"Helping you connect with the world one clack at a time"

Project Title: Clack

Authors:
Manjot Singh
Adam Fipke
Sitt Hmue Paing
Beck Corkle
Levi Boswell

Product Description

A Windows client-server application with a modern slim flat UI with a small sleek form factor that allows users to chat with other users (both individually and in groups). Users can have an account in which they must log in to use the program. If the user forgot their password, they can indicate as such and receive a recovery email to change their password. Once logged in user can add friends by providing their friend's username. The user will be able to view the online status of their friends at anytime. The user can also unfriend a friend in their friend list. Users can only send messages to people in a group or directly to friends. Users can create groups and invite friends to the group upon creation or after creation. The user that creates the group will be an admin for that group, allowing them to kick other users. Users can leave groups, and if the admin leaves a group, the next oldest member will become the new admin. An admin can transfer their adminship to another member in the group. Users can log out when they are done.

Stretch goals (features to add if we have time, priority in order):

1. Upload images
2. Upload files
3. Delete messages
4. Change username
5. Blocking user
6. Editing a message
7. Voice chat
8. Servers (aka group chats with multiple text and voice chats with a role hierarchy)
9. Pin chat messages (so they are easier to access)
10. Delete account

Requirements

Functional requirements

- **Create Account**: Allow users to provide an email address, a username, and a password to create their own account with Clack
- **Login**: Allow users to log in to their account
- **Logout**: Allow users to log out of their account
- **Add Friend**: Allow users to add friends
- **Remove Friend**: Allow users to remove friends
- **Create Group**: Allow users to create a messaging group with 1 or more people
- **View Online Users**: Allow users to see the online status of their friends
- **Kick Member**: Allow users to kick people from a messaging group that they own
- **Leave Group**: Allow users to leave a messaging group they are in
- **Add User To Group**: Allow users to add people to an existing messaging group

- **Send Message**: Allow users to send messages in a messaging group or a direct message. The user must be able to receive a message even if they are not actively online
- **View Group Messages**: Allow users to read messages and the message history from others in messaging groups
- **View Direct Messages**: Allow users to read direct messages and the message history from others with one other person
- **Password Recovery**: When users are not able to remember their password, they can easily recover their password by providing their email to receive a link for password recovery.

Non-functional requirements

- We will use Kanban for our software development process
    - Using a Trello board
    - We will have 2 weekly meetings (one on Wednesday at 7:30pm and another on Friday at 1pm), which group members will be expected to participate
- The program must be able to run on Windows 10 and above
- The program must be made in Python 3.10
- Chat history must be stored indefinitely.
- Another user's messages must be received within a second.
- The app can be used anytime anywhere.
- Files and images can be shared easily within a few clicks.
- All the private information must be kept strictly confidential.
- User account must be linked to a valid email address.
- The system must be able to handle up to thousands of online users at a time
- Viewing the online status of friends should take no more than 1 click to view
- The system must not let users have the same username as other other users

Clack Use Cases

Use Case 1: Create Account

Primary actor: User
Description: Allows a user to create their own account with Clack
Pre-condition: None
Post-condition: User account is created and user is prompted to log in

Main scenario:

1. Users input their account information (username, email, password etc.)
2. User tells system to make an account with the account information
3. System creates an account and stores account information

4. System tells user that the account was successfully created and prompts them to log in

Extensions:

1.1 Invalid account information
- 1.1.1 System displays an error informing the user their information is invalid (e.g. password too short)

2.1 Internal System error
- 2.1.1 The system fails to create the account
- 2.1.2 System displays an error message informing the user to try again later

3.1 User already exists
- 3.1.1 User submits an email or username which already exists
- 3.1.2 System displays an error informing the user to change their input

Use Case 2: Log in

Primary actor: User
Description: Allow users to log in to their own private account.
Pre-condition: User must have an account
Post-condition: The user will be able to log in to the Clack and have access to all features of Clack.

Main scenario:
1. User provides username and password
2. User tells system to login

Extensions:
1.1 Invalid username
- 1.1.1 The user entered invalid characters or username that doesn't exist. The system tells the user this

1.2 Invalid password
- 1.2.1 The user entered an invalid password. The system tells the user this and offers password recovery

Use Case 3: Log out

Primary actor: User
Description: Allow users to log out of their own account.
Pre-condition: The user should be logged in
Post-condition: User is no longer logged in

Main scenario:

1. User tells system to log out

Extensions:
1.1 System error
    1.1.1 System errors when logging out user, inform the user they are not logged out

Use Case 4: Add friend

Primary actor: User
Description: Allow users to add friends
Pre-condition: The users must be logged in
Post-condition: Friend is added.

Main scenario:
1. User locates a user
2. User indicates an intention to add them as friends

Extensions:
1.1 Unable to add friends.
- 1.1.1 When the user has been blocked, they will no longer be able to send messages to that person or send friend requests.

1.2 User does not exist
- 1.2.1 If the indicated user does not exist, the system will tell the user as such and the friend will not be added

Use Case 5: Remove friend

Primary actor: User
Description: Allows the user to remove friends
Pre-condition: User must have friends; User must be logged in
Post-condition: Friend is removed.

Main scenario:
1. User indicates they want to remove a friend
2. System breaks the friendship 🙁

Extensions:
1.1 Friend does not exist
    1.1.1 User indicates a friend which does not exist

1.1.2 System displays an error message indicating friend does not exist
2.1 Error breaking friendship
2.1.1 System displays an error message saying friendship wasn't broken

Use Case 6: View message history of friend

Primary actor: User
Description: Able to view previous messages
Pre-condition: The user must be logged in; The user must be friends with the recipient.
Post-condition: Can view messages with friends

Main scenario:
1. User indicates friend they want to view message history of
2. Message history is displayed to user

Extensions:
2.1 History could not be retrieved
2.1.1 System reaches error attempting to retrieve history
2.1.2 System displays an error to the user indicating history is not complete

Use Case 7: Create group

Primary actor: User
Description: The user is able to create a group with other users to have a group interaction and sharing of different information.
Pre-condition: User must be logged in.
Post-condition: Successfully created a group.

Main scenario:
1. User indicates they want to create a group
2. The system creates group

Extensions:
2.1 Error creating group
2.1.1 System errors when creating group
2.1.2 Display error indicating group was not created

Use Case 8: View chat history

Primary actor: User
Description: Users are able to send messages in groups and view group’s messages.
Pre-condition: User is logged in; User is in a group;

Post-condition: Message is sent in a group;

Main scenario:

1. User indicates they want to send a message to a group
2. User inputs message contents
3. System stores message and displays it to all other users in group

Extensions:

2.1 Invalid message

2.1.1 User inputs invalid message contents

2.1.2 System displays error indicating the requirements they failed to meet

Use Case 9: Send Message:

Primary actor: User

Description: User can send a text message.

Pre-condition: User is logged in;

Post-condition: Validate message that confirms that it was added to history

Main scenario:

1. User locates group or direct message they want to send a message in
2. Enters input to prompt
3. System sends message to group or direct message

Extensions:

2.1 Format error

2.1.1 Character length reached and will refuse to send a message.

Use Case 10: Leave group

Primary actor: User

Description: Allow user to leave a group

Pre-condition: The users should be logged in; User should be in a group

Post-condition: User has left the group

Main scenario:

1. User indicates they would like to leave a group
2. The system removes the user from group

Extensions:

2.1 Error leaving group
    2.1.1 System has an error removing user from group
    2.1.2 System informs user has not left the group
2.2 User is admin for the group
    2.2.1 The system will automatically transfer the adminship to the next oldest member in the group

Use Case 11: Kick member

Primary actor: User
Description: User is allowed to kick member from the group
Pre-condition: User must be logged in; User is in a group; User has admin privileges in group;
Post-condition: Kicked member is no longer in group

Main scenario:
1. User indicates group they would like to use
2. User indicates user the would want to kick
3. User indicates intention to kick user
4. System removes user from the group

Extensions:
4.1 Error removing user from group
    4.1.1 System errors when removing user from group
    4.1.2 User should be informed the user was not kicked

Use Case 12: Transfer adminship

Primary actor: User
Description: User can transfer the ownership of a group to another person
Pre-condition: User must be logged in; User must own the group;
Post-condition: Another person has ownership of the group

Main scenario:
1. User indicates group they would like to use
2. User indicates user they would want to swap ownership with
3. User indicates intention swap ownership
4. System swaps ownership

Extensions:
4.1 Error swapping ownership
    4.1.1 System errors when swapping ownership
    4.1.2 User is informed the operation failed

Use Case 13: Add user to group

Primary actor: User
Description: User is can add another user to a group
Pre-condition: User is logged in; User has permission to add another user to group; User is in group
Post-condition: User is added to group

Main scenario:

1. User indicates group they would like to use
2. User indicates user they would want to add
3. System adds user to group

Extensions:
3.1 Failed to add user
    3.1.1 System errors when adding user to group
    3.1.2 User is informed the operation failed

Case 14: View online Users

Primary actor: User
Description: Can see who of their friends is online in a group
Pre-condition: Logged in (case 2.)
Post-condition: User knows who of their friends is online

Main scenario:

1. User wants to see who of their friends is online
2. System gathers statuses of all the users friends
3. System displays statuses to user

Extensions:
2.1 Error getting status
    2.1.1 System errors when retrieving the status of a friend
    2.1.2 User is informed that the status is unknown

Case 15: Forgot Password

Primary actor: User
Description: If a user forgot a password, they can recover their password

Pre-condition: The user must have an account
Post-condition: The password is changed, user can login

Main scenario:

1. The user indicates that they want to recover their password.
2. The user will provide their email for password change.
3. The system will send a link to the email for account recovery.
4. The user will input a new password for the account.
5. The system will be notified of the password changes
6. The user will be able to login in successfully after password recovery.

Extensions:

2.1 Wrong email/Forgot email

2.1.1 Link will be sent to provided email without checking whether the email existed

2.1.2 Users who forgot their email will not be able to receive a link for password recovery.

3.1 System error

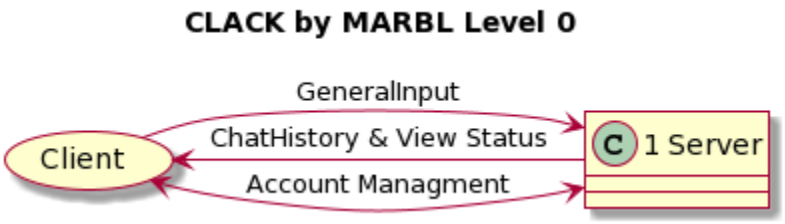
3.1.1 The system will not be able to send a link for account recovery.

3.1.2 System displays error message

Clack DFD

Level 0

Diagram



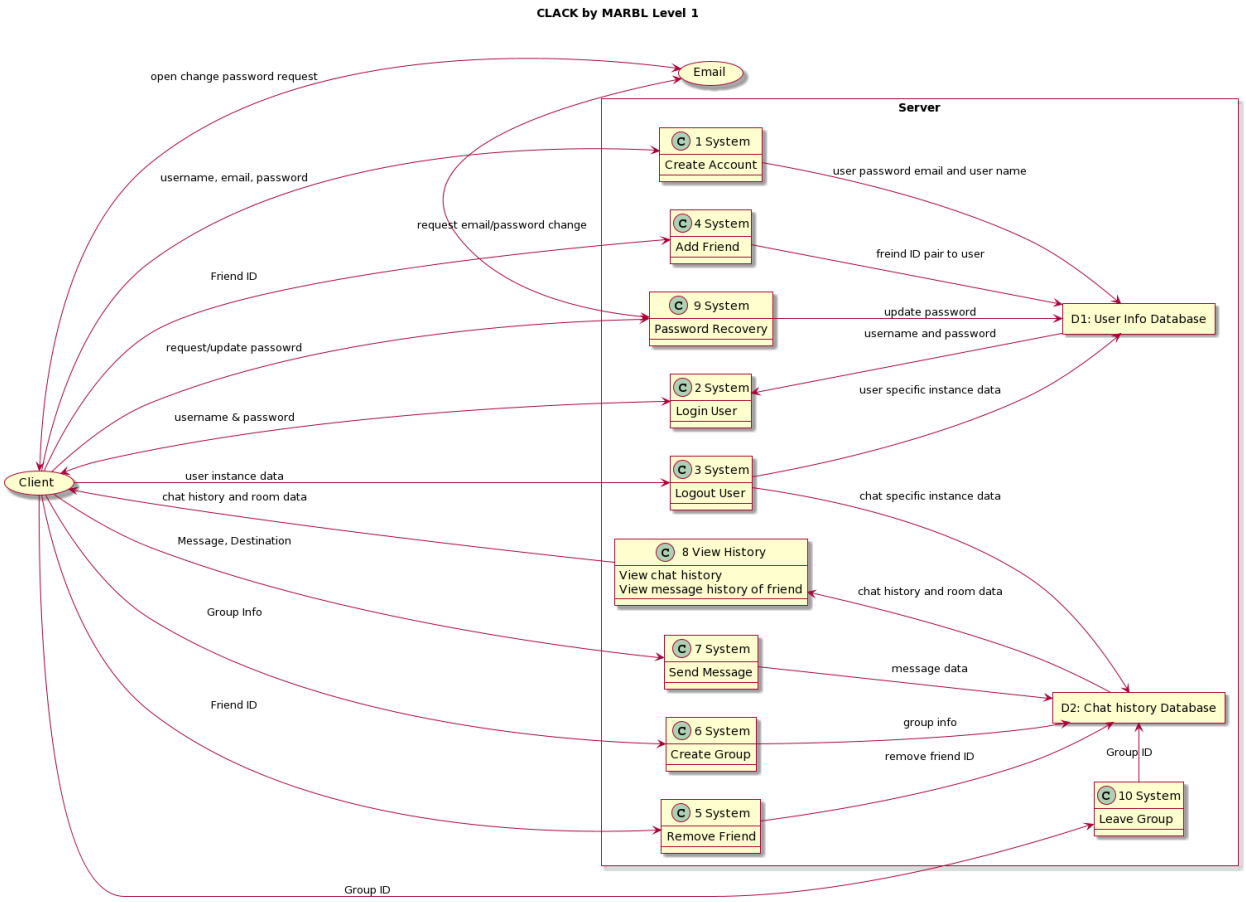
Code

```
@startuml
```

```
skin rose

title  CLACK by MARBL Level 0
allowmixing

usecase "Client" as User

class "1 Server" AS rest {
}
User-right-> rest:  GeneralInput
rest -left-> User: ChatHistory & View Status
rest <-left-> User: Account Management


@enduml
```

Level 1

Diagram



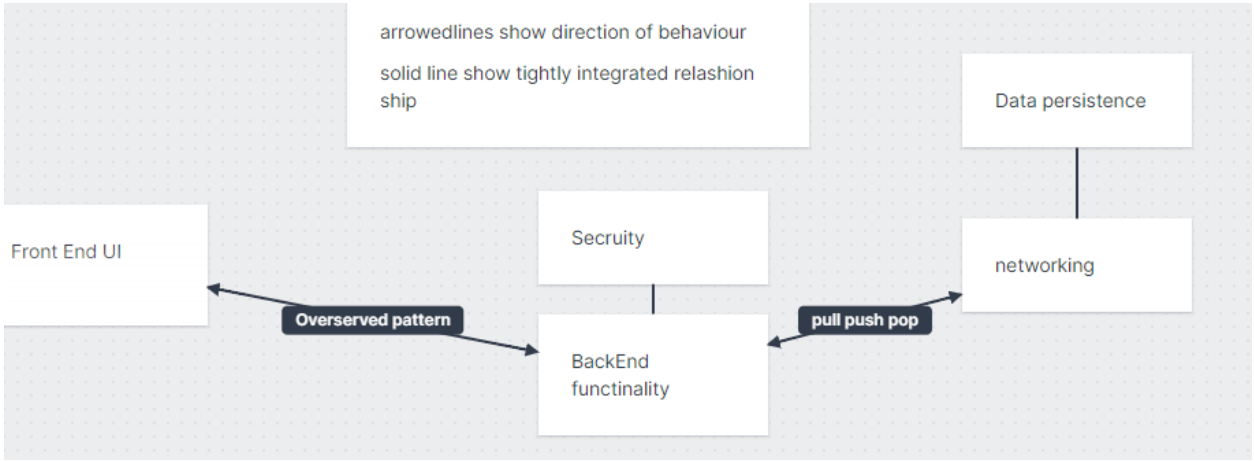
Code

```
@startuml

skin rose

title  CLACK by MARBL Level 1
allowmixing
left to right direction

usecase "Client" as C
usecase "Email" as E
```

```
rectangle "Server" {

class "1 System" AS CA{
      Create Account
}
class "2 System" AS L{
      Login User
}
class "3 System" AS LO{
      Logout User
}
class "4 System" AS AF{
      Add Friend
}
class "5 System" AS RF{
      Remove Friend
}
class "6 System" AS CG{
      Create Group
}
class "7 System" AS SM{
      Send Message
}
class "9 System " as PR{
      Password Recovery
      }
class "8 View History" as VH{
      View chat history
      View message history of friend
      }

class "10 System" as LG{
      Leave Group
      }

rectangle "D2: Chat history Database" as CDB

rectangle "D1: User Info Database" as UDB

PR --> UDB : update password
PR <-> E : request email/password change
E <-left-> C : open change password request
C -> PR : request/update password
```

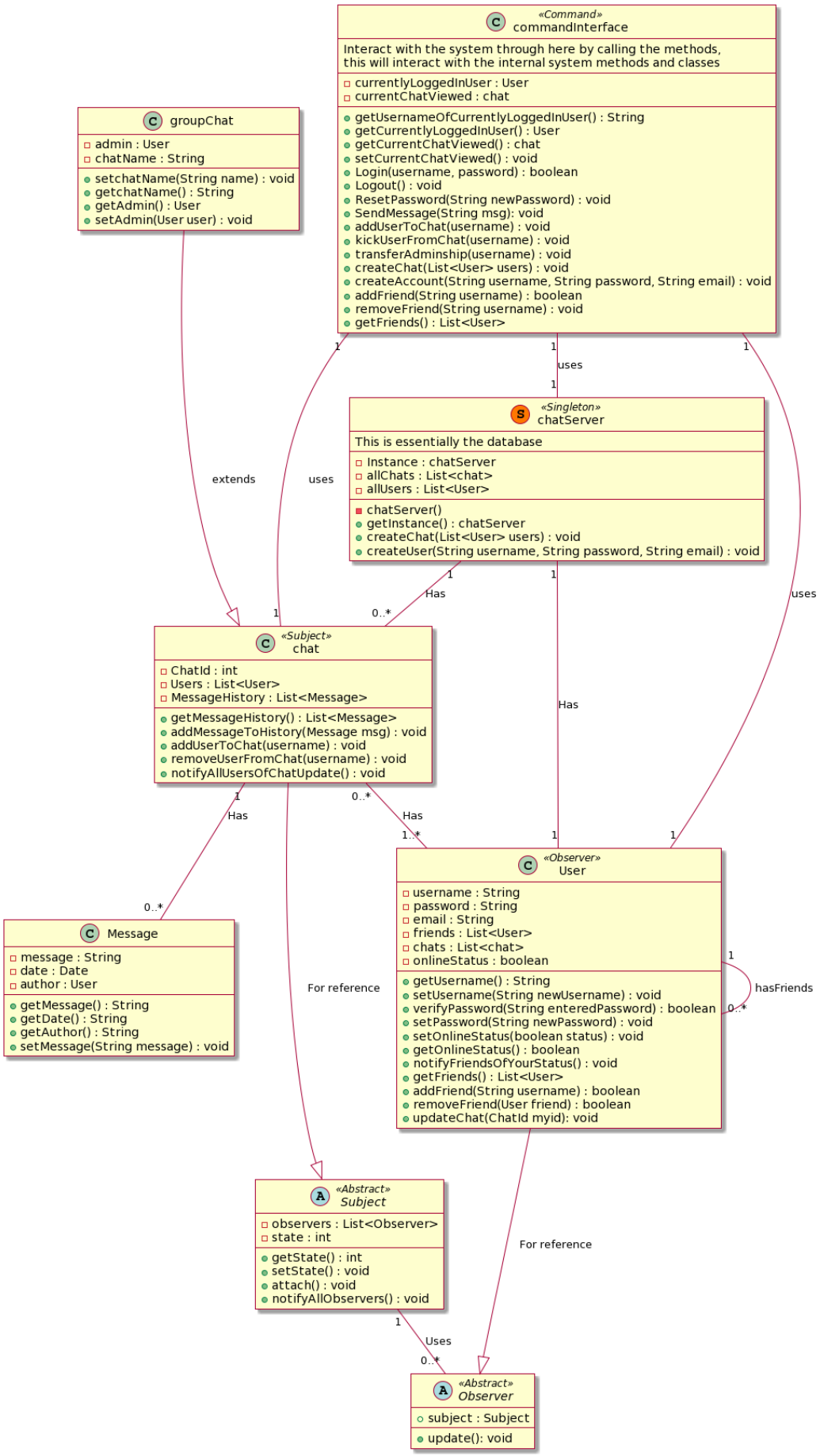
```
CA  --> UDB : user password email and user name
L <-- UDB : username and password
C <-> L : username & password
C -> CA :username, email, password
C -> RF : Friend ID
C -> SM : Message, Destination
C -> CG: Group Info
C -> LG: Group ID
LG -> CDB : Group ID
RF --> CDB : remove friend ID
C -> AF : Friend ID
C -> LO : user instance data
LO --> UDB : user specific instance data
LO --> CDB : chat specific instance data
VH <-- CDB : chat history and room data
C <---- VH : chat history and room data
CG --> CDB : group info
AF -down-> UDB : friend ID pair to user
SM -down-> CDB : message data
}
```

Architecture Diagram



Class Diagram

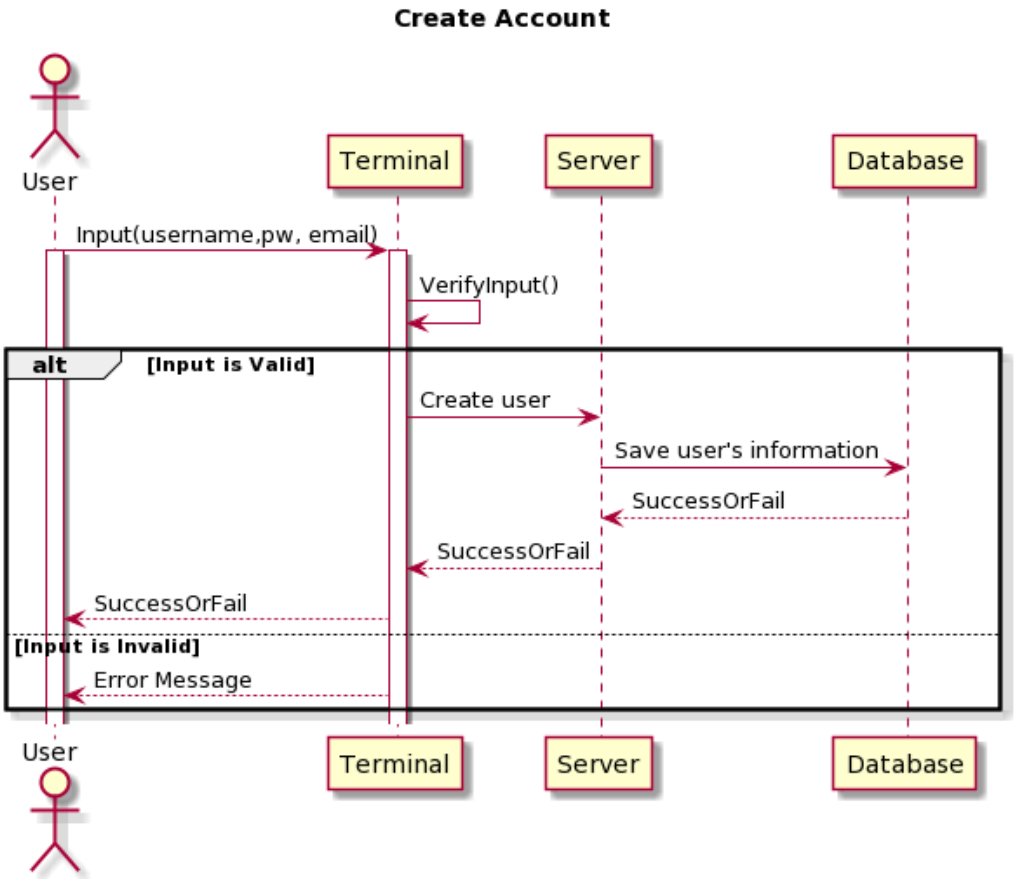
Types - Class Diagram



Sequence Diagram

Create Account

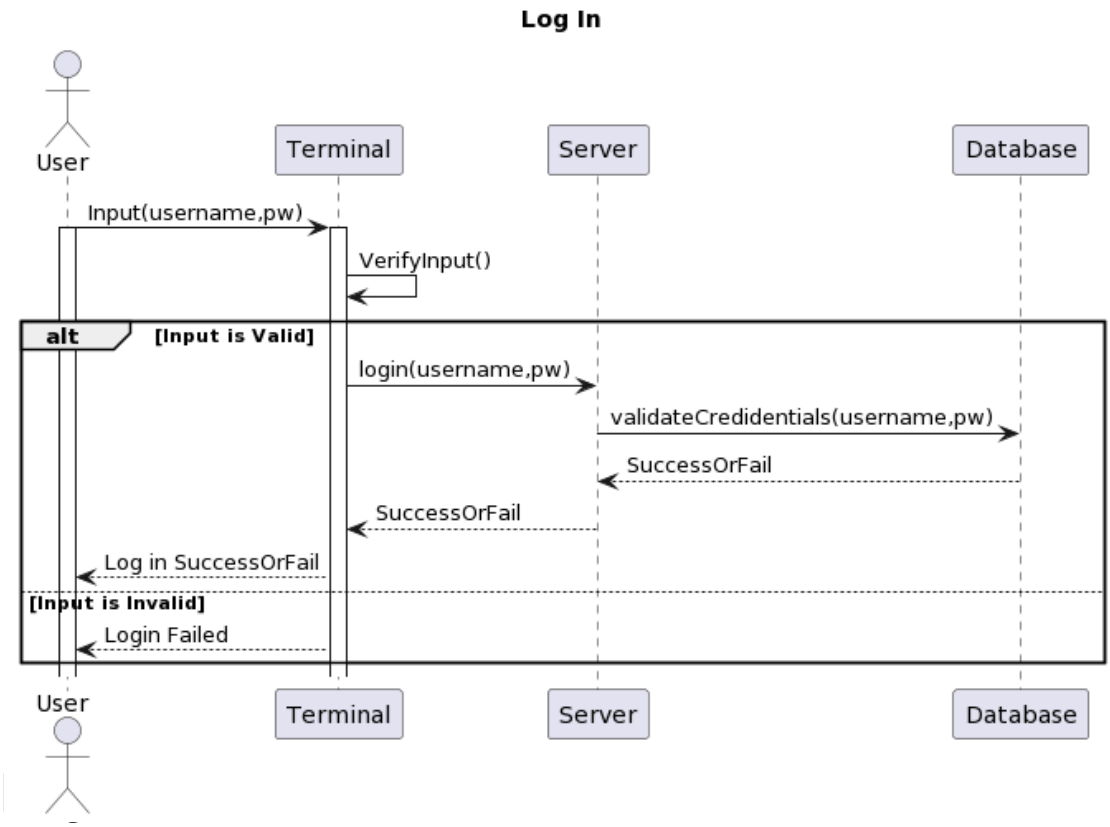
```
@startuml

skin rose

title  Create Account
actor User
participant Terminal
participant Server
participant Database


User -> Terminal : Input(username,pw, email)
activate User
activate Terminal

Terminal -> Terminal :VerifyInput()
alt Input is Valid
Terminal -> Server : Create user
Server -> Database : Save user’s information
Database --> Server: SuccessOrFail
Server --> Terminal: SuccessOrFail
Terminal --> User: SuccessOrFail
else Input is Invalid
Terminal --> User: Error Message

end
```



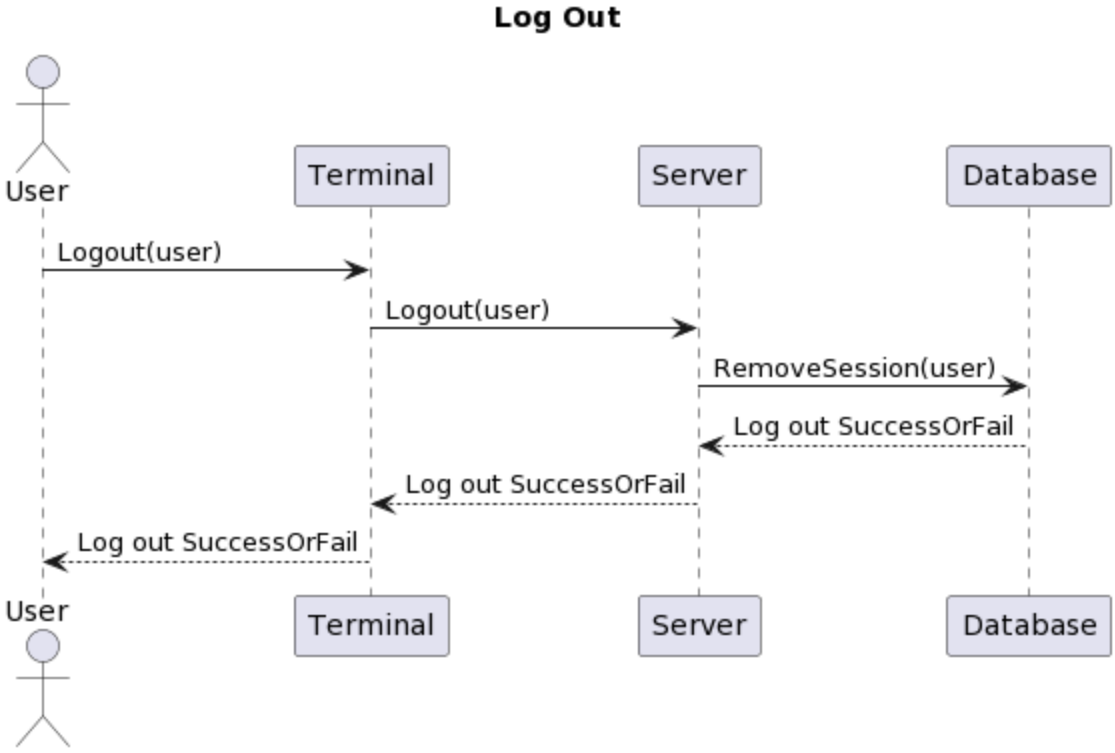
Login

```
title  Log In
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Input(username,pw)
activate User
activate Terminal

Terminal -> Terminal :VerifyInput()
alt Input is Valid
Terminal -> Server : login(username,pw)
Server -> Database: validateCredidentials(username,pw)
Database --> Server : SuccessOrFail
```

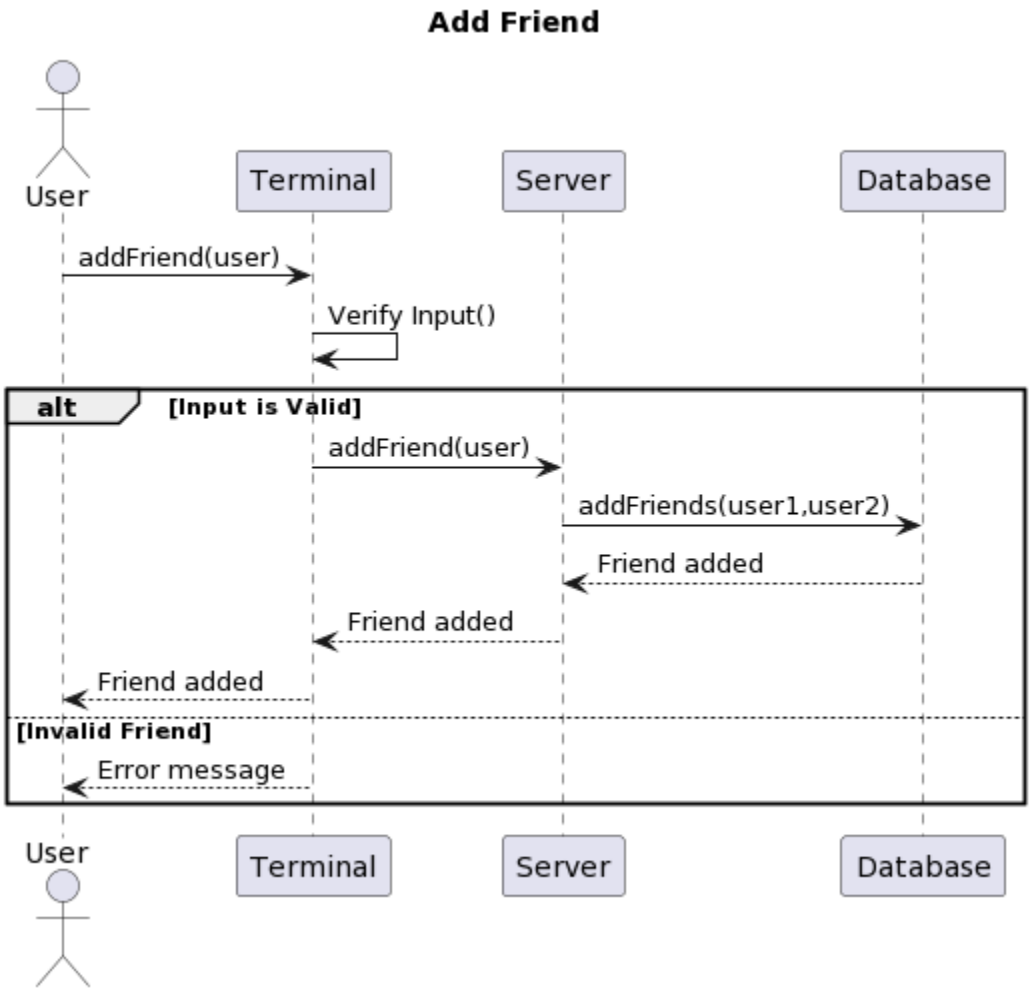
```
Server --> Terminal : SuccessOrFail
Terminal --> User : Log in SuccessOrFail
else Input is Invalid
Terminal  --> User : Login Failed

end
```




```
title  Log Out
actor User
participant Terminal
participant Server
participant Database

User ->Terminal : Logout(user)
Terminal -> Server : Logout(user)
Server -> Database: RemoveSession(user)
Database --> Server: Log out SuccessOrFail
Server --> Terminal : Log out SuccessOrFail
Terminal --> User : Log out SuccessOrFail
```


Log Out
User
Terminal
Server
Database
Logout(user)
Logout(user)
RemoveSession(user)
Log out SuccessOrFail
Log out SuccessOrFail
Log out SuccessOrFail
User
Terminal
Server
Database

Add Friend



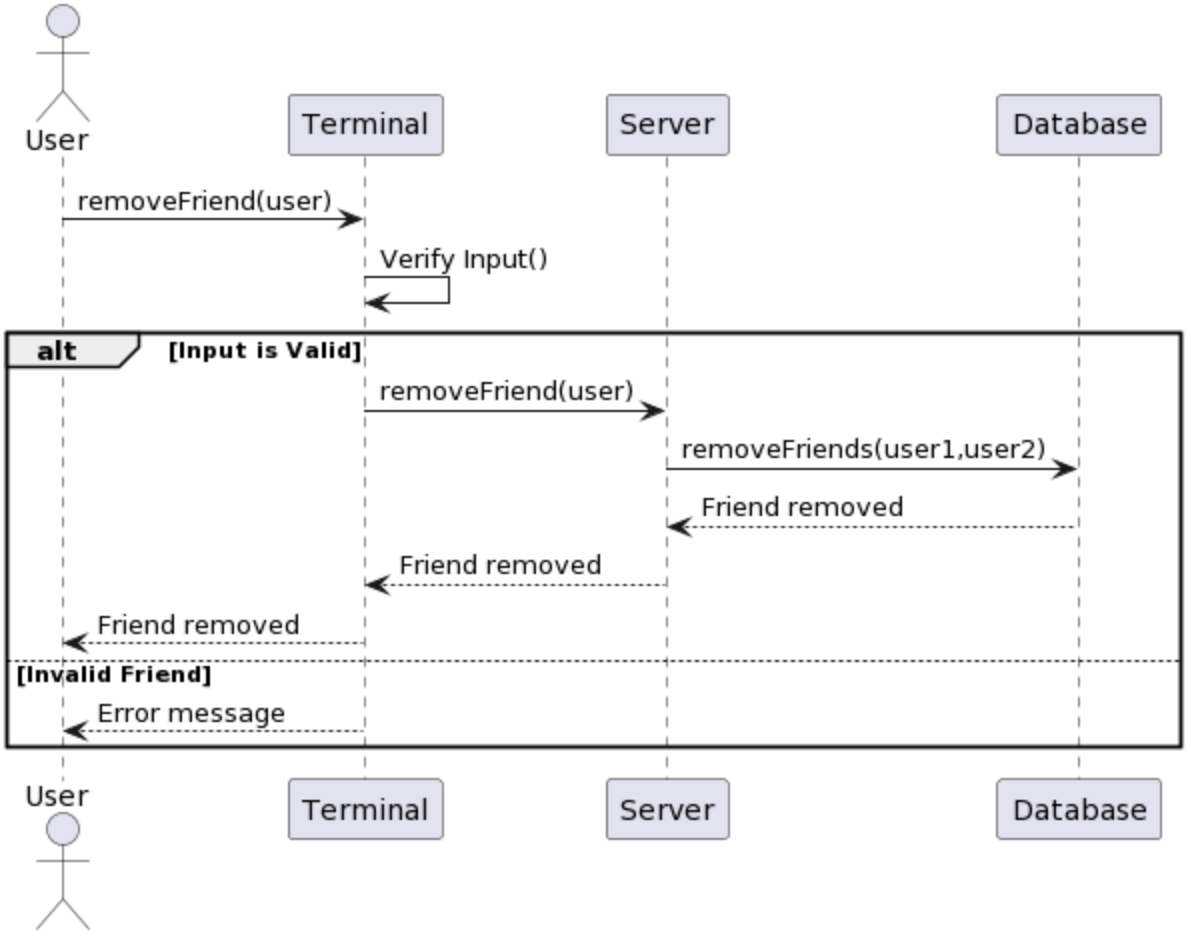

```
title  Add Friend
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : addFriend(user)

Terminal -> Terminal : Verify Input()
alt Input is Valid
Terminal -> Server : addFriend(user)
Server -> Database :  addFriends(user1,user2)
Database --> Server : Friend added
```

```
Server --> Terminal : Friend added
Terminal --> User: Friend added
else Invalid Friend
Terminal --> User : Error message

end
```

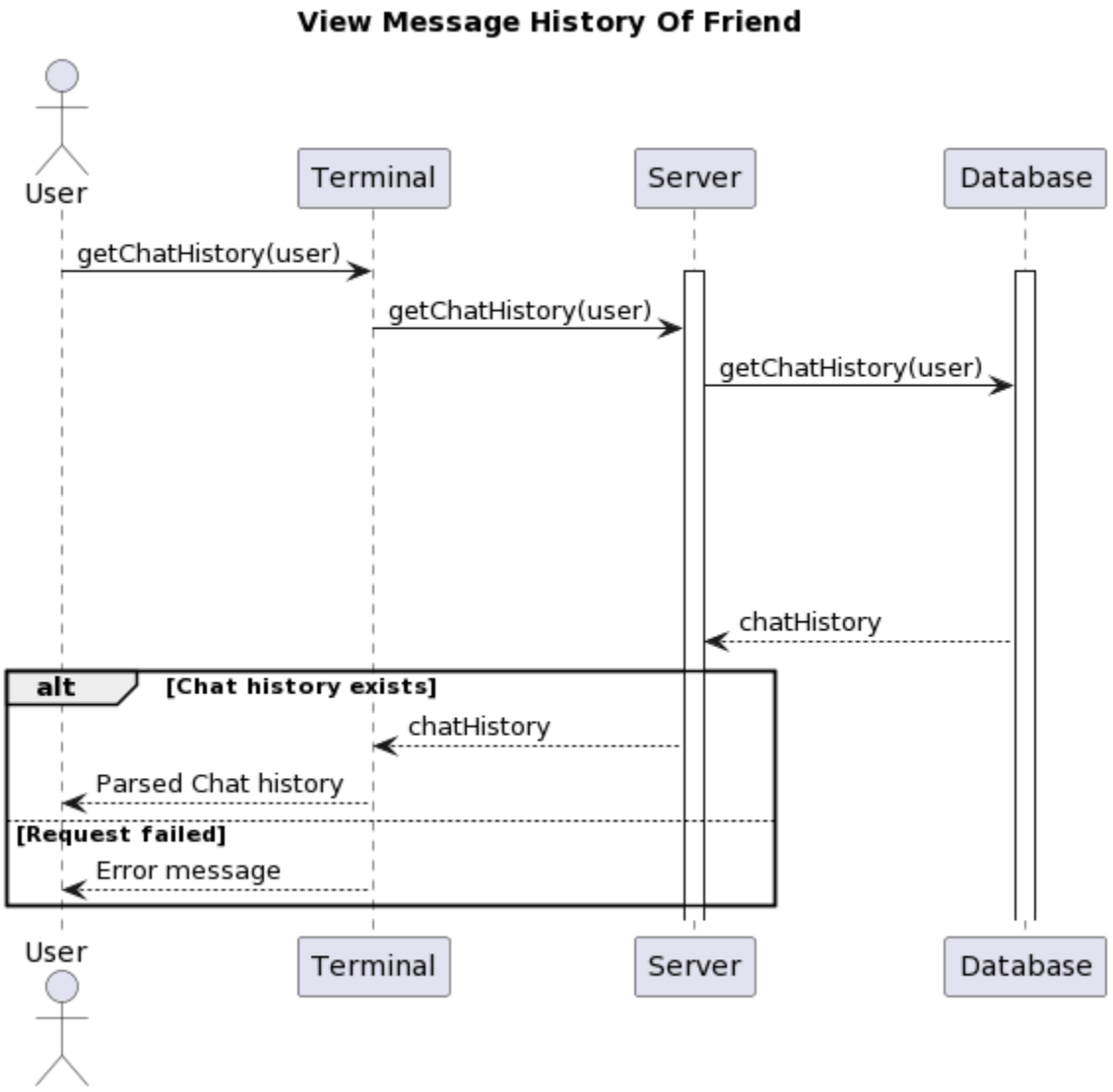
Remove Friend

```
title Remove Friend
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : removeFriend(user)

Terminal -> Terminal : Verify Input()
alt Input is Valid
Terminal -> Server : removeFriend(user)
Server -> Database :  removeFriends(user1,user2)
Database --> Server : Friend removed
Server --> Terminal : Friend removed
Terminal --> User: Friend removed
else Invalid Friend
Terminal --> User : Error message

end
```

Remove Friend



View Message History Of Group/Friend

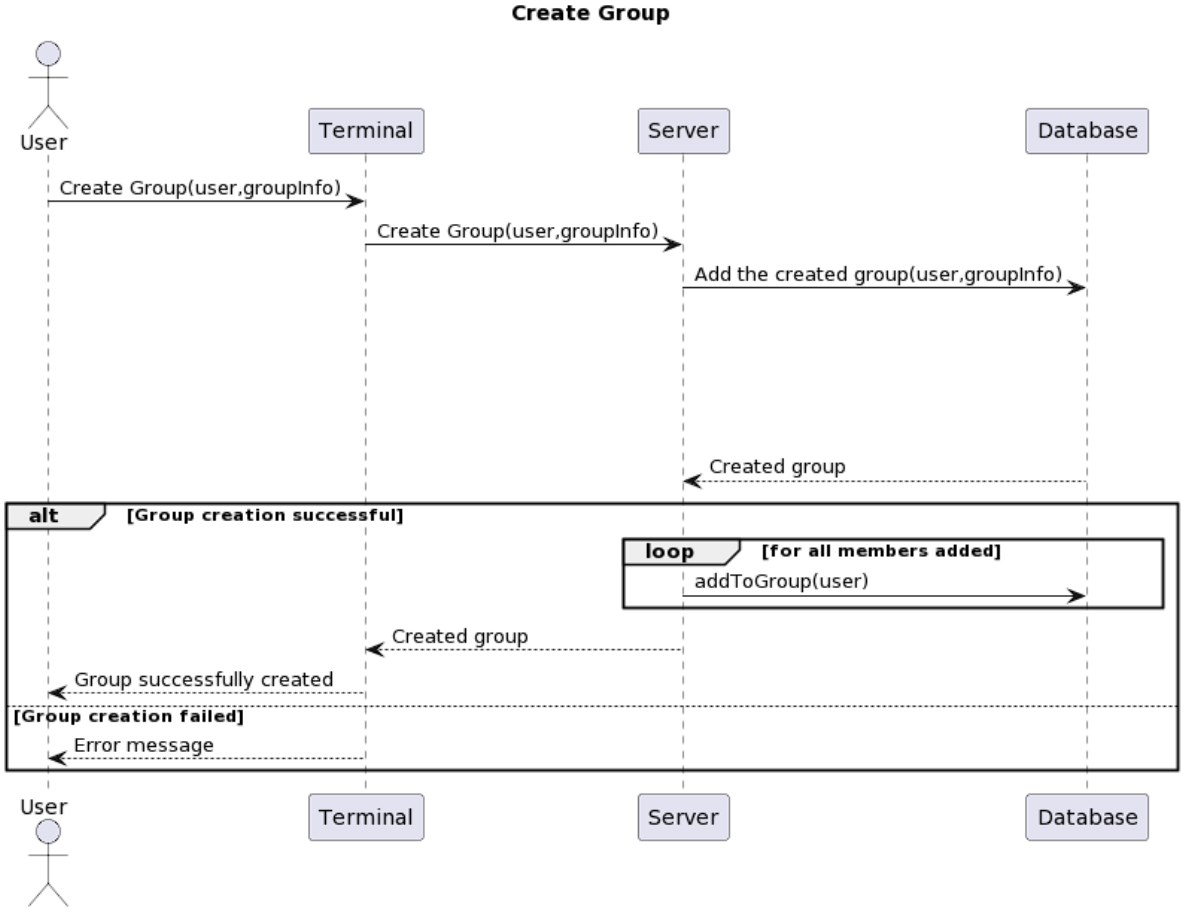



```
title View Message History Of Friend
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : getChatHistory(user)
activate Server
activate Database

Terminal -> Server : getChatHistory(user)
```

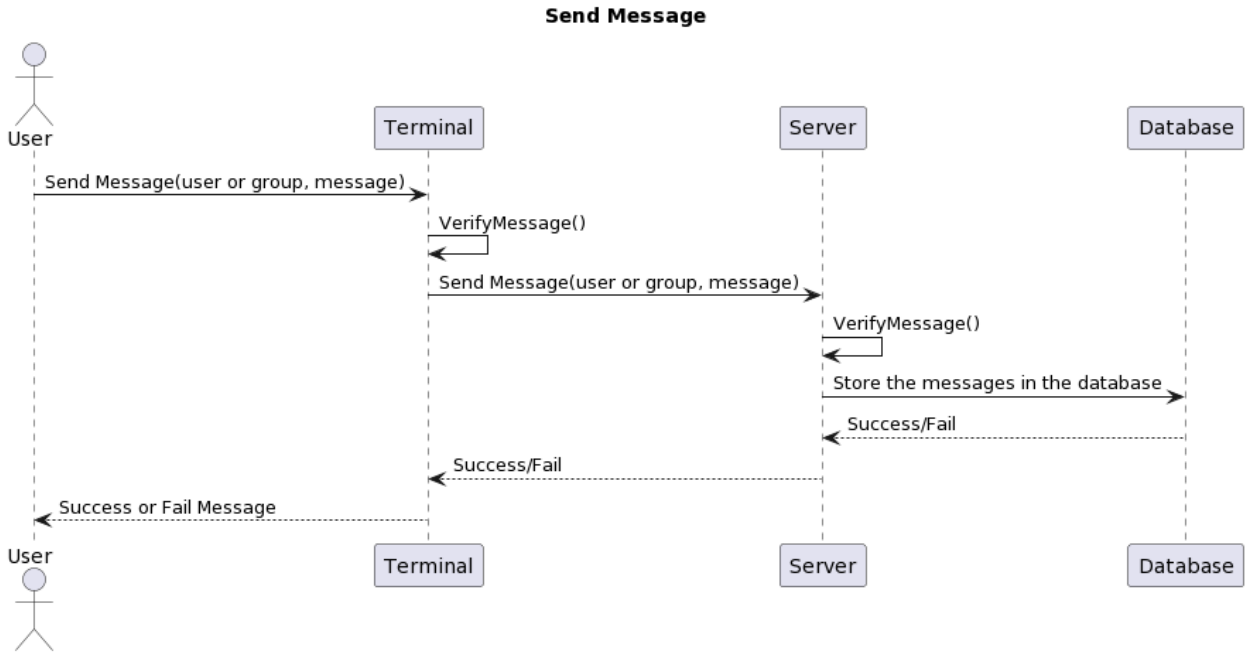
```
Server -> Database :  getChatHistory(user)
||100||
Database --> Server : chatHistory
alt Chat history exists
Server --> Terminal : chatHistory
Terminal --> User: Parsed Chat history
else Request failed
Terminal --> User : Error message

end
```

Create Group

```
title Create Group
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Create Group(user,groupInfo)

Terminal -> Server : Create Group(user,groupInfo)
Server -> Database :  Add the created group(user,groupInfo)
||100||
Database --> Server :Created group
alt Group creation successful
loop for all members added
      Server-> Database: addToGroup(user)
end
Server --> Terminal : Created group
Terminal --> User: Group successfully created
else Group creation failed
Terminal --> User : Error message

end
```


Create Group
User
Terminal
Server
Database
Create Group(user,groupInfo)
Create Group(user,groupInfo)
Add the created group(user,groupInfo)
Created group
alt
[Group creation successful]
loop
[for all members added]
addToGroup(user)
Created group
Group successfully created
[Group creation failed]
Error message
User
Terminal
Server
Database

Send Message to DM/Group

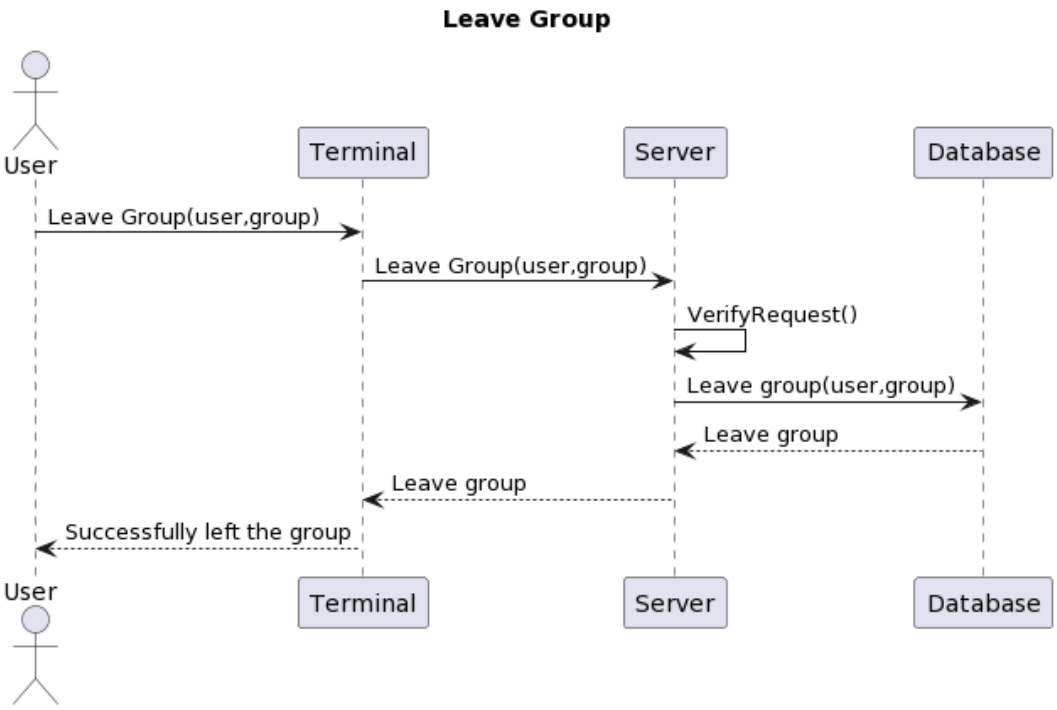



```
title Create Group
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Create Group(user,groupInfo)
activate Server
activate Database

Terminal -> Server : Create Group(user,groupInfo)
Server -> Database :  Add the created group(user,groupInfo)
||100||
Database --> Server :Created group
alt Group creation successful
loop for all members added
      Server-> Database: addToGroup(user)
end
Server --> Terminal : Created group
Terminal --> User: Group successfully created
else Group creation failed
Terminal --> User : Error message
```

```
end
```

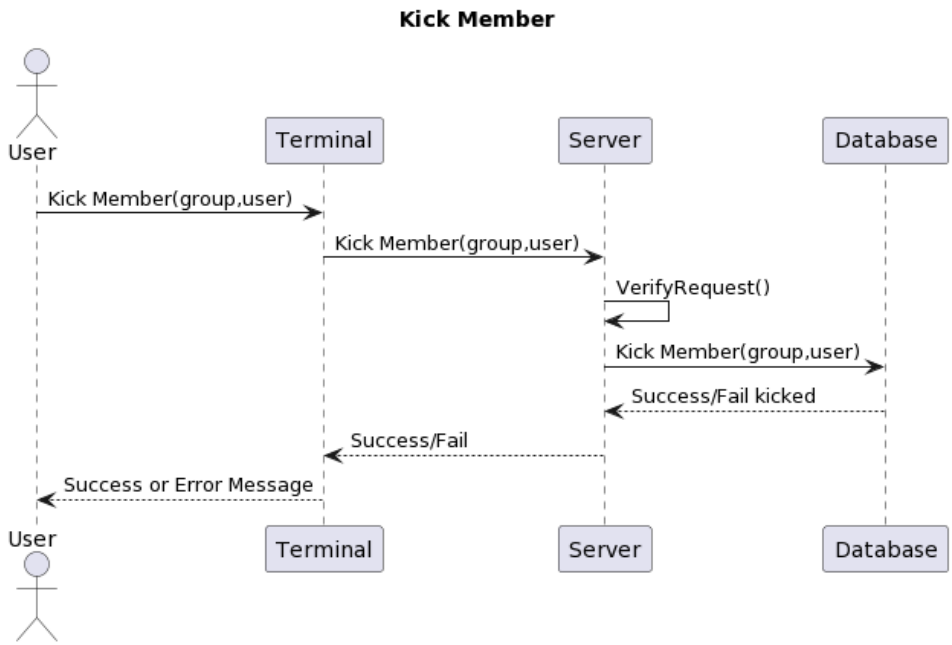
Leave Group

```
title Leave Group
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Leave Group(user,group)

Terminal -> Server : Leave Group(user,group)
Server-> Server:VerifyRequest()
Server -> Database :  Leave group(user,group)
Database --> Server :Leave group
Server --> Terminal : Leave group
Terminal --> User: Successfully left the group
```



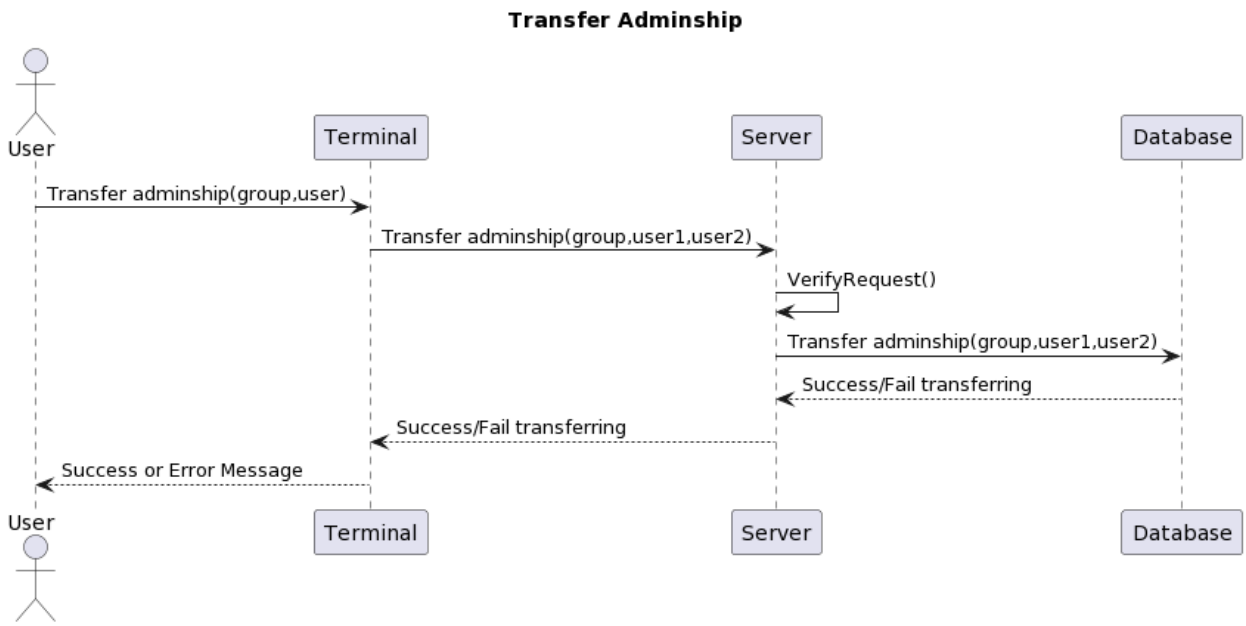
Kick Member

```
title Kick Member
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Kick Member(group,user)

Terminal -> Server : Kick Member(group,user)
Server-> Server:VerifyRequest()
Server -> Database : Kick Member(group,user)
Database --> Server : Success/Fail kicked
Server --> Terminal : Success/Fail
Terminal --> User: Success or Error Message
```



Transfer Admin

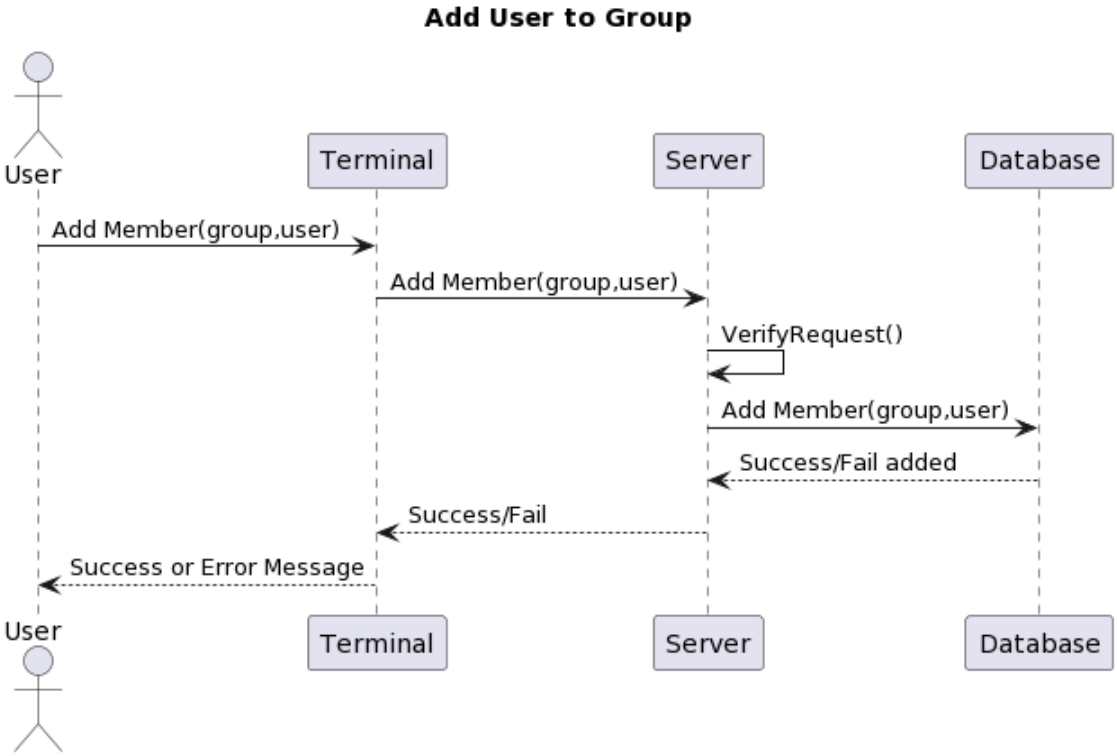
```
title Transfer Adminship
actor User
participant Terminal
participant Server
participant Database

User -> Terminal : Transfer adminship(group,user)

Terminal -> Server : Transfer adminship(group,user1,user2)
Server-> Server:VerifyRequest()
Server -> Database : Transfer adminship(group,user1,user2)
Database --> Server : Success/Fail transferring
Server --> Terminal : Success/Fail transferring
Terminal --> User: Success or Error Message
```



Add User

```
title Add User to Group
actor User
participant Terminal
```

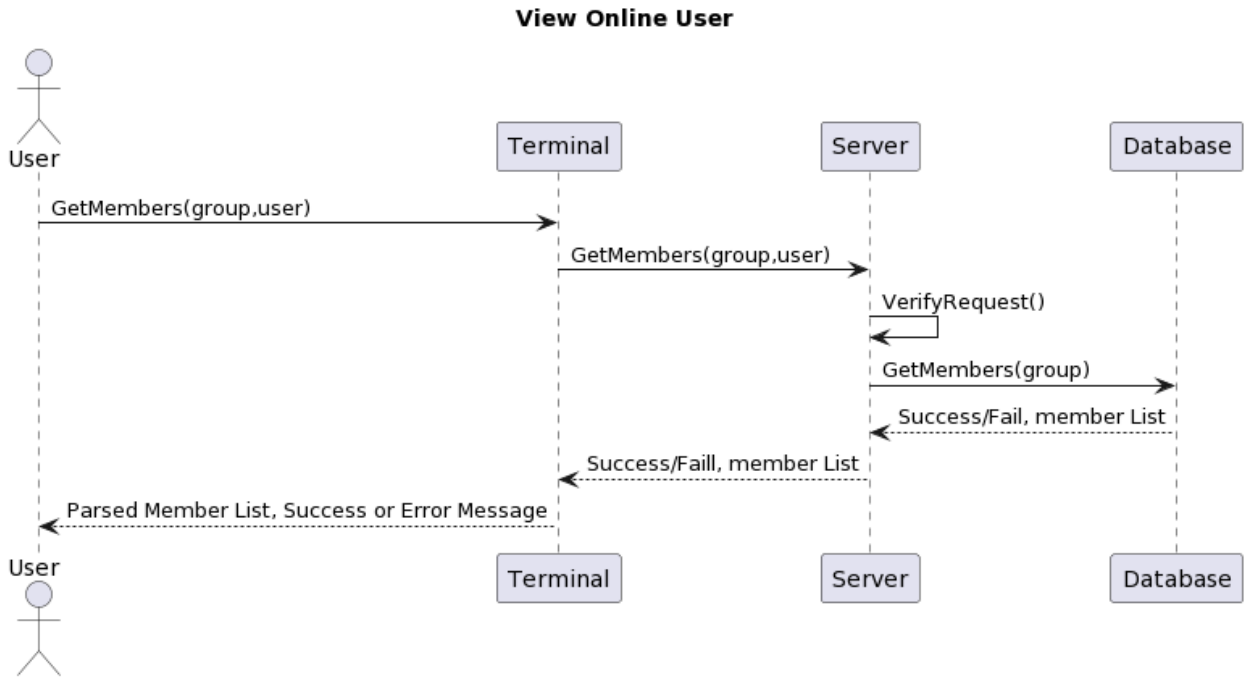
```
participant Server
participant Database

User -> Terminal : Add Member(group,user)

Terminal -> Server : Add Member(group,user)
Server-> Server:VerifyRequest()
Server -> Database : Add Member(group,user)
Database --> Server : Success/Fail added
Server --> Terminal : Success/Fail
Terminal --> User: Success or Error Message
```



View Online Users

```
title View Online User
actor User
participant Terminal
```

```
participant Server
participant Database

User -> Terminal :GetMembers(group,user)

Terminal -> Server :GetMembers(group,user)
Server-> Server:VerifyRequest()
Server -> Database : GetMembers(group)
Database --> Server : Success/Fail, member List
Server --> Terminal : Success/Fail, member List
Terminal --> User: Parsed Member List, Success or Error Message
```



Forgot Password

```
title Forgot Password
actor User
participant Terminal
participant Server
```

```
participant Email
participant Database

User -> Terminal : forgotPassword(email)
activate Server
activate Email
activate Database

Terminal -> Server : forgotPassword(email)
Server-> Server:VerifyRequest()
Server -> Email : SendEmail(resetPasswordLink)

User -> Terminal : request(resetPasswordLink)
Terminal -> Server : request(resetPasswordLink)
Server --> Terminal : (in)ValidLink
Terminal --> User : (in)ValidLink
User -> Terminal : changePassword(newpw,link)
Terminal -> Terminal : validateInput(newpw, link)
Terminal -> Server : changePassword(newpw, link, user)
Server -> Server : validateRequest()
Server -> Database : changePassword(user,newPw)

Database --> Server : Success/Fail changed Pw
Server --> Terminal : Success/Fail changed Pw
Terminal --> User: Success or Error Message
```

Forgot Password