Distributed Computing: Documentation

Zhong-Xi Lu, Angela Mizero, Thomas Van Bogaert

1 Microservice Oriented Architecture

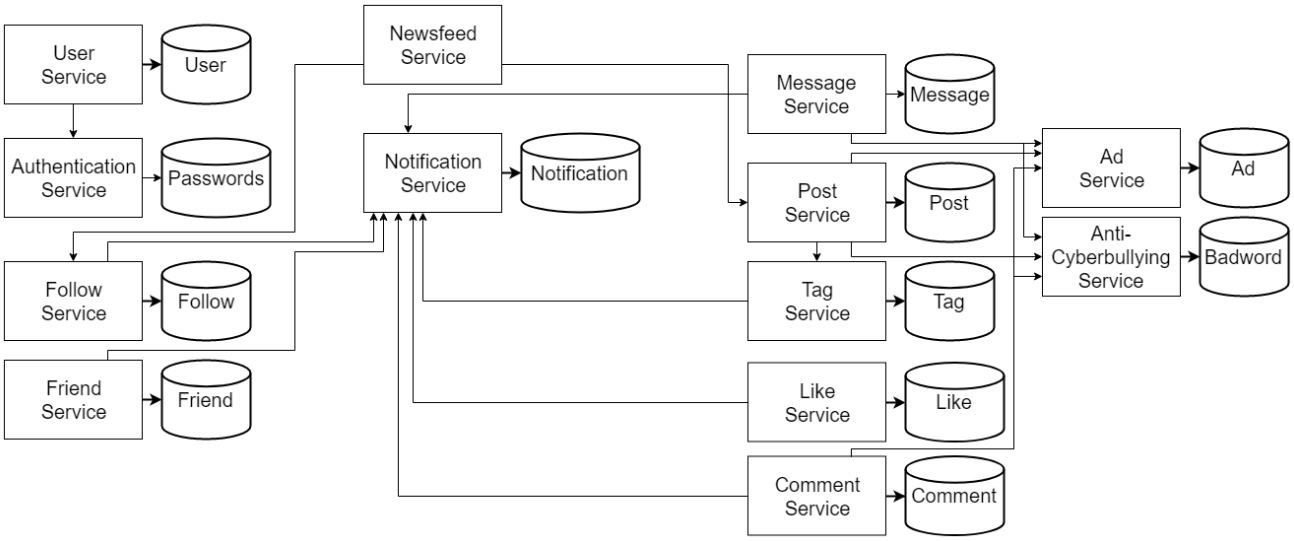

Figure 1: Architecture of core microservices with their interaction

The core microservices that form the backend of our system can be found in figure 1. The interactions, i.e. dependencies, between the different microservices are shown as well. Most of these services also have a corresponding database, in the actual implementation the database is ran on a separate service. More details on these services can be found in section 2.

