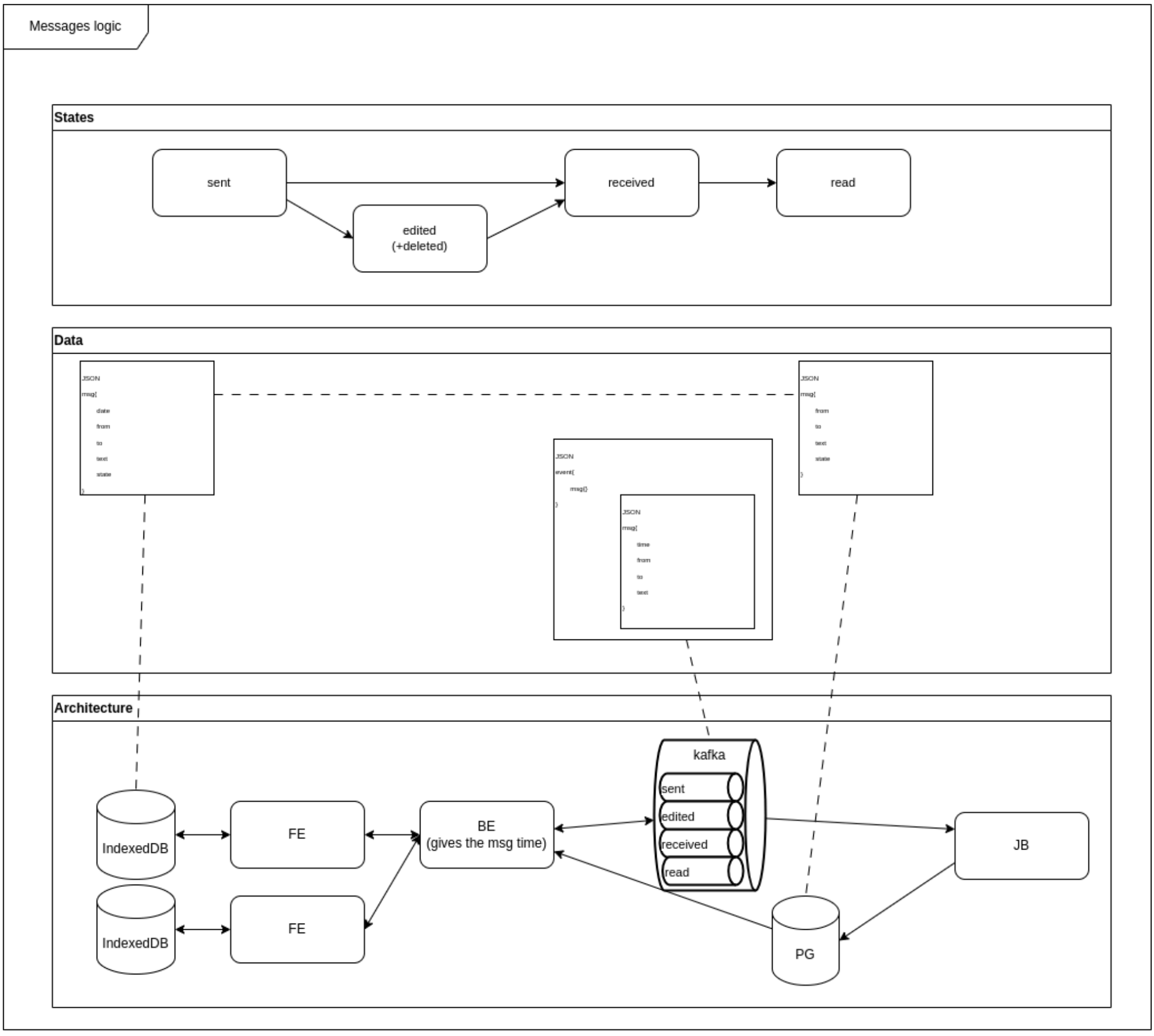


Drafts

Spring Kafka



Messages


Messages logic
States
sent
edited
(+deleted)
received
read
Data
JSON
msg{
date
from
to
text
state
}
JSON
event{
msg{}
}
JSON
msg{
time
from
to
text
}
JSON
msg{
from
to
text
state
}
Architecture
IndexedDB
IndexedDB
FE
FE
BE
(gives the msg time)
kafka
sent
edited
received
read
JB
PG

api

Last edited by **Andrew** 1 month ago

API

Paths	Method	Summary	Description	Parameters	Request Body	Responses
`/auth?...`	POST	Register auth record	if userService return userUUID	userHash	JSON `{ userUUID: null }`	200
`/auth?...`	GET	Login	if userName valid	userName, userHash		200
`/auth?...`	DEL	Logout				200
---	---	---	---	---	---	---
`/users?...`	POST	Create new user	if userUUID not exists		JSON `{ userUUID: null }`	200
`/users?...`	PUT	Update user	if authUserUUID valid	authUserUUID	JSON `{ userUUID }`	200
`/users?...`	DEL	Delete user	if authUserUUID valid	authUserUUID		200
`/users?...`	GET	Get user info by username	if username exists	authUserUUID, username		200
`/users/{userUUID}?...`	GET	Get user info by UUID	if userUUID exists	{userUUID}, authUserUUID		200
---	---	---	---	---	---	---
`/chats?...`	GET	Get list and chats info	if authUserUUID valid	authUserUUID, limit, offset		200
`/chats/private?...`	POST	Create new private chat	if userUUID exists	authUserUUID, userUUID		200
`/chats/private?...`	DEL	Delete authUserUUID from chat	if authUserUUID in private chat	authUserUUID, chatUUID		200
`/chats/groups?...`	GET	Get list of owned group	if authUserUUID owner	authUserUUID		200
`/chats/groups?...`	POST	Create group and chat	if authUserUUID exists	authUserUUID	JSON `{ chatUUID: null, ownerUUID}`	200
`/chats/groups/{chatUUID}?...`	GET	Get group info	if authUserUUID in chat	{chatUUID}, authUserUUID		200
`/chats/groups/{chatUUID}?...`	PUT	Update group	if authUserUUID owner	{chatUUID}, authUserUUID	JSON `{ chatUUID, chatUUID, ownerUUID}`	200

Paths	Method	Summary	Description	Parameters	Request Body	Responses
`/chats/groups/{chatUUID}?...`	DEL	Delete group and chat	if authUserUUID owner	{chatUUID}, authUserUUID		200
`/chats/groups/{chatUUID}/members?...`	POST	Add user to group	if authUserUUID owner and userUUID exists	{chatUUID}, authUserUUID, userUUID		200
`/chats/groups/{chatUUID}/members?...`	GET	Get list of owned group	if authUserUUID in chat	{chatUUID}, authUserUUID, limit, offset		200
`/chats/groups/{chatUUID}/members?...`	DEL	Delete the user from chat or leave	if authUserUUID owner or member	{chatUUID}, authUserUUID, userUUID		200
---	---	---	---	---	---	---
`/messages?...`	POST	Send message	if authUserUUID in chat	authUserUUID	JSON `{ messageId }`	200
`/messages?...`	PUT	Edit message	if authUserUUID in chat	authUserUUID	JSON `{ messageId }`	200
`/messages?...`	DEL	Delete message	if authUserUUID in chat	authUserUUID	JSON `{ messageId }`	200
`/messages?...`	GET	Get messages	if authUserUUID in chat	authUserUUID, offset, limit, chatUUIDs		200