Assignment 2 – Part 1

Designing an Instant Messaging Application

Bogazici University
SWE 530 – Software Design Process

Assist. Prof. H. Birkan YILMAZ

Prepared by: S. Erdem Turanlıoğlu
Due Date: 11.05.2022

Criteria:

The scope of the instant messaging application will be defined as follows. There are some initial constraints and knowledge for the base of the application creation.

Information about the system:

- Users may create a workspace by just providing an email and possibly a password for that (email, workspace) pair.

- Joining to a workspace should be done by invitation link.

- There should be channels in workspaces, where some of them can be protected, i.e., only some of the workspace users are allowed.

- Users should be able to send messages to channels that they are allowed to send

- Users in a channel should be able to create tasks with simple task management operations such as assigning task and viewing the status of the tasks.

- Users related to a task should be able to change the task status, where the status can be ToDo, InProgress, Completed, and Verified.

- Users should be able to see the tasks assigned to them or another user from the task management tab in a channel

The purpose is to provide a software system design, which will satisfy functional and nonfunctional requirements of “Instant Messaging Application”, will serve as a guideline for developers.

Important Note: There will be two implementation teams (in-house and external), so it is better to focus on a modular design. Please explain your strategy about this constraint.

How to design an instant messaging app -- assumptions:

Outlining key features:

- One-to-one chat or group chat in between users,
- Allowing users to send text, pictures, videos, and other files,
- Storing chat histories,
- Showing the status of users,
- Forwarding messages to other users,
- Notify users with the status of messages (sent/delivered/read)

Expected capacity of the application:

Message storage:

Estimating around 10 GB messages sent each day, with an average size of 100-150 bytes for each message, the system will need around 1.5 TB of capacity for daily messages traffic.

10 billion messages * 150 bytes = 1.5 terabytes

For the simplicity of the calculation, estimation assumes that there is no meta data and there will be a just a single copy of messages saved in the system. Though, in actual messaging applications the traffic is much more, and messages are replicated across multiple caches for scalability and reduced latency.

To extend the daily messages to a 10 year-basis: (we are assuming the storage will keep the posts for 10 years)

10 years * 365 days * = 1.5 TB = 5 petabytes

Status storage:

The system is not store only messages, but also stores status of the users as well. However, these are not needed to be stored in the database as messages, just keeping stored them in cache is fine.

Assuming the number of people will use the application is 1 million/month. But these are not active all the time, so the number can be reduced to half of it roughly: 500 thousand. And if each entry to the application takes 1 KB, the system will need 0.5 terabyte of memory spaces for a cache.

500 T * 1 KB = 0.5 terabytes

The main important feature of an instant messaging application is to send messages within heartbeats. The service is also sending timestamps to cache, which confirms the precedence of the user.

Updated on 11.5.22

Requirements Elicitation:

- How will be password set, will there be any constraints on password selection in the registration?
- After sharing of the space, will there be any approval period?
- With shared link, can person use that link more than once?
- With shared link, can person send the link to another person for to join the space?
- How will be the registration to the platform? What information would be needed?
- How many people can join in a particular space? Will there be any limitation?
- Will there be any different user type such as free, standard, and premium?
- What will be the difference between the user types?
- Will there be any direct messaging area for people in the space?
- Will sharing images be allowed?
- Will space support the voice channels?
- Will there be any type of notifications?
- How will notification work?
- Will there be any searching & filtering feature in the space and in the platform?
- When people will be notified?
- Can people set the notification settings as they desired?
- Can people send a message with tagging some user?
- Can platform permit same named spaces, or should there be any warning system for duplicate named spaces?
- What people should see in the task management tab?
- Are tasks can be editable by anyone?
- Will closed tasks be saved in the system?
- When should people create task and assign to someone?
- Can tasks assigned to more than one person?
- Will there be any tags to add on tasks for ease of searching?
- Will not related messages be deleted from the space?
- Can space owner block users from the space?
- How can people enter to the protected workspaces?
- Can people share documentations in the space?
- What can be the maximum size of the documentation to share?

Constraints:

- Documentation sharing is restricted according to subscription types.
- Number of members in spaces is restricted according to subscription types.
- Only users can edit their content, not other users can change them.
- Added documentations cannot delete from other users, just person who added and space owner can delete it.
- Documentations before uploaded, administrators should be controlled.
- Space invitations are set with one time use unless the space owner set with different settings.

As indicated in the assignment, the implication of the system will be done teams by internal and external. Thus, the design will be modular.

Requirements Analysis:

- The platform shall support web and mobile applications.
- The instant messaging application shall have an easy registration system.
- The application shall have a minimum 6 characterized password with upper- and lower-case characters.
- The application shall have space registration system, which can be done by shared links from space owner.
- The space owner shall share links to invite people into the space.
- The space owner shall block users from the space if the user has violated the rules.
- The space shall have user limitation regarding on subscription type: free users may have 10 persons in their room, premium users may have up to 100 persons.
- The application shall permit users to share documentations in the space.
- The application shall permit free users to share with 5GB maximum size of the documentations.
- The application shall permit premium users to share with 20GB maximum size of the documentations.
- The space owner shall choose the type of group invitation method, the shareable link should be in one time use, or the link may permit to just verified user e-mails.
- The application shall permit the image sharing in the space.
- The application shall have a direct messaging area between users in the same space.
- The application shall have a notification system to notify people when some tasks assigned to them, or some new questions raised in the space.
- The application shall have a task management system to see which tasks are completed, verified, in-progress or under to-do buckets.
- The application shall notify people by sending e-mail or sending messages to their mobile phones.
- The application shall restrict the editing tasks from different users.
- The application shall restrict the editing messages in the spaces.
- The application shall permit to answer some question or comment in the space by tagging the related person.
- The application shall have tags in comments or tasks.
- The application shall enable users to have search in spaces.
- The application shall enable users to search some content in the task management page.
- The application shall enable users to create different channels under the space.
- The application shall have administrators to check that there is not a harassment in the workspaces.
- The application shall have administrators to check that there is not a spam user in the workspaces.

Modular design

Modular design can be explained basically a decomposed small module of complex systems to have more efficient organizations of designs and processes.

