Design Documentation

for

LFG-IRL

"Looking For Group - In Real Life"

Prepared by: The General

Beardmore, Zachary; Cai, Yide; Day, Brennan;

Dunn, Korbin; Hester, Travis; Jaiswar, Rahul;

Kresta, Cody; Macjones, Grace; Ortiz, Liliana;

Seddon, Richard; Bhandari, Shradha; Roberson, James

11/1/2017

1. You must submit a functional design, static data structured design, dynamic data structured design, and object oriented design. You may submit additional design approaches, if desired. Your design document must compare and contrast the designs using technical metrics to establish the good, the bad, and the ugly for each design. Metrics should include (but not be limited to) cohesion, coupling (data and control), Miller's Law, Graciunas' Law, factoring, scope of effect, scope of control, utilization of black boxes, and fan-in (etcetera). Clearly state which design has been selected for the implementation phase and why it was selected.

Cohesion:

Top-Down Design: Communicational and Sequential cohesion.

Jackson's Technique: Communicational and Logical cohesion.

Dynamic DataFlow: Procedural, Communicational, and Sequential Cohesion.

Object Orient Design: Communicational and Functional Cohesion.

Final Thoughts on cohesion: While all designs exhibit cohesion, OOD has a higher cohesion than our other designs. The Jackson's Design we made had the least desirable amount of cohesion.

Coupling:

Top-Down Design: The produced top-down exhibits a high degree of coupling.

Jackson's Design: Exhibits a lower degree of coupling.

Dynamic DataFlow: Exhibits a mild degree of coupling.

OOD: Exhibits a high amount of coupling.

Final thoughts: Jackson's designs exhibit the lowest degree of coupling while the produced Top-Down and OOD exhibit high degrees of coupling. The lower the degree of coupling the better the resulting design is.

Miller's Law:

Top-Down: Follows Miller's law but begins to violate this at lower levels of the design.

Jackson's: At certain stages of the design modules exhibit more than one idea.

Dynamic DataFlow: Follows Miller's Law well with 1-2 ideas per module.

OOD: Modules or classes in the OOD sometimes contain more than 7 ideas at a time.

Final Thoughts: Out of all the produced designs Dynamic Data Flow seems to follow Miller's Law more closely with 1-2 ideas per module. While this is not perfect it is the best out of all produced designs so far.

Graciunas Law:

Top Down: Graciunas is rolling in his grave with how bad the top-down design is. His rolling corpse could power a small village until his corpse loses most of his mass while spinning so violently.

Jackson's: The generally of the produced Jackson's technique does not make the amount of relationships clear. People may assume data is needed for certain computations and it is not and vice versa.

Dynamic DataFlow: Follows Gracinuas' Law quite well with how data relates to other modules down the line.

OOD: The relationships are clear but the amount of relationships may make some things confusing.

Final Thoughts: Dynamic Data flow is the preferred design in terms of Graciunas' Law.

Factoring:

Top Down: Mid levels are factored well but the at lower levels the design fans in.

Jackson's: Jackson's is quite well factored.

Dynamic Data Flow: Dynamic Data flow is also factored well

OOD: Interconnected but still mildly factored.

Final Thoughts: Jackson's and Dynamic Data Flow have achieved desirable levels of factoring.

Scope of effect and control:

Top-Down: Follows the idea of scope of control and scope of effect. Upper Level modules have control over lower ones, but this design leaves some to be desired in terms of both

Jackson's: Jackson's technique also follows the ideas of scope and effect quite well for some of the modules, but not all. The module get group record is a good example of this, as the lower level modules are commanded by the upper level module. Other modules do not exhibit this behavior.

Dynamic Data Flow: Dynamic data flow does not exhibit the traditional scope of effect/control as other designs. It mainly focuses on where the data is going, not what is happening to it.

OOD: OOD, like Jackson's technique, exhibits scope of effect/control to some state. It is more detailed that Jackson's in regards to data and what is needed, but has multiple ideas per modules.

Final thoughts: Top down follows the criteria the best, followed closely by OOD and Jackson's with Dynamic data flow coming in last with regards to scope of effect/control.

