Assignment 2 - The Actor Model using Akka

> *Published: 25.11.2019*
>
> *Due: 25.12.2019 23:59*

In this assignment we will be implementing a textual-based WhatsApp! clone. It will contain most of the features of the application. Make sure to imeplement it in Akka, preferebly using Java.

We will detail below the list of required features that must be implemented. Be sure to think in Actor Model mode, and not object-oriented mode. Make sure to incorporate the different behaviors needed for each Actor in the system.

The application will rely on two main features:

- 1-to-1 chat. In this feature a person using the application will send an addressed message to a specific destination.
- 1-to-many chat. In this feature a person using the application will send a message addressed to the group itself. This message is broadcast to all members found in the specific group.

Chat Features

a person may send a message containing either text, or binary date to an addressed target. The textual data will be displayed on screen. The binary data will be saved, and the full path of the file will be printed on screen. The binary data may contain any form of data, such as image, video, executable, etc.

- each message to be sent will be sent using this command:
    - textual message: **/user text <target> <message>**
        - if <target> does not exist, an error will be printed: "<target> does not exist!
        - otherwise, the <message> will be sent to <target>
            - Message received will be printed in the following format at <target>: [<time>] [user][<source>]<message>
    - binary message: **/user file <target> <sourcefilePath>**
        - if <target> does not exist, an error will be printed: "<target> does not exist!

- if <filePath> does not exist, an error will be printed: "<sourcefilePath> does not exist!"
- otherwise, <sourcefilePath> will be sent, in binary mode, to <target>. <target> will store the file locally.
    - binary message received will have the following message printed: [<time>][user][<source>] File received: <targetfilepath>

User Features

A user may connect to the server and disconnct from the server. A user connecting to the server will do so by choosing their username. The username must be unused one, and is unique. The server will hold the details of online users only. A disconnected user equals to a deleted user. Make sure to implement the disconnect feature properly.

- Connecting a user to the server: **/user connect <username>**
    - if the server is down, an error will be printed: "server is offline!"
    - if the <username> is in use, an error will be pritned: "<username> is in use!"
    - otherwise:
        - the user will connect to the **managing server** and a successful operation will print: "<username> has connected successfully!"
- Disconnecting a user from the server: **/user disconnect**
    - if the server is down, an error will be printed: "server is offline! try again later!"
    - otherwise, a disconnecting user means the following:
        - <username> gracefully leaves all groups:
            - if <username> is admin of a group, the groups needs to be dismantled
            - otherwise <username> will be removed from a group and lose previledges
        - then, all information of <username> in the managing server will be purged.
        - success message will be returned to user who prints: "<username> has been disconnected successfully!"
            - make sure to user the already implemented features to implement the disconnect command. Use the leave group functionality!

Group Chat Features

Group chat has multiple commands and multiple features. A user creating a group will be its admin. The admin can invite users to group, remove users from group, and promote users found in group to be co-admins. Lastly the

admin can demote co-admins back to users or even remove them from group. A co-admin removed from group will lose their co-admin previledges as well as the ability to post in the group. A user in group can be of one of 4 possible modes: (1) admin (2) co-admin (3) user (4) muted user. These modes are in relation to a *specific* group! A user may be an admin of group one, while being a muted user in group two.

Previledges

- group admin:
    - may send text/file messages
    - may add/remove co-admins
    - may invite/remove users
    - may promotes: muted user to user to co-admin
    - may demote: co-admin to user to muted user
    - may create new group
    - may leave existing group
- group co-admin:
    - may send text/file messages
    - may invite/remove users
    - may promote: muted user to user
    - may demote: user to muted user
    - may create new group
    - may leave existing gorup
- group user:
    - may send text/file messages
    - may create new group
    - may leave existing group
- group muted user:
    - may create new group
    - may leave existing group

Printing

Printing any <message> to screen will be prefixed by the following:

- [<time>][<groupname>][<sourceusername>]<message>
- **Note:** <X> means X is a variable. For example, if X equals “HELLO”, then you will be printing HELLO to screen, and **not** <HELLO>!

Commands

- create group: **/group create <groupname>**
    - if the group exists, an error will be printed: “<groupname> already exists!”.
    - otherwise:
        - <groupname> will be created
        - the user creating the group will be its **admin**
        - a success message will be printed: “<groupname> created successfully!”
- leave group: **/group leave <groupname>**
    - if <sourceusername> is of type group muted user or group user:
        - if <sourceusername> is not in <groupname>, an error message will be printed: <sourceusername> is not in <groupname>!.
        - otherwise:
            - <sourceusername> will be removed from <groupname>
            - a notification message will be sent to <groupname> of the action, to be broadcast to each <targetusername> in <groupname>.
            - each <targetusername> will print: <sourceusername> has left <groupname>!"
    - if <sourceusername> is of type group co-admin:
        - same as above, in addition:
        - <sourceusername> is removed from co-admin list in <groupname>
    - if <sourceusername> is of type group admin:
        - same as above, in addition:
        - all users of all types will be removed from group and a notification message is broadcast to each <username> in <groupname>

        - <groupname> admin has closed <groupname>!
    - group will be deleted
- send text to group: **/group send text <groupname> <message>**
    - if <groupname> does not exist, an error will be printed: "<groupname> does not exist!
    - otherwise:
        - <message> will be sent to <groupname>, which will broadcast it to every <targetusername> in <groupname>
        - <targetusername> will print the received message in the following format: [<time>][groupname][<sourceusername>]<message>
- send binary message: **/group send file <groupname> <sourcefilePath>**
    - if <groupname> does not exist, an error will be printed: “<groupname> does not exist!”
    - if <sourceusername> not part of <groupname>, an error will be printed: “You are not part of <groupname>!”
    - if <sourceusername> is muted in <groupname>, an error will be printed: “You are muted for <time> in <groupname>!”
    - if <sourcefilePath> does not exist, an error will be printed: “<sourcefilePath> does not exist!”
    - otherwise:
        - <filePath> will be sent, in binary mode, to all <targetusername> found in <groupname>.
        - Each <targetusername> will store the file locally, and a message will be printed: “File received: <targetfilepath>”
- invite user: **/group user invite <groupname> <targetusername>**
    - if <groupname> does not exist, an error will be printed: “<groupname> does not exist!”
    - if <sourceusername> not admin or co-admin of <groupname>, an error will be printed: “You are neither an admin nor a co-admin of <groupname>!”
    - if <targetusername> does not exist, an error will be printed: “<targetusername> does not exist!”
    - if <targetusername> is in <groupname>, an error will be printed: “<targetusename> is already in <groupname>!”
    - otherwise:
        - a message will be sent to <targetusername> regarding the invite
        - <targetusername> will print a message: “You have been invited to <groupname>, Accept?”
        - <targetusername> may accept [Yes] or deny [No] the invite. Response will be sent back to <sourceusername<

        - <sourceusername> will add <targetusername> to <group> is [Yes] has been received. And send a “Welcome to <groupname>!” to <targetusername> who will print the result.
- remove user: **/group user remove <groupname> <targetusername>**
    - if the <groupname> does not exist, an error message will be printed: "<groupname> does not exist!
    - if <targetusername> does not exist, an error message will be printed: “<targetusername> does not exist!”
    - if <sourceuser> is not an admin or co-admin of <groupname>, an error message will be printed: “You are neither an admin nor a co-admin of <groupname>!”
    - otherwise:
        - <targetusername> will be removed to the <groupname> co-admin list.
        - a notification message will be sent from <sourceusername> to <targetusername>: “You have been removed from <groupname> by <sourceusername>!”
        - <targetusername> will print the received message: [<time>][<groupname>] [<sourceusername>]: message
- mute user: **/group user mute <groupname> <targetusername> <timeinseconds>**
    - if the <groupname> does not exist, an error message will be printed: "<groupname> does not exist!
    - if <targetusername> does not exist, an error message will be printed: “<targetusername> does not exist!”
    - if <sourceuser> is not an admin or co-admin of <groupname>, an error message will be printed: “You are neither an admin nor a co-admin of <groupname>!”
    - otherwise:
        - <targetusername> will be muted in the <groupname> for <timeinseconds>
        - a notification message will be sent from <sourceusername> to <targetusername>: “You have been muted for <timeinseconds> in <groupname> by <sourceusername>!”
        - <targetusername> will print the received message: [<time>][<groupname>] [<sourceusername>]: message
        - <targetusername> will be unmuted *automatically* after <timeinseconds> passes!
            - unmuting will be done automatically

        - <targetusername> will change status from muted user to user
        - a message will be printed: “You have been unmuted! Muting time is up!”
- unmute user: **/group user unmute <groupname> <targetusername>**
    - if the <groupname> does not exist, an error message will be printed: "<groupname> does not exist!
    - if <targetusername> does not exist, an error message will be printed: “<targetusername> does not exist!”
    - if <sourceuser> is not an admin or co-admin of<groupname>, an error message will be printed: “You are neither an
      admin nor a co-admin of <groupname>!”
    - if <targetusername> is not muted, an error message will be printed: “<targetusername> is not muted!”
    - otherwise:
        - <targetusername> will be unmuted in the <groupname>
        - a notification message will be sent from <sourceusername> to <targetusername>: “You have been unmuted in <groupname> by <sourceusername>!”
        - <targetusername> will print the received message: [<time>][<groupname>] [<sourceusername>]: message
- promote to co-admin: **/group coadmin add <groupname> <targetusername>**
    - if <groupname> does not exist, an error message will be printed: "<groupname> does not exist!
    - if <targetusername> does not exist, an error message will be printed: “<targetusername> does not exist!”
    - if the user is not an admin or co-admin of <groupname>, an
      error message will be printed: “You are neither an admin nor co-admin of
      <groupname>!”
    - otherwise:
        - <targetusername> will be be added to the <groupname> co-admin list.
        - <targetusername> will receive a notification message of the promotion
        - <targetusername> will gain co-admin previledges in
          <groupname> and print: “You have been promoted to co-admin in
          <groupname>!”
- demote co-admin: **/group coadmin remove <groupname> <targetusername>**
    - if <groupname> does not exist, an error message will be printed: "<groupname> does not exist!

    - if <targetusername> does not exist, an error message will be printed: "<targetusername> does not exist!"
    - if the user is not an admin or co-admin of <groupname>, an error message will be printed: "You are neither an admin nor co-admin of <groupname>!"
    - otherwise:
        - <targetusername> will be removed from the <groupname> co-admin list.
        - <targetusername> will be receive a notification of the demotion.
        - <targetusername> will lose co-admin previledges in <groupname> and print: "You have been demoted to user in <groupname>!"

Managing Server

To handle groups, and usernames we need a server to manage the system. It will store details regarding the Groups as well as the Users created. It will also act as an access point to the Whatsapp! system.

The managing server will handle these features:

- Creating a User:
    - a user initiating a **connect** command will first contact the managing server to check whether the username is **not** used. The managing server will notify the requesting target of the result.
    - a user sending a *disconnect* command will remove the user information in the Manager.
- Creating a Group
    - a user initating a **group create** command will first contact the managing server to check whether the group has been created before or not.
- Deleting a Group:
    - an admin user initiating a **group leave** command will notify the managing server, which as a result will free the received group name for use.
- Managing group user list:
    - **Messages sent to a group** will be sent to the managing server, which in return will broadcast it to all group members.
    - **Users invited to the group** will be stored in the group found at the managing server.
    - **Users removed from the group** will be updated in the managing server.
    - **Active Users:** users will hold on their unique usernames as long as they are connected to the system. Once a user **disconnects**, anything related to this user must be cleared up! This means

their details in the managing server as well as their information in participating groups.

Manager-Client Relationship

Remember, the Actor model is a **distributed** system! Messages to be sent from user to user must be sent *directly*.
Sending a message from user to manager to user is **wrong**!
The manger handles **managing** related messages **only**.

Deliverables

You need to create a file callled **id1_id2_.zip**, where id1 is the id number of the first partner, and id2 is the is number of the second parter in group. Inside the archive file, you need to have four files:

- Your akka project folder - it needs to contain the complete code! No compiled code please! Make sure to remove class files!
- README - containing the names and ids of the group. It will also contain the design of the Actor Model you've implemented using Akka, the different actors, hierarchy, implemented behaviors, and kinds of messages passed and their effect on behavior. Be sure to detail it to the fullest!

Grading

You will be assessed on five main categories:

- **Functionality:** Your functionality grade is determined purely by adherence to the above specification. The submitted code must behave as requested in terms of executing and output then ✓+ for functionality.
- **Akka practices:** Proper use of the Akka tools and ways of thinking introduced in this class - using the Actor model in a correct way, immutability, routing, and everything related to akka.
- **Program design:** General programming style - functional programming such as decomposition, commenting, logic,
  algorithm design, adhering to the Actor model, correct use of Akka tools.
- **Akka mechanics:** Basically everything we have covered - in regard of Akka and the Actor model.
- **Bonus Task:** Any group implementing the front end using the Play Framework correctly, and explain in detail their imeplementation at the frontal check, will receive **up to 25** points as a bonus to the final grade.