Advantages:

- Allows each module to written separately without having to know architecture or code blocks of other modules
- Allows faster development, due to parts have separated into small parts across team members to handle any problem
- Allows flexibility
- Individual models are highly reusable in different projects
- Bugs are less frequent and easy to detect

Disadvantages:

- Requires documentation for future works
- Designed to use with large systems rather than small systems
- More effective when several people contributed to project
- Program complexity has increased with different types of architectures can be found

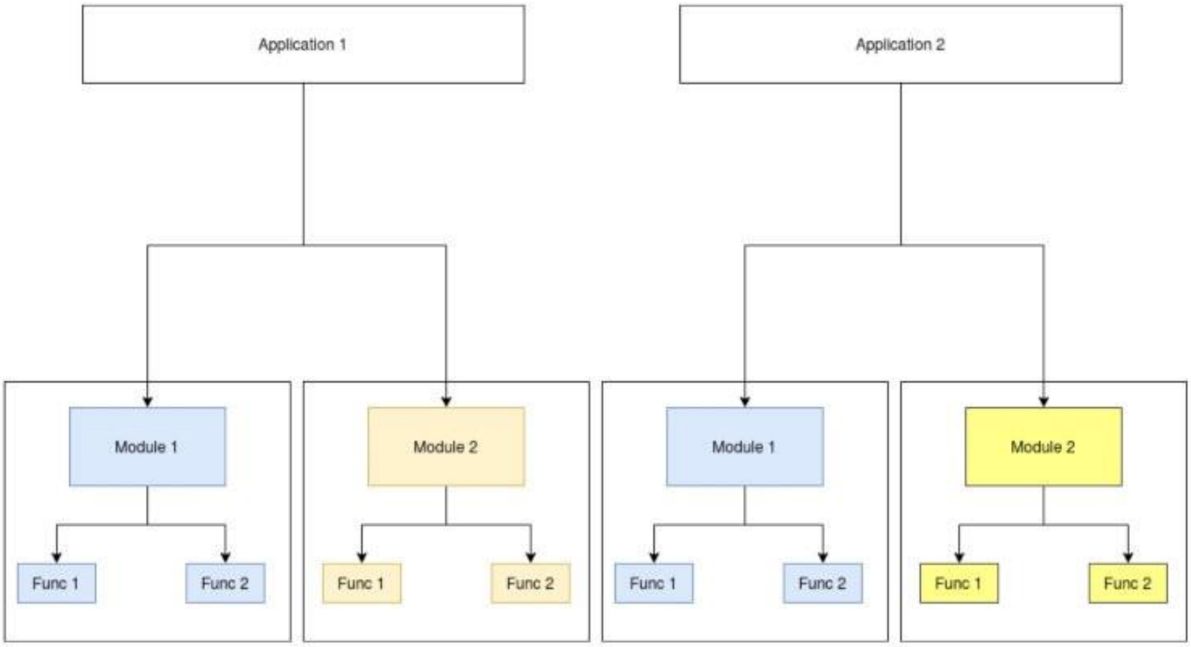
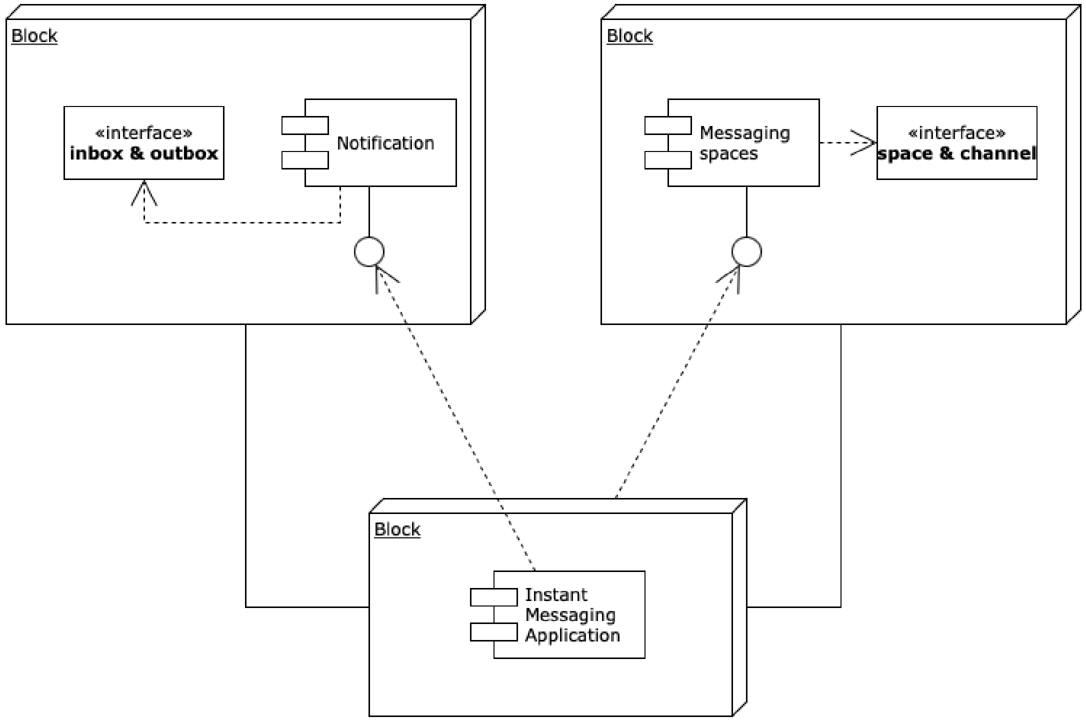


Figure 1 - Example of a modular design
*Retrieved from “https://miro.medium.com/max/1400/0*iJTwe1UxDx4VV020.jpg”*

High-level design:



As explained in the previous page, modular design concerns with small parts of a large project. In assignment 2, our project aim is to design an “Instant messaging application”, which will be implemented by two teams. According to that, the project divided into two different subgroups:

1. Space & channel:

One of the teams will work on the architecture of the spaces & channels, which they will create an area for people to discuss, share information between them.

2. Inbox & outbox:

The other team will work mainly on the notification part of the application. One of the important features of the application is the notification, because this feature connects people with notifying people about their contributions in the system.