Black Box:

Top-down: Top-down does not utilize a high number of black boxes, unless you were to count the end routine and the chat routine.

Jackson's: Jackson's design exhibits a number of black boxes including the modules chat, get new group, and get existing group.

Dynamic: Dynamic data flow incorporates black boxes in the modules chat, log out, and drop.

OOD: The chat module is the only black box in the OOD design. We would like to include the data base handler, however it is a grey box, as its function depends on the input parameters it receives.

Final thoughts: Top down is obviously not the choice in regards to black box implementation, and OOD is not that far ahead either. Jackson's is possibly the best in regards to the use of black boxes, however Dynamic Data Flow is not that far behind.

Fan in:

Top-Down: The top down design exhibits a high degree of fan out, but little degree of fan it. The only modules that exhibit a high degree of fan it is the end module.

Jackson's: Jackson's design also features a high degree of fan out, with no degree of fan in. All the lower levels are mainly get modules, and do not need to continue down another level.

Dynamic Data Flow: Dynamic Data Flow implements a small amount of fan in with the modules get user info and get database fanning into the module process account. Further down, this design again exhibits a high degree of fan out, not fan in.

OOD: Like Jackson's design, the OOD design has a high degree of fan out and little to no fan it. It's low level modules are like Jackson's design in that they are getters, not passers, so no further decomposition was necessary.

Final thoughts- In regards to fan in Dynamic is the best choice though not in high degree, followed by Top-down. The worse choice for fan in is Jackson's with OOD being just a little better.

Conclusion:

Dynamic Data Flow is the design we will choose to follow for implementation. Dynamic Data Flow seems to be most desirable on most metrics. While not the best according to some metrics it does not consistently place as the worse design in any metric. Also, it does not leave much of implementation to the imagination. Everyone will know what data goes where and will minimize "freelancing'.

4. You must explicitly indicate how race conditions have been avoided. Note this implies the specification is sufficiently challenging to create potential race conditions. If the specification is inadequate for the task, modify the specification. Include the problem description and solution in your design document (for a grade of "B" or better).

The potential for race condition will be high, since multiple processes will have the potential to access the data at the same time, it is to say multiple processes will be able to update data at the same time. It will be reasonable if the processes meet the determinate requirements to enter critical section and be able to update data. Therefore, in order to avoid race condition, database must be normalized and a cache management would be implemented to store key values and in this way gain access to keys immediately as required by the application, the purpose to do this would be to pair keys appropriately to prevent data from being inconsistent, thus, data will be updated uniformly. The application will enforce transaction atomicity to prevent race condition, so every transaction and process should be synchronized and lock till it may be called again and be allowed into critical section. A second option was discussed to prevent race condition in case of excessive traffic in the application; it is a technique called "Gumball Technique" (GT) it work with an augmented cache and by extending simple operations like delete, get and put.

On the other hand, to avoid multithreading race conditions each process will be synchronized to its own block allowing race conditions to be avoided. We are using java to implement our design, hence Java programming language provides a very handy way of creating threads and synchronizing their task by using synchronized blocks. You keep shared resources within this block, we are using locks as synchronization methods so, we can add a lock that protects each module.

The first approach to prevent race condition will be avoiding it once it is detected throughout the implementation face. A specific example where we may face the race condition will be in the group creation and when join into a group, since multiple users will be able to create groups and join into groups at the same time, so such action will only be committed when pre established conditions have met. Regarding the group creation no repetitive groups will be allowed, for the same geographical area, so in this way only groups that meet the criteria will be allowed to modify the groups listing, so once a group is no longer available to join, then a new transaction will be allowed to create a group. Since the possibility of many new groups on the same subject may be created at the same moment in time only one transaction will be allowed for that particular criteria to access the critical section.

Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

Problems often occur when one thread does a "check-then-act" (e.g. "check" if the value is X, then "act" to do something that depends on the value being X) and another thread does something to the value in between the "check" and the "act". E.g: An example of a race condition in regards to our project would be if two groups are created at similar times, we do not want to assign them the same group number, so we will use the methods listed above to avoid such conditions.

5. Explicitly explain all potential deadlocks for data access and how you avoid them to obtain a grade of "B" or better. Again, if your original specification is insufficient to exhibit deadlock modify the specification. Trivial specifications do not meet minimum requirements. For a deadlock to occur, you must have at least two related documents over which separate processes/threads may compete, e.g., an invoice and related purchase order, shipping record, or inventory transaction. Include the problem description and solution in your design document. Indicate how to avoid your specific deadlock using database or file system features (discuss another product if the database used your implementation does not have deadlock control), avoidance via programming, and management techniques (consider job scheduling)

We anticipate the potential for deadlock occurrence during the following scenarios:

1. A user is attempting to perform a group search/filter (read from the Groups table) while a separate user is attempting to create a new group, or a user is being added to a group (write to the Groups table)
2. A user is attempting to search for a second user's account (read from the Accounts table) while a third user is attempting to create an account (write to the Accounts table)

Deadlocks will be avoided by forcing each transaction to be read-committed and by locking tables when write operations are being performed. When tables are locked, they are in use by another transaction and cannot be accessed. Since we are using an SQL database, in the event of a lock preventing a transaction's access to a table an SQL LOCK EXCEPTION will be thrown. This exception will trigger a WAIT command for the transaction that is attempting to lock information that has already been locked by another transaction. This WAIT command will terminate the transaction for a specified amount of time at the application level. Once the WAIT command has expired, the action that triggered the exception will be attempted once more. This exception – wait – try loop will occur until the lock has been resolved.

To ensure the rapid completion of transactions, and therefore release of locks, we will ensure that all queries accessing multiple tables will access said tables in the same order. This way the lock transaction A (which needs access to table X and Y) has on table X is detected before transaction B (also needing access to table X and Y) can lock table Y as it has already seen the lock on table X and has been issued a WAIT command.

6. Supply Entity-Relationship diagrams for your data showing all data, keys, foreign keys, one-to-one, one-to-many, and many-to-many relationships. As part of your design document, explain specifically why your database or file system is at least in third normal form. Boyce-Codd normal form or higher would be preferred (to attain a grade of "B" or better).

Schema:

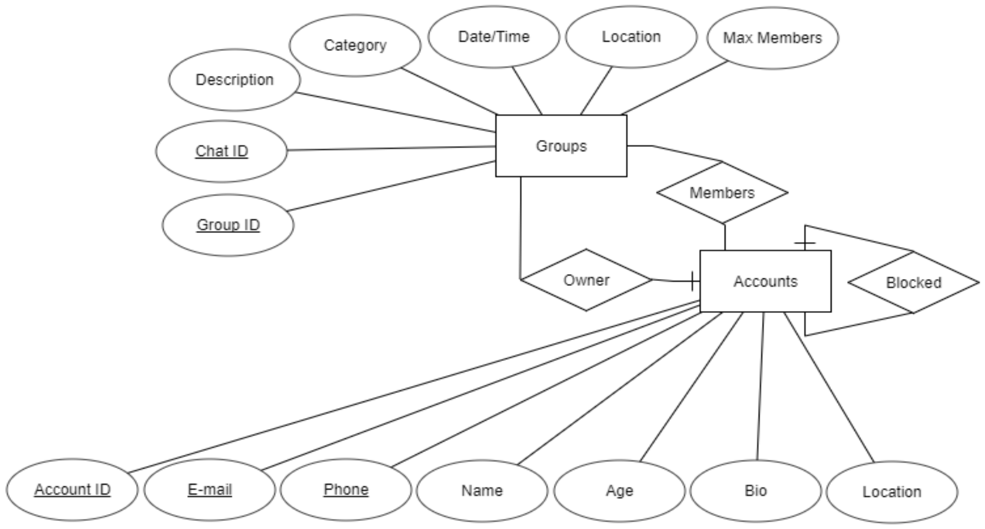
Groups(<u>Group Id</u>, <u>Chat Id</u>, Category, Location, Date/Time, Description, Max Members)

Members(<u>Group Id</u>, <u>Account Id</u>)

Owner(<u>Group Id</u>, <u>Account Id</u>)

Account(<u>Account Id</u>, <u>E-mail</u>, <u>Phone</u>, Name, Age, Description, Location)

Blocked(<u>Account Id</u>, <u>Account Id</u>)



Functional Dependencies:

Groups: Group Id, Chat Id → Category, Location, Date/Time, Max Members, Description

Members: Group Id, Account Id → Group Id, Account Id

Owner: Group Id, Account Id → Group Id, Account Id

Account: Account Id, E-Mail, Phone → Name, Age, Description, Location

Blocked: Account Id, Account → Account Id, Account Id

In the database design presented above, for every non-trivial functional dependency in a relation, the left hand side is a superkey for that relation. There are no direct relationships between the attributes that aren't represented in the structure of the schema.

8. **As part of the design, consider user Remote Procedure Calls (RPC), Servlets, and multithreaded Sockets for implementation. Convenience me you understand the difference between technologies. Explain why you have selected the technology you will utilize during the implementation phase.**

Remote Procedure Calls - Client-server model. The client is the requesting program and the program providing the service is the server. RPC is a synchronous operation and requires the requesting program to be suspended until the results of the remote procedure are returned. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

Multi threaded Sockets - Sockets create connections between the server ports and the client ports. Upon acceptance, the server gets a new socket and has an endpoint set to the address and port of the client. It needs a new socket so that it can listen to the original socket for connection requests and process the needs of the client. On the client side, if the connection is accepted, a socket is created and the client can use the socket to communicate with the server. The client and server communicate by writing or reading the sockets. With multi threaded sockets, we assume each socket will have its own thread ideally.

Servlets - A servlet is a Java utility that helps with the creation of web applications. Prior to the implementation of servlets, CGI, or common gate interface was the the popular server programing language. A web container create threads for handling multiple requests to the servlets, so it is fast because each request gets it's own process. It is also managed by the JVM so memory leaks and garbage collection is automatic.

For our implementation, we will use multi-threaded sockets to communicate between the clients and servers, for the simple fact that a majority of the group has experience with sockets and as a group are more confident in a system we know.

9. Your design document must specify how the final product will be tested and how the customer will measure the result for acceptance.

When the project is ready to be used in beta, we will have “test subjects” who will make an account using the app. Then after they’ve become members, they will go through a quick tutorial on how to use the application that will take no longer than 20 seconds. New users will have to verify their email address and/or phone number. Furthermore, our test subjects will give us feedback after using the application for one day, then we will record their feedback using the scale below. This will measure their satisfaction and feeling of ease when using the app. **The design will encompass a data flow which can be triggered through user actions, starting from a main module.**

Rate your Satisfaction	Scale
1	Poor
2	Average
3	Decent
4	Good
5	Excellent

The design only specifies the process on which data flows through modules. It shows multiple actions the user may choose from so he/she can feel a sense of autonomy. Furthermore, when we implement the review, it has its own module from the main component. In this module, the user will be able to review the program and offer feedback. While the user uses the app, a random pop-up will appear occasionally to ask for users review, like the ones seen on common apps like Tinder which asks the user to rate their application from 1-5 stars. During the testing stage we will also have error messages that gives the user the option to send a report, and since we are just testing the application, the test subjects are required to send the report.

There are typical quality measures that our design seeks to implement. The quality is used to determine the user’s acceptance. For this app, we will perform:

MTBF(*Mean-Time-Between Failure)*: It will determine how long on the average does the application function properly before encountering an error. This is weighted by the number of processes in the background and the application size and cumulative actual time of use.

MTTR*(Mean-Time to Repair)*: When the application does encounter an error, we will determine how long it takes on average to fix it, whether if the user can repair it or not.

10. Your design document should indicate the potential for optimization for space and optimization for time.

After the system has been completely designed, the potential for optimization can begin. Local optimization has been considered however we do not want to slow the system down hence have opted for global optimization due to the efficiency gains and less costs with a main focus on highly cohesive modules with low coupling.

Optimization of this system is not limited and will involve the following considerations:

1. Attempt to uniformly distribute execution time over the code. Some modules such as the group module with more code will take a longer time to execute hence the goal is to minimize CPU running time of these modules to get rid of bottlenecks during execution of real data. After determining the execution time for each module, each module will be examined and its potential for improvement will be determined. Since the Group module will be the biggest, it will have the longest runtime. This module will be the central focus so that sorting times and information retrieval are minimized.
2. The entire program should be running with minimum errors for optimization to be relevant. To prevent extra bugs from occurring, the system will have to be designed properly to aid optimization of time. Code clarity is of utmost importance because the simplest way is the most efficient.
3. The cost for improvement must be estimated. Changing to a different search algorithm may greatly increase complexity and processing time hence it may not be justifiable if the current system meets requirement standards.
4. The highest priority module is the group module. This module will be the central focus for improvement with the goal of improving performance for the least cost. This module will also be the first module to be optimized.
5. The use of external routines will be implemented and compiled at separate times. The purpose of this is so that the code in these routines can be shared with other programs. For example, the group module will contain code that can also be used for output in the updated group module.
6. The external routines will be grouped by sequencing all the code in the same file for the language translator.
7. Overhead will be minimized using inline expansion. Subprograms that contain essential code will have the subprogram body copied at each location that it is referenced. This allows the translator to utilize free registers as subprograms will be compiled at different times and share a limited number of registers.
8. Data type conversions will be kept to a minimum and emphasis will be placed for using the same data type for all variables.
9. An optimizing translator will be used to look for loop invariants and subscript optimization.
10. Secure-Socket-Layers will be used to encrypt and decrypt communications between the server and client side. A connection will be made when the web server identifies itself upon request from the browser. A copy of the SSL certificate will be sent from the server to ensure that the SSL certificate is trustworthy. Upon receipt of an acknowledgement, an SSL encrypted session will be created.
11. The Gumball Technique will be used in the relational database to detect race conditions and prevent key-value pairs from becoming inconsistent with the tabular data.

11. Indicate specifically the parts of the specification and design that will lend themselves to re-use.

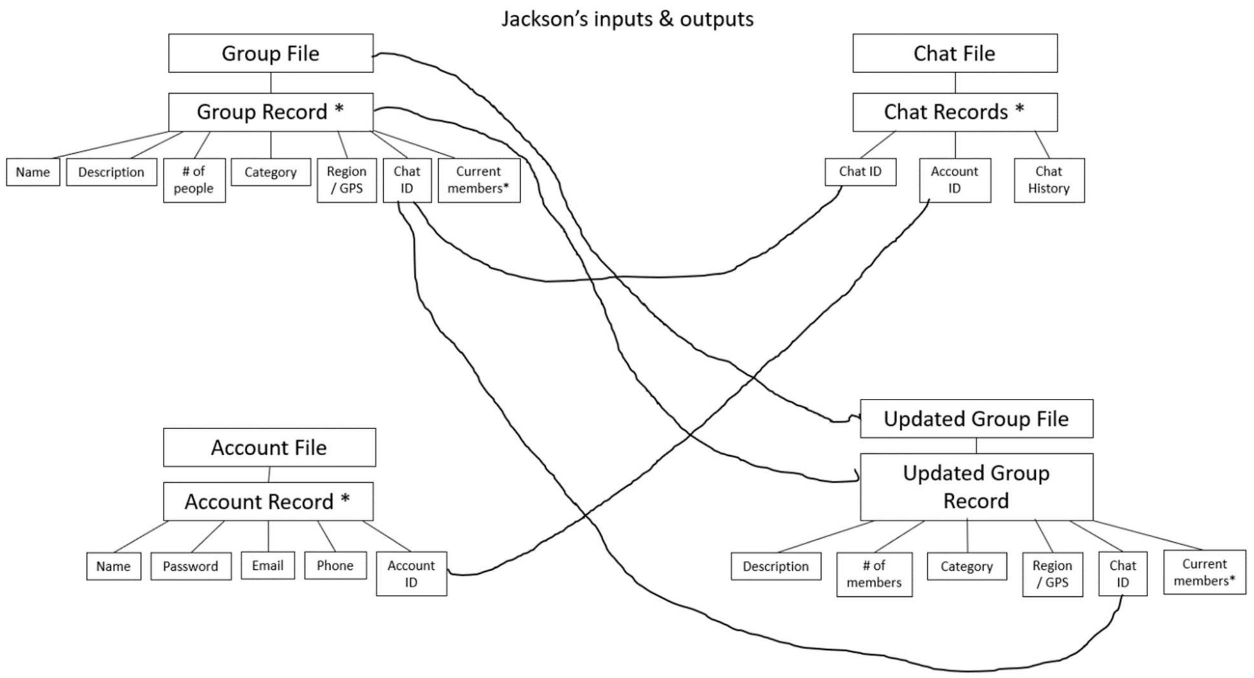
After the project is fully completed, part of the design, code (implementation), and database structure can lend themselves to reuse. In some cases, the database structure can be reused verbatim, while the code and design can be reused with minor modifications in new applications. Since our application was designed using dynamic dataflow, the modules for login and sign in could be reused in other systems that require a similar login or sign in features. Also, the RDBMS in our database design can be reused with slight modifications.

12.Your design must include an implementation plan to complete the project and present the results this semester.

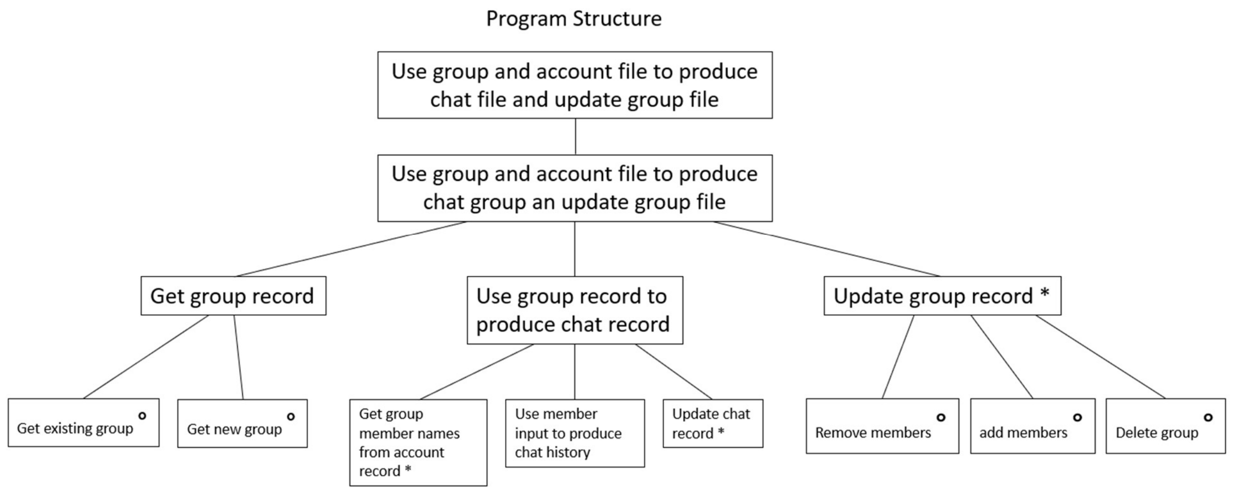
We have a time limit of 28 days to complete this application. For the first 7 days, we should work on building the database and the login function. For the second week, we should create the main modules and add samples into the database. Third week we should implement the user settings and finish the group functions. On our fourth week, the application should be able to run properly with little-to-no errors. The presentation will go over the features of the application, with importance placed on the systems to prevent deadlocks and race conditions etc., as well as include a practical demo of the software.

Jackson's Technique

Step 1:



Step 2:



Program Structure
Use group and account file to produce chat file and update group file
Use group and account file to produce chat group an update group file
Get group record
Use group record to produce chat record
Update group record *
Get existing group °
Get new group °
Get group member names from account record *
Use member input to produce chat history
Update chat record *
Remove members °
add members °
Delete group °

Step 3:

1. Access Files/DB
2. Read File/DB info
3. Close database
4. Create/update chat
5. Read chat history

6. Create group
7. Access Group
8. Users in group
9. Add a member
10. Remove a member
11. Delete a group
12. Update group record
13. Update chat
14. Get member name

Step 4

Mainline:

- Access Database
- Open Group
- Open Chat
- **GetGroupRecord**
- **ProcessChat**
- **UpdateGroupRecord**

GetGroupRecord:

- