Master Informatique 1, Groupe F
MAGIEU Pierre, CHEMIN Elisée

Group F :
Tchat App Report

Features **1**
Logging in/out of an account 1
History 1
Description of the architecture **1**
Figure 1. Architecture of the shared objects 1
How to use the application **2**
Manual compilation using Maven 2
Running the application 2

Useful links :

- Imag's Gitlab Repository : IDS Groupe F : TchatApp

⚠ At the moment of writing the report, the im2ag's Gitlab which we've used for the majority of the project is in readonly mode. The repository will be updated once write permissions are restored.

- **Backup Github Repository** : IDS Groupe F : TchatApp

Features

Our implementation of the TchatApp allows multiple users to communicate in real time using a broadcast messaging system very similar to a discussion thread.

Logging in/out of an account

When launching the client, the user chooses either to log in with an existing account or to create a new account. Accounts are persistent and stored on the server's side. When a login attempt is made, the server verifies the credentials before allowing the user to chat. Users can log out at any time using the $/quit$ command.

History

All the messages are automatically stored in a .ser file upon reaching the server to ensure data persistence. Any user can access the history at any time using the $/history$ command.

Description of the architecture

Our code is articulated around "services", which are implementations of shared interfaces. These services are provided to the client by the server to be executed on the server's side.

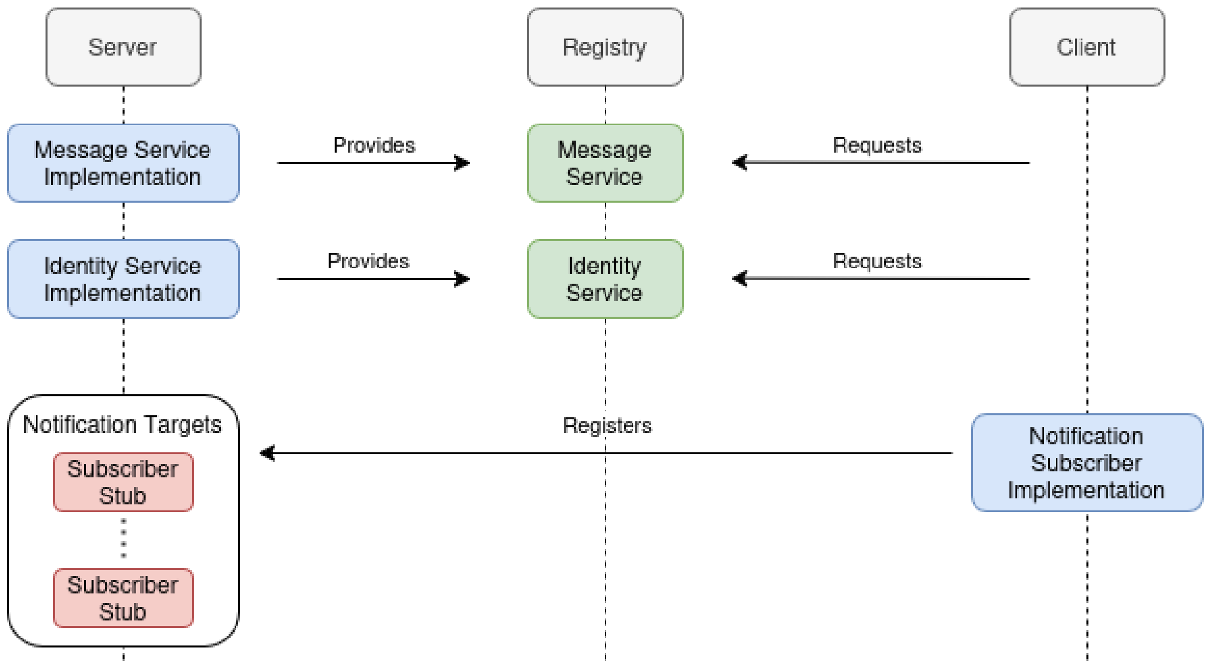

Figure 1. Architecture of the shared objects

The first service is the Identity Service; it allows a client to create an account and connect to it. When connecting, the service will return an Identity instance corresponding to the client. This instance is then used to identify which client made a request.

The second service is the “Message Service”’; It’s used by the client to send and receive messages. To receive a message, a client must be registered as a receiver, which is done by forwarding the client’s NotificationSubscriber instance to the server. When a notification needs to be shared, the Message Service will call the corresponding callback function of the NotificationSubscriber.

As this is only a proof of concept, we’re currently storing the messages and accounts into files inside of the data directory, but for future releases we might want to use something more scalable such as a database. Fortunately for us, the choice of storage is not described in the public interface. Which means we can easily switch to another type of storage by registering a different implementation of the services to the registry.

How to use the application

To use the TchatApp, you must run 3 separate programs: the remote class registry (rmiregistry), the server and one or more clients. To make it easier, we provided 3 bash scripts available in the root directory.

Manual compilation using Maven

If you want to explicitly compile the sources, you can use the following commands :

```
mvn clean compile
```

This command should create a “target” folder containing the compiled classes.

Running the application

- ⚠ The client / server scripts don’t automatically compile the sources. We recommend either compiling the sources manually or running the *rmi* script first.
- ⚠ By default, the client script uses “localhost” as the server address.

rmi.sh : Compiles the sources and starts the registry in the background using port 1099
server.sh : Starts the server using port 1099 for the registry
client.sh : Starts a client using localhost:1099 as it’s server address