Constraints:

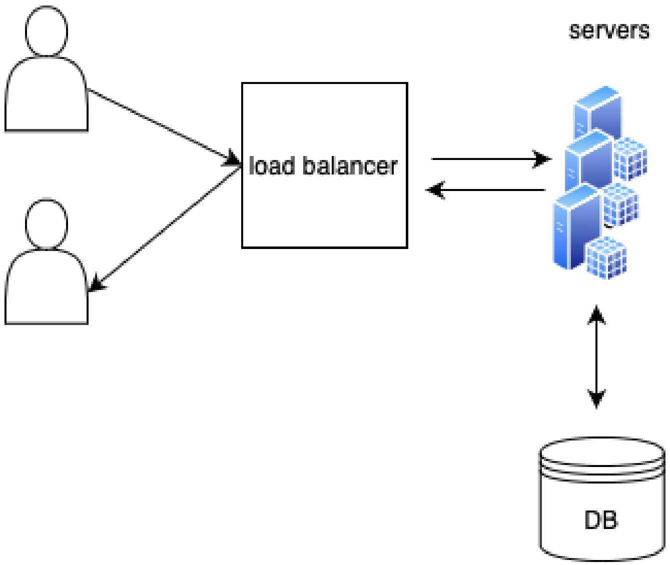
- Design teams should prepare a documentation after they implemented a new feature into the application.
- Complexity should be used when it is necessary, beside that code comments should be applied for future needs.
- Systems that design teams will work on it should be large enough to divide into subcategories.
- There should be communication between teams, because these two modules should be work collaboratively.

Basic architecture:

When we thought about messaging application, the most primitive way to describe it with the communication in between two people.

In the ideal scenario, A knows B and B knows A. Thus, they can send information to each other via application.

High-level architecture – message sending and receiving:



In a bigger community, close to the real-world scenario, two persons may not have any information on their IP addresses, thus it is also not feasible.

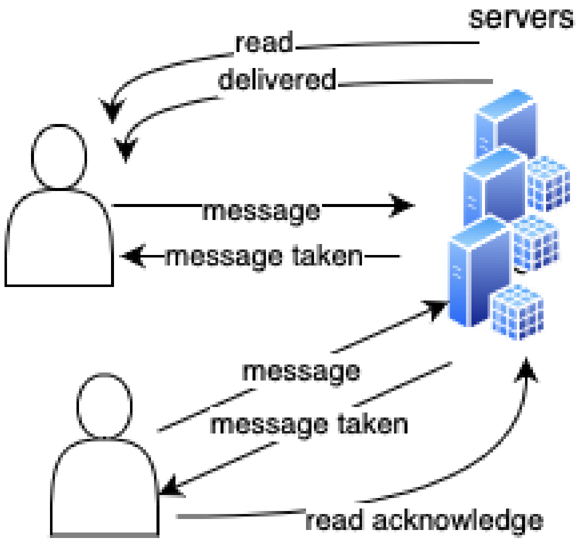
In case of a direct communication between users, there must be a server in between to manage all messaging traffic. In scaled systems like in the figure, there are multiple servers, which is part of the messaging system.

A load balancer: the main task of the object is to send the messages regarding to server's status. If the server is free and the capacity is okay, then load balancer sends the information to sustain a communication in between people.

Servers are also playing an active role, which sends to DB to store the messages.

Updated on 11.05.2022

High-level architecture –notifications:



Other important feature is expected from the messaging application is to see the notifications. Notifications can be categorized into two:

- Notifying person with sending e-mail or mobile messages
- Notifying with sent, delivered, read signs.

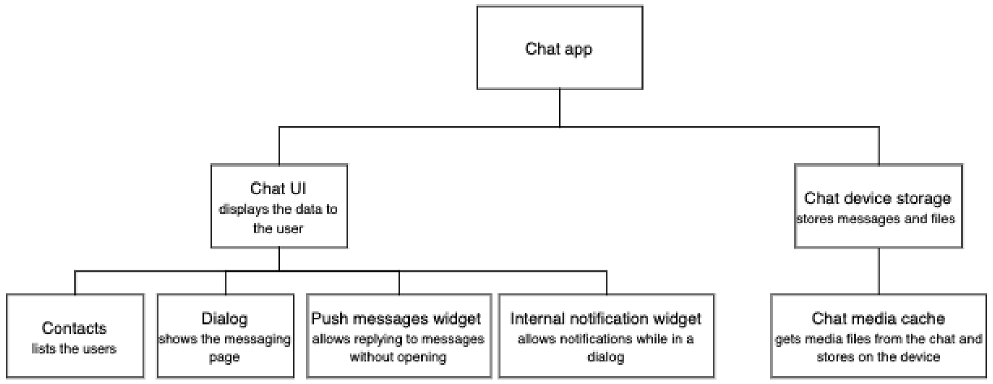
The aim is to give basis of second kind of notifications in the architecture, which is figured on left.

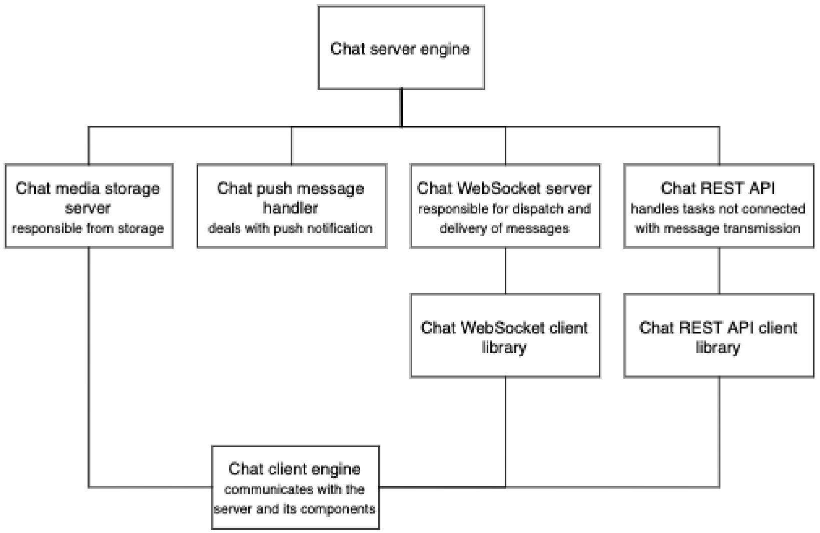
When a user sent a message, the system should return to user, that message has been sent, delivered, and read by the receiver.