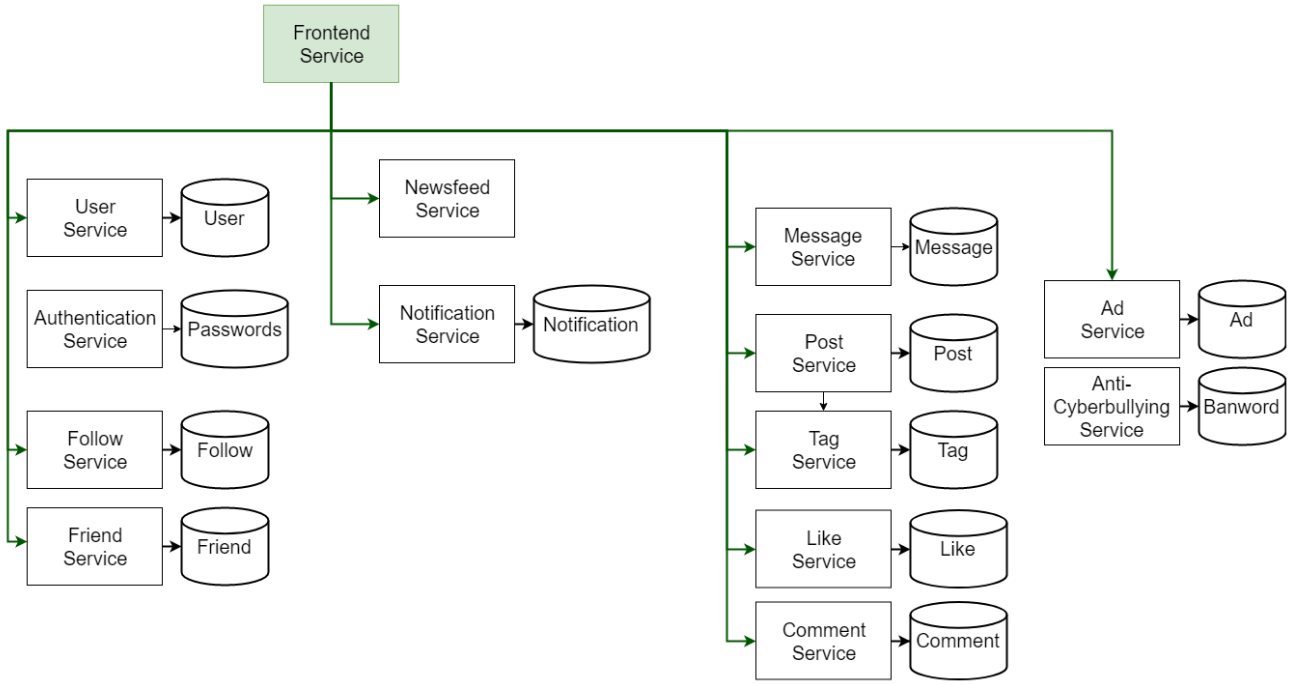

Figure 2: Architecture of core microservices plus frontend with their interaction

On top of all these components, we defined a specific service for the frontend as well (see figure 2). This service is only there to offer a view for our application and will therefor communicate with our backend by making use of the provided api. As a side note, (almost) each request that is sent from the frontend implicitly goes through the authentication service to verify the user credentials that was stored in a token.

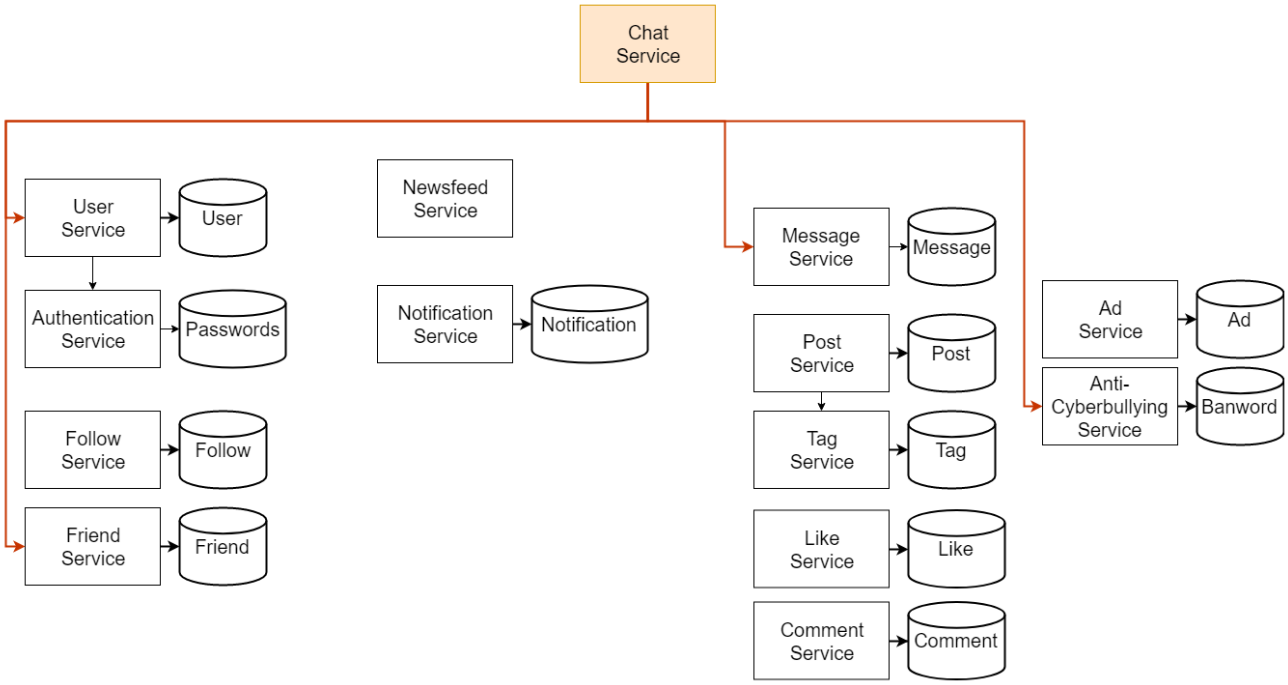

Figure 3: Architecture of core microservices plus chat service with their interaction

Similarly to the frontend service, there exists a standalone chat application as well. This application is as minimalistic as possible while still providing the basic functionality to login/register and sending messages to your friends. Of course, it uses the same backend as the frontend service. This will allow cross-communication between users from the chat and main application, since they're using data from the same services.