1. User A sends a message to user B.
2. Servers return a message of acknowledgement that the message is taken by server.
3. When user B receives the message, the acknowledge sent to the server that shows there is no problem on sending the message.
4. After that, sever sends a message to user A that explains that the user B has got the message. This step is also indicated as “delivered”.
5. As soon as user B opens the message, the system sends acknowledgment to the server. Then the server sends “read” notification to the user A.

Updated on 11.5.2022

MVP design:


Chat app
Chat UI
displays the data to the user
Chat device storage
stores messages and files
Contacts
lists the users
Dialog
shows the messaging page
Push messages widget
allows replying to messages without opening
Internal notification widget
allows notifications while in a dialog
Chat media cache
gets media files from the chat and stores on the device


Chat server engine
Chat media storage server
responsible from storage
Chat push message handler
deals with push notification
Chat WebSocket server
responsible for dispatch and delivery of messages
Chat REST API
handles tasks not connected with message transmission
Chat WebSocket client library
Chat REST API client library
Chat client engine
communicates with the server and its components

Chat Client:

The client is responsible for interacting with the operation system. Interactions includes:

- Sending push notifications,
- Displaying data to the user,
- And storing messages and files.

When the user types a message and sends it, the chat client transmits that message to the chat server.

Chat Server:

Server is responsible for securely receiving message, identifying correct recipient, queuing message, and then forwarding the message to the recipient's chat client.
Chat server's resources can include REST API, a WebSocket server, and an ASW for a media storage etc.

Updated on 11.5.22

Chat REST API:

It is often used to facilitate the functionality of the application outside of messaging.
For instance, authentication of users, user profile settings and notification settings are managed through REST API.

WebSocket Server:

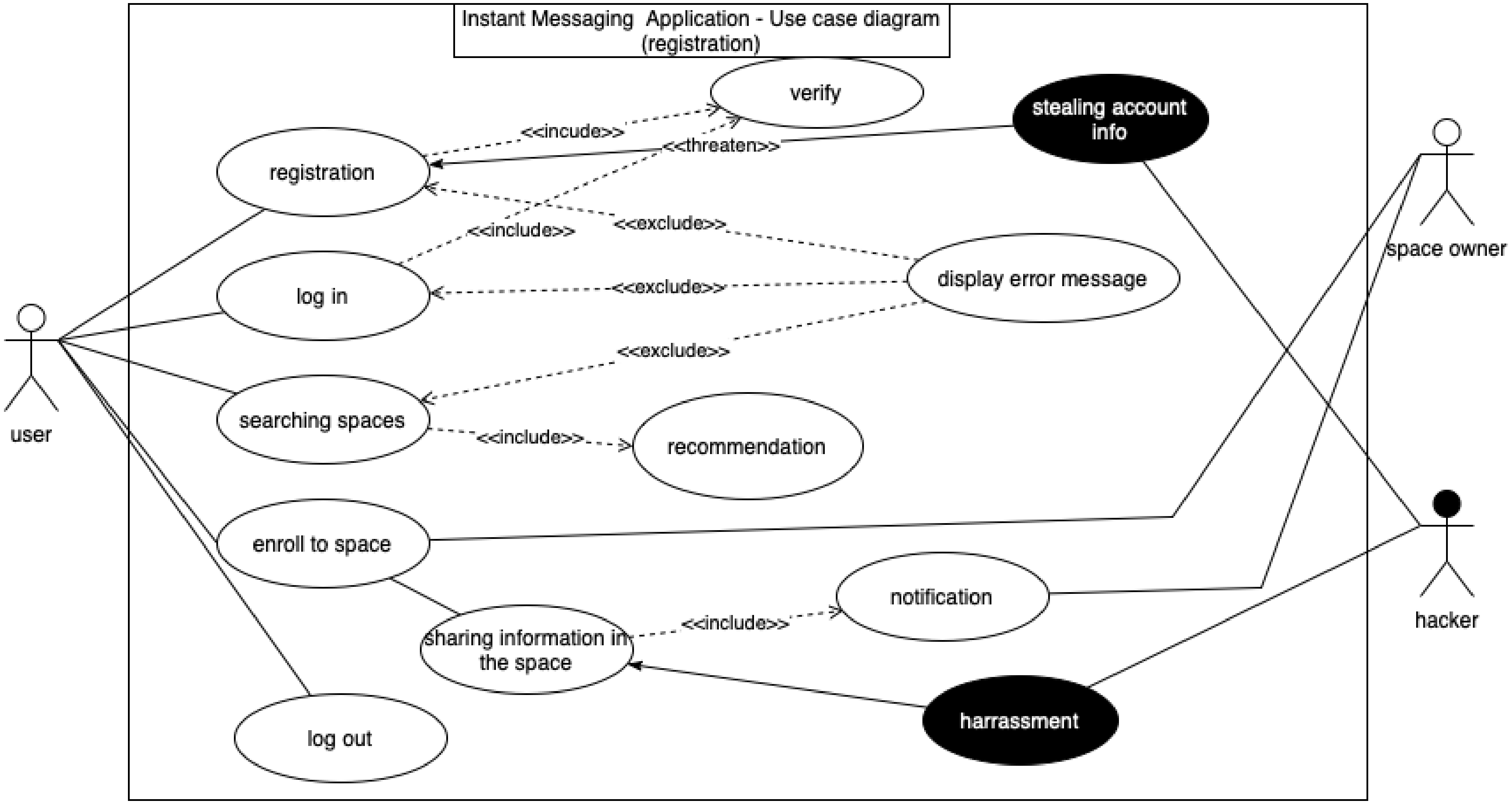
The typical communication between user and server is done using HTTP and it requires that the user makes a request for data from server. The server itself cannot push data to the user without the user first making a request.

In the messaging application, users every new message client should start a poll request to the server, which can create inefficiencies. Thus, a WebSocket supplies persistent connection between users and server that provides bidirectional communication path. That means, server can send data to the user without first getting a request.

Media Storage:

In the messaging application, it is also essential have a media storage as well as data storage.

Use-case diagram (registration)



Registration: Users, who may want to use the instant messaging application, should register to the system. They need to specify a unique username and password for registration, and they are required to enter their name, surname, and e-mail information.

Log-in: Users may sign into the system by entering their unique username and password, which are set in the registration phase.

Log-out: Users, who have been signed may sign out by clicking sign out button.

Search: Users may search the information that seek in the chat area. Also, they can search person whom they want to talk in the chatting page.

Enroll to the space: Space owner may share the link to add users into the space that is created. The link invitation is valid for just one use unless the space owner changes the type of the invitation link. Other link types are listed as link created for everyone can use to enter or link created for unique e-mail addresses.

Sharing information: Users may share documents in the spaces. The sharing constraint is related with the user types: free users may only share up to 5GB and premium users can share max 20GB files.

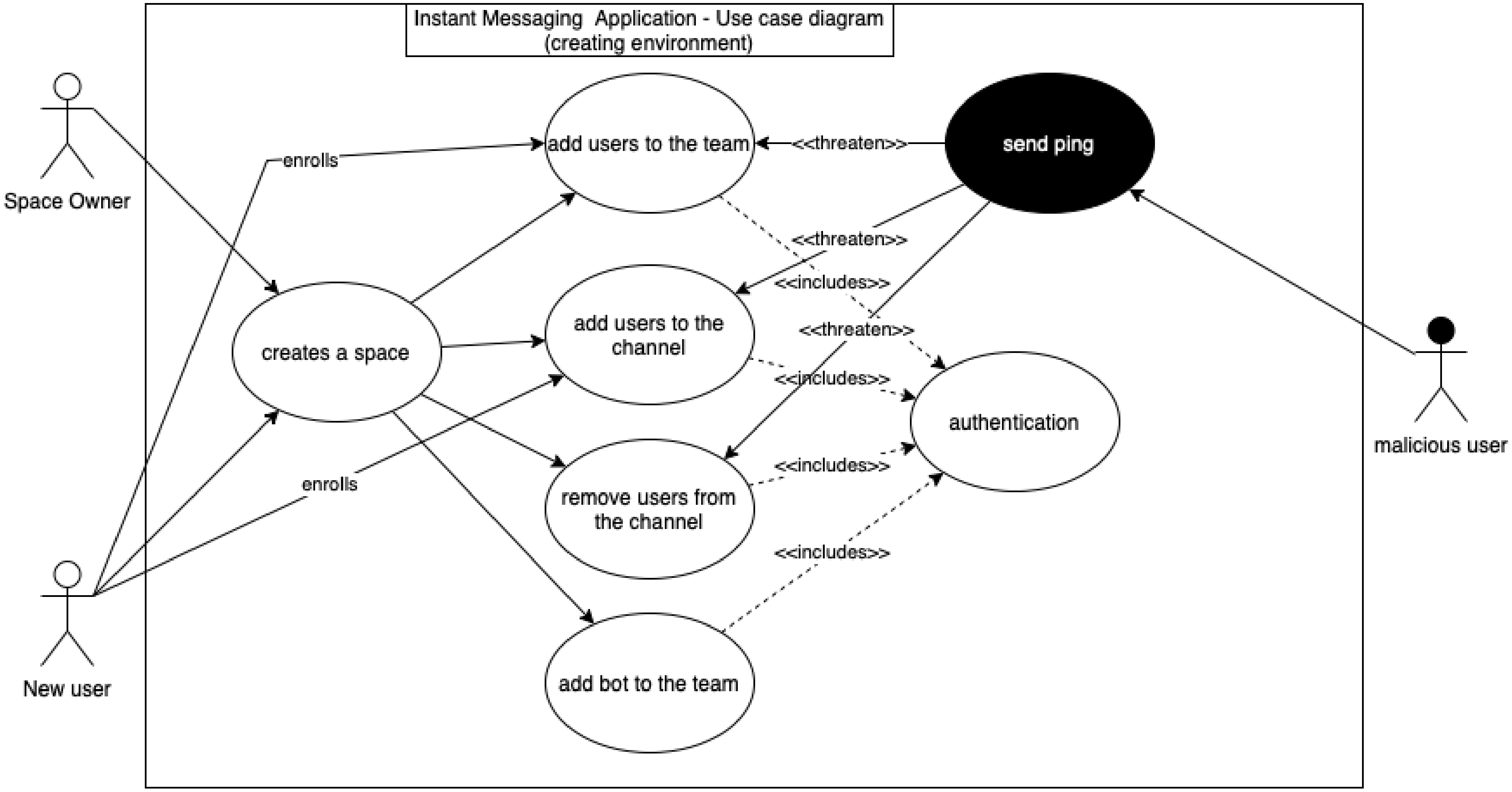
Recommendation: The instant messaging application has a feature that recommends comments in feed according to the user interests.

Notification: The instant messaging application notifies users when there is new message sent to the space, or someone new joined to the space. Notifications can be set according to user preferences.

Harassment: Users may harass other users with trash information or disturbing with messages.

Stealing account info: Hackers may intend to steal account information, which holds user’s e-mail address. This can cause a problem for the reliability of the application.

Use-case diagram (creating environment)



Creating channels: Users may create spaces to discuss and share information with each other. Users can create spaces, while they are in the main page. When created a space, system asks some information to create such as space name, space description.

Inviting users to the channels: Users may add new users to previously created spaces. For adding, space owner and space users can send an invitation link to the users. The invitation link is one time use unless the link created with different settings.

Inviting users to the channel: Users may be added to the created channels under spaces. Channel owner can directly add to the channels. Also, space users can join to the channels if there are no restrictions.

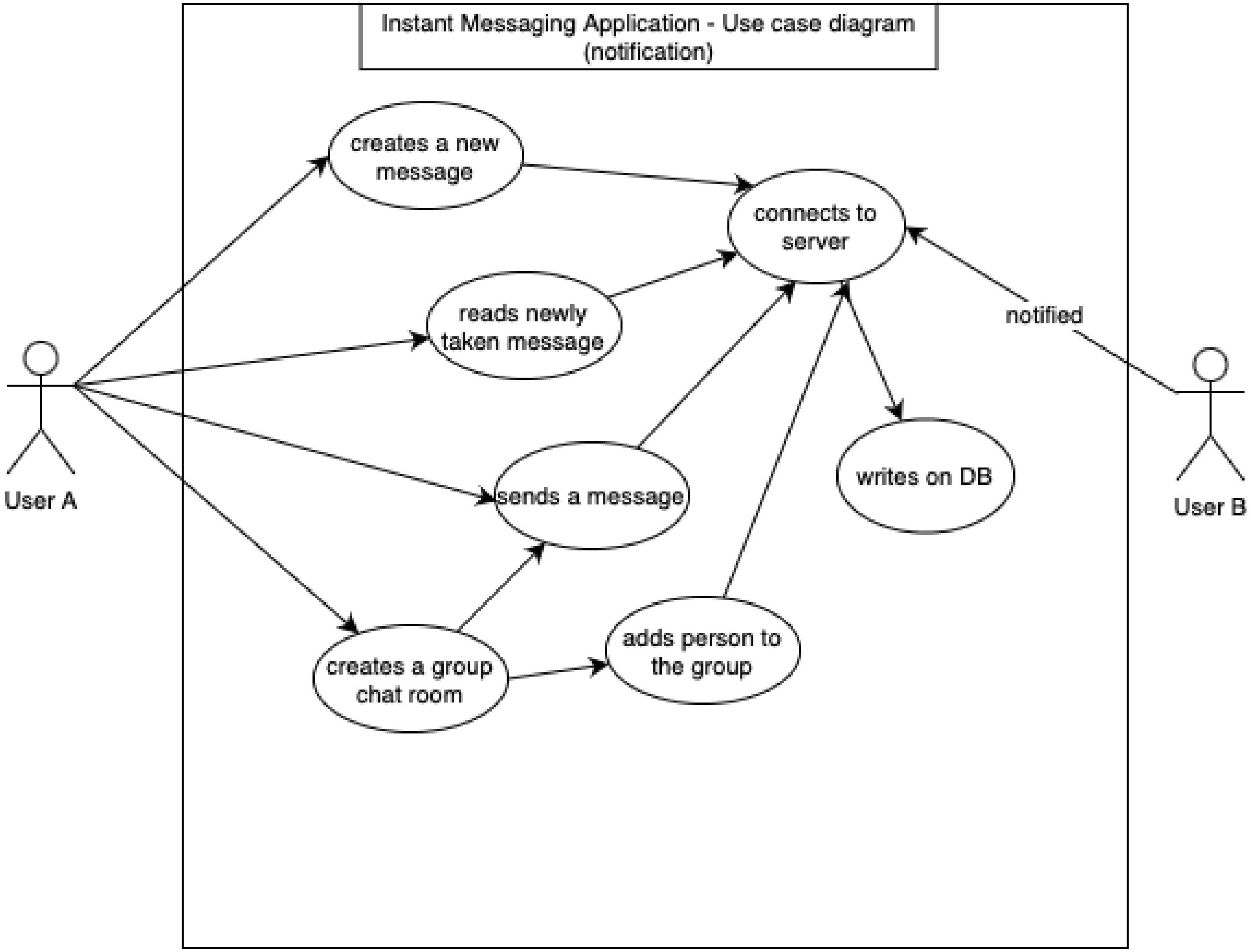
Additionally, channel owners can add people from outside with an invitation link. However, added user firstly added to the space.

Remove users from the channel: Users may want to remove users related to use of purpose of the channel may change.

Authentication: System does authentication for all use cases above.

Send ping: Malicious users may want to ping the space for creating problems for users in the spaces.

Use-case diagram (notification)


Instant Messaging Application - Use case diagram (notification)
creates a new message
connects to server
reads newly taken message
notified
writes on DB
sends a message
creates a group chat room
adds person to the group
User A
User B

Updated on 12.5.2022

When a user sends a new message, the receiver should be informed by system notification feature. The aim is to inform the receiver that he/she has a new information, that its should be checked.

Notifications can be grouped into two:

- User-generated notifications
- Context-generated notifications
- System-generated notifications

For the simplicity, I developed a use case is based on first two types of notifications. The basic idea behind it when a user creates a message in the application, the receiver gets notification from the application to indicate the new unread message is waiting.

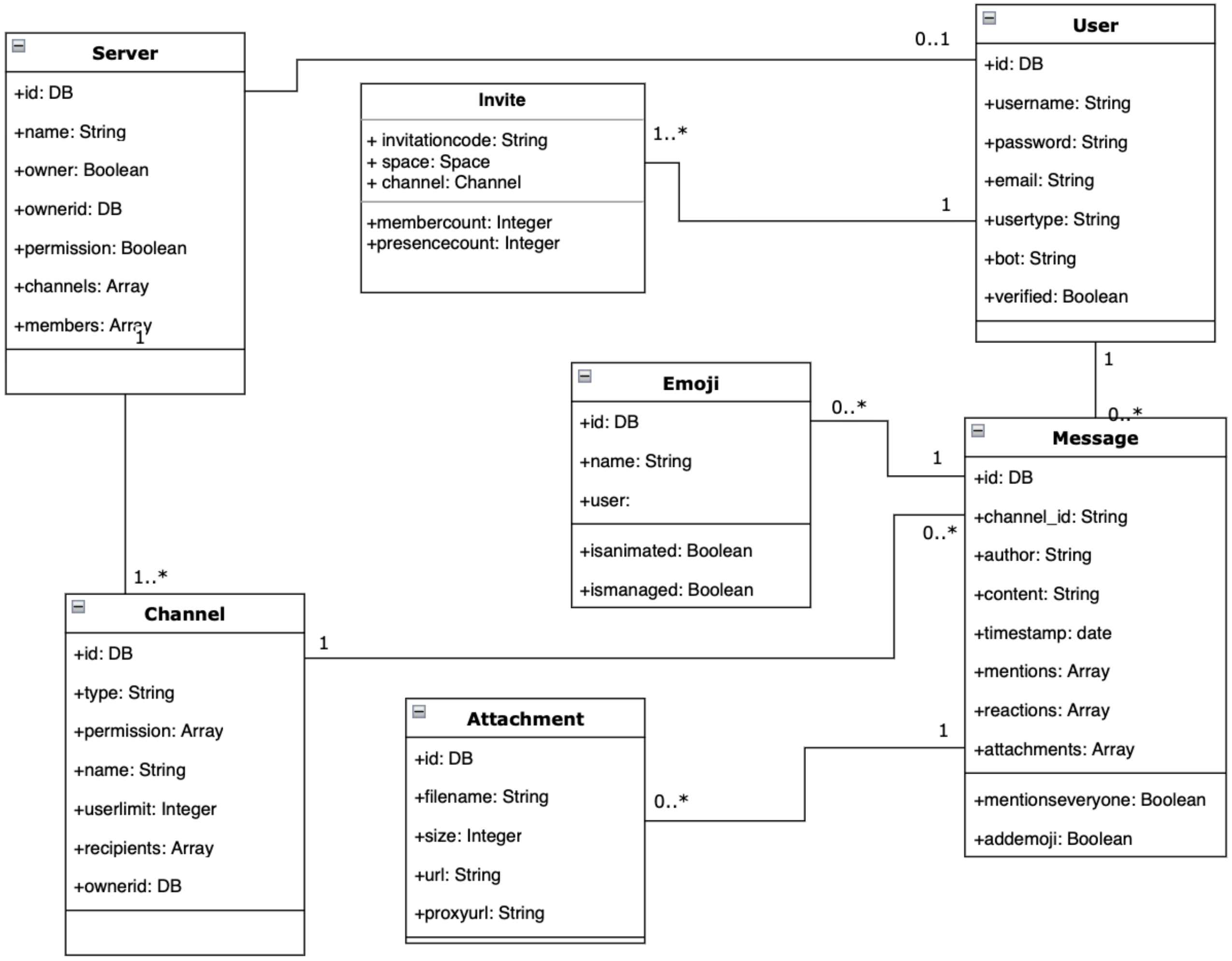
The second use case is generally about reading the message. When a user -who gets a new message- opens a message, the system gets a signal that the user has opened the message and read. The change on the status on the message directly send to the sender that his/her message is read by receiver.

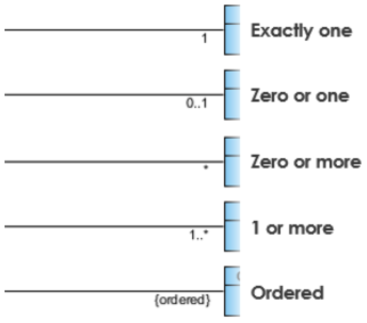
Of course, examples can be increased. As:

- The system can send advertisement notifications
- The system can send version update warning notifications
- The system can notify the user regarding their friend is online or offline etc.

Updated on 12.5.2022

Class Diagram





User class: This class holds information of registered user, such as username, password, email etc.

Message class: This class holds information of messages that sent or received, or where it sent or received. Mainly class holds channel id where it had been sent, author of the message, content of the message etc. Also, it consists of some functions a message can consists of such as emoji and mention.

Attachment class: This class holds information of attachment in sent/received messages, such as filename, size etc.

Channel class: This class holds information of channel under created spaces, such as name of the channel, user limit of the channel etc.

Invite class: This class holds information of invite request to add a specific channel, such as invitation code, space name. It also consists of functions of count of member that invitation link sent.

Server class: This class mainly a bridge in between user and channel for working of the instant messaging application.

Also, the notifications are controlled from the server itself. Main notifications are:

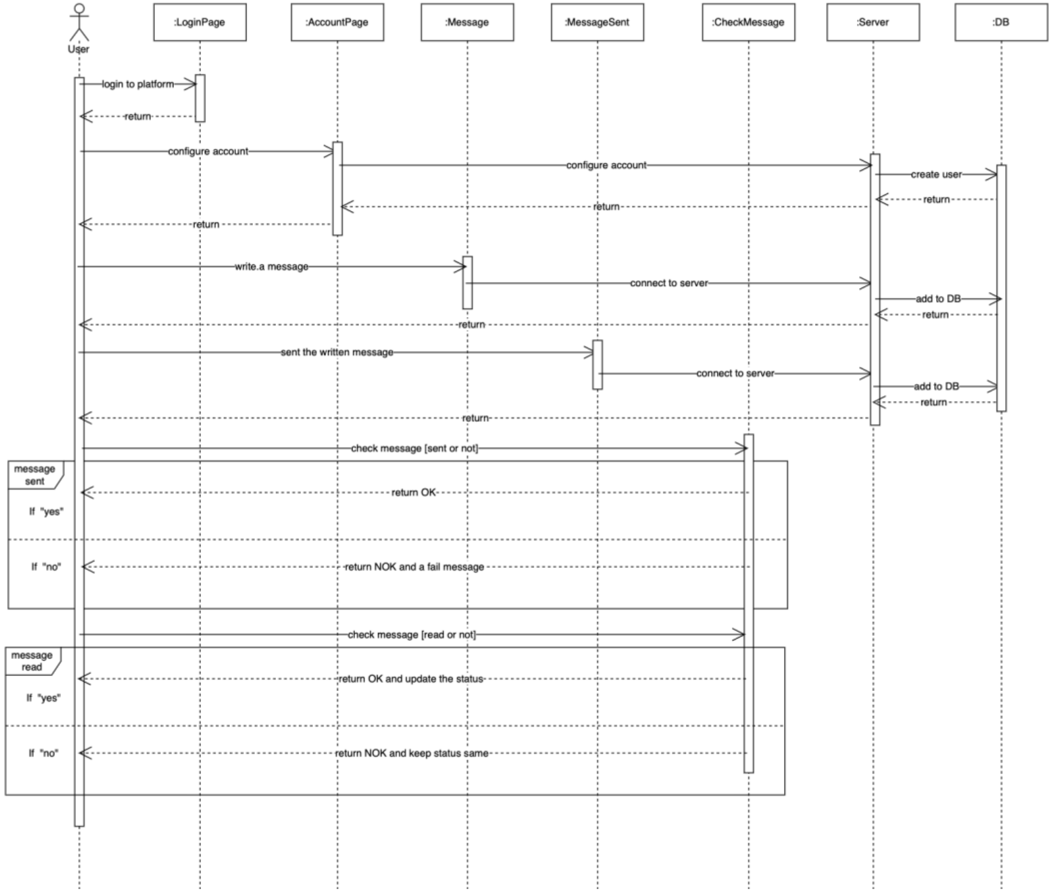
- Sent
- Acknowledged
- Delivered
- Read

Beside them, there is also system notifications to notify people with the requirements of version update, or any other kind of maintenance activities.

Updated on 12.5.2022

Sequence Diagram

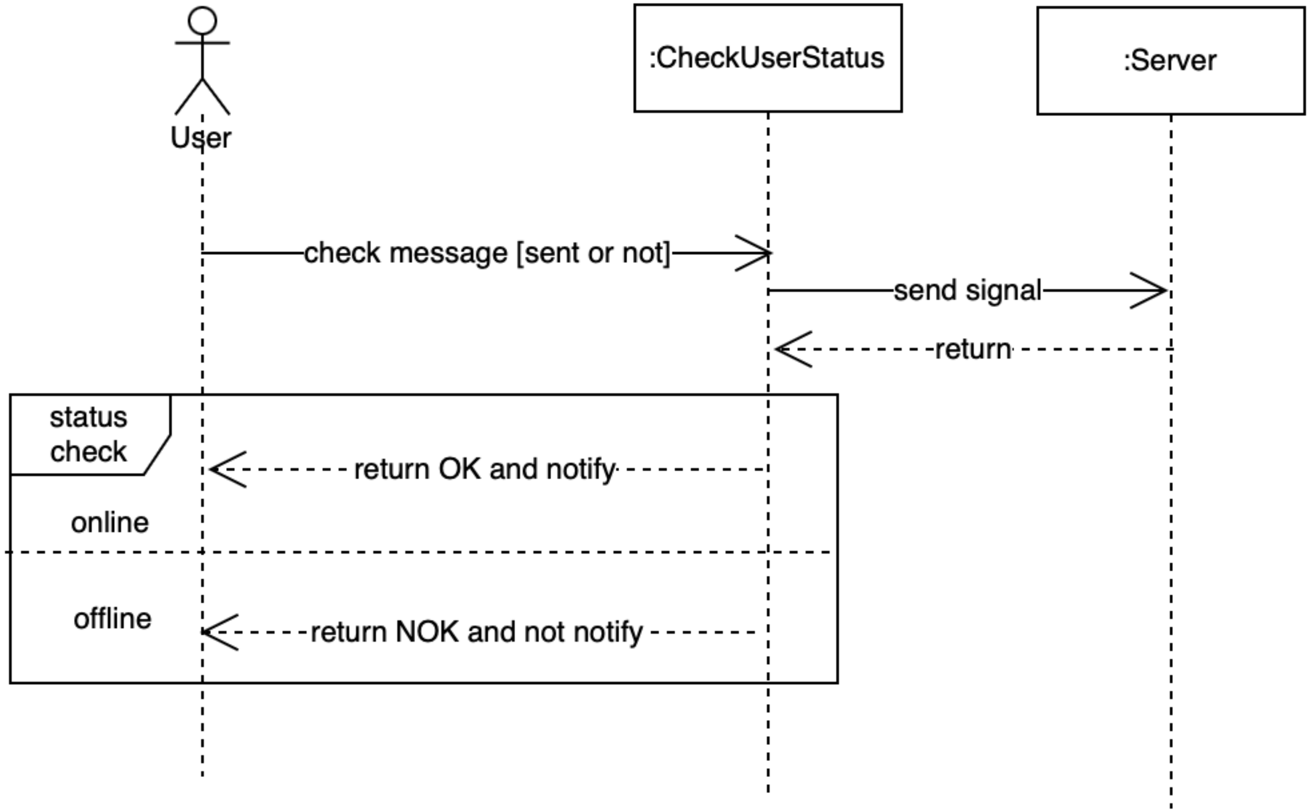
- Login
- Account configuration
- Creating message
- Delivery report
- Read report



Updated on 12.5.2022

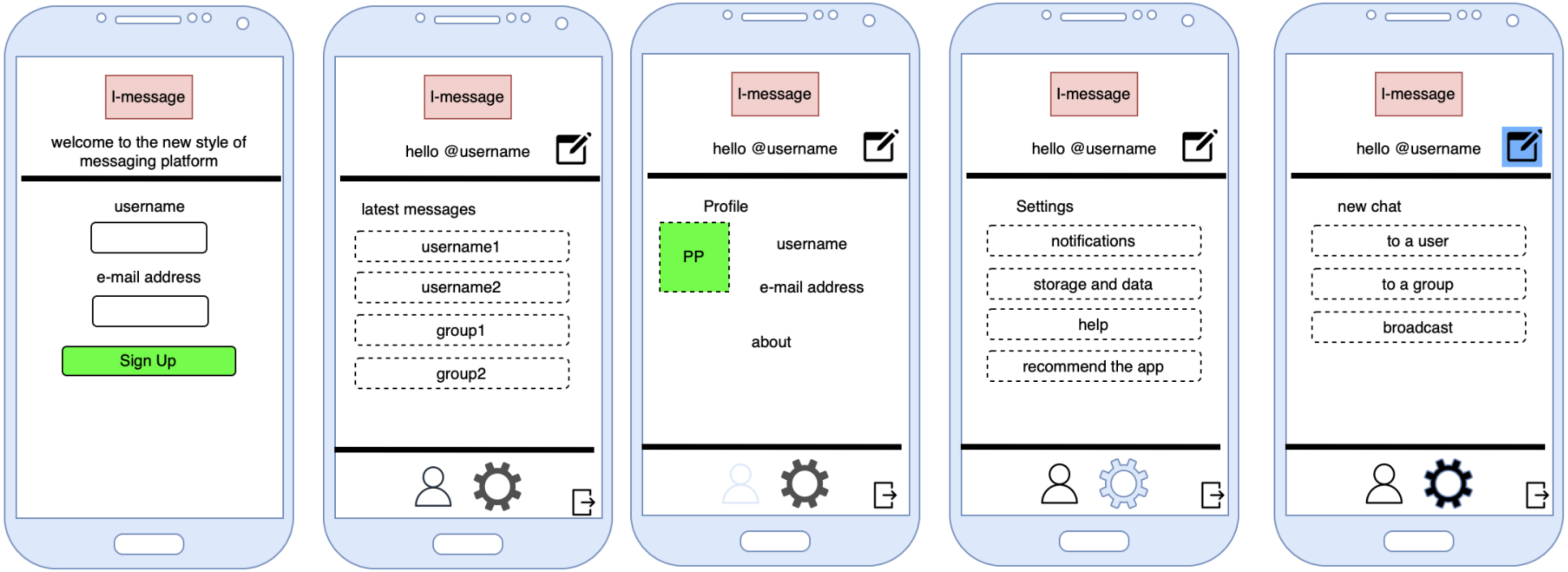
Sequence Diagram

- Check status of user



Updated on 12.5.2022

Mock-up design of the app



Updated on **12.05.2022**