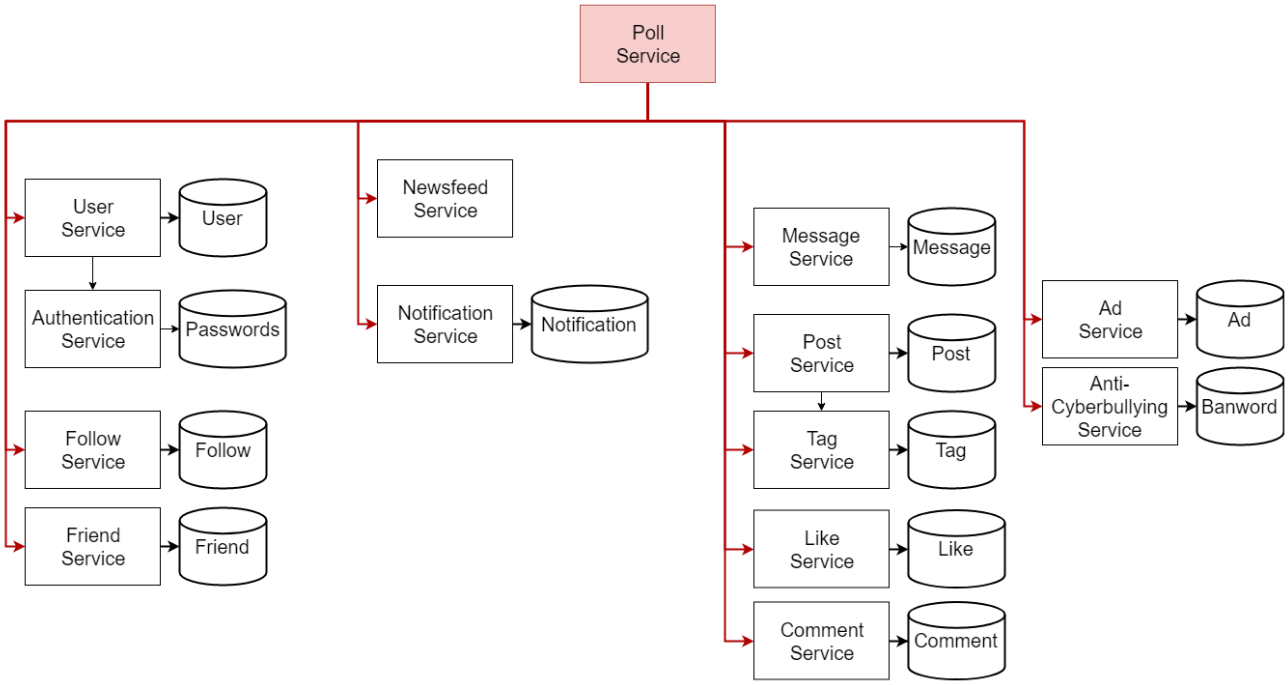

Figure 4: Architecture of core microservices plus poll service with their interaction

Lastly, we added a separate poll service as well that has the responsibility to poll (send ping requests to) every service (except user, authentication and notification) and if it detects that a certain service is down for some reason, it will then create a notification for all users respectively.

2 API Services

This section will go more into these core microservices by precisely defining their REST API. Note that most of these services also have a ping url.

User service:

In: POST users (email, username, password)

Out: /

Logic: Registers or creates a user

In: POST users/login (username, password)

Out: Response indicating whether the login was successful or not (and return some token if it was successful)

Logic: Tries to login a user

In: GET users

Out: users (user_id, email, username)

Logic: Gets all the users

In: GET users/user/username

Out: users (user_id, email, username)

Logic: Gets a user by username

In: GET users/user_id

Out: user (user_id, email, username)

Logic: Get a particular user

In: DELETE users/user_id

Out: /

Logic: Deletes a user

In: PUT users/user_id/block

Out: /

Logic: Blocks a user

In: PUT users/user_id/unblock

Out: /

Logic: Unblocks a user

Authentication service:

In: POST passwords (user_id, password, is_admin)

Out: /

Logic: Adds user and password pair to the database to verify tokens in the future

In: PUT passwords (password)

Out: /

Logic: Updates a user password

In: DELETE passwords

Out: /

Logic: Remove password from the database

In: PUT passwords is_admin

Out: Flag indicating whether the user is an admin or not

Logic: Check if the user is an admin

In: GET verify_credentials (token)

Out: Flag indicating whether the authentication was successful

Logic: Tries to authenticate a user by checking their credentials

Message service:

In: POST message (contents, sender_id, receiver_id)

Out: /

Logic: Add a new message to DB

In: GET messages/receiver_id/sender_id/<amount>

Out: Messages (content, timestamp, id)

Logic: Get messages in conversation between receiver and sender sorted by timestamp

In: GET messages/user_id/unread

Out: Messages (content, timestamp, id)

Logic: Get all unread messages where the user is the receiver

Notification service:

In: POST notifications (content, recipients)

Out: /

Logic: Add notification to DB

In: GET notifications/notification_id

Out: Notification (ID, timestamp, content)

Logic: Get a specific notification

In: PUT notifications/notification_id (user_id)

Out: /

Logic: Mark notification as read

In: GET notifications/user/user_id

Out: Notifications (ID, timestamp, content)

Logic: Get all unread notifications from a specific user

Ad service:

In: POST ads (image, categories)

Out: /

Logic: Add advertisement image to DB (note that the images are actually saved locally instead of in the database; only the image path is stored)

In: GET ads

Out: /

Logic: Homepage for the ads service (only accessible for admins)

In: GET ads/filename

Out: Advertisement (image link, category)

Logic: Get a specific advertisement based on the filename

In: POST ads/user/user_id

Out: /

Logic: Update (if necessary) the categories of a user (personalized ads)

In: GET ads/users/user_id

Out: Advertisements (image link, category)

Logic: Get ads specifically targeted to a user if any

Anti-Cyberbullying service:

In: POST anti_cyberbullying (words)

Out: /

Logic: Add new "bad words" to DB

In: GET anti_cyberbullying

Out: /

Logic: Homepage for the anti_cyberbullying service (only accessible for admins)

In: GET anti_cyberbullying/bad_words

Out: All the bad words

Logic: Get all the bad words from the database

In: POST anti_cyberbullying/contains_bad_word (sentence)

Out: True if the provided sentence contains one or more bad words, false otherwise

Logic: Check if there's a bad word in the sentence

Follow service:

In: POST follow (follower_id, followee_id)

Out: /

Logic: User follows another user

In: DELETE follow/follower_id/followee_id

Out: /

Logic: Unfollow a user

In: GET followees/follower_id

Out: Follows (follower_id, followee_id)

Logic: Get all users the follower follows

In: GET followers/followee_id

Out: Follows (follower_id, followee_id)

Logic: Get all followers of the user

Friend service:

In: POST friend/request (friend_initiator_id , friend_acceptor_id)

Out: /

Logic: Create a friend request

In: PUT friend/accept (friend_initiator_id , friend_acceptor_id)

Out: /

Logic: Accept a friend request

In: DELETE friend/friend1_id/friend2_id

Out: /

Logic: Unfriend a user

In: GET friend/user_id/requests

Out: Requests (friend_id)

Logic: Get all open friends requests sent to the user

In: GET friend/user_id

Out: Friends (friend_id)

Logic: Gets the friends of a user

Newsfeed service:

In: GET newsfeed/user_id

Out: Posts

Logic: Get the newsfeed of a user (posts of the people the user is following + own posts)

Post service:

In: POST posts (creator, content, tags (usernames))

Out: /

Logic: Create a new post

In: GET posts/user/user_id

Out: Posts (post_id, creator, timestamp, content)

Logic: Get all the posts from a user

In: GET posts/post_id
Out: Post (post_id, creator, timestamp, content)
Logic: Get a specific post

In: DELETE posts/post_id
Out: /
Logic: Delete a specific post

In: GET posts/stats
Out: Number of posts on each day
Logic: Get the post statistics, i.e. number of posts on each day

Like service:

In: POST likes (post_id, user_id)
Out: /
Logic: User likes a specific post

In: GET likes/posts/post_id
Out: Likes (post_id, user_id)
Logic: Get all the likes of a post

In: DELETE likes/posts/post_id (user_id)
Out: /
Logic: Undo a like

Tag service:

In: POST tags (post_id, user_id's)
Out: /
Logic: Create user tags on a specific post

In: GET tags/posts/post_id
Out: Tags (post_id, user_id)
Logic: Get all the tags on a post

Comment service:

In: POST comments (post_id, creator, content)

Out: /
Logic: Create a comment on a specific post

In: GET comments/comment_id
Out: Comment (comment_id, creator, timestamp, content)
Logic: Get a specific comment

In: GET comments/posts/post_id
Out: Comments (comment_id, creator, timestamp, content)
Logic: Get all the comments on a post

In: DELETE comments/comment_id
Out: /
Logic: Delete a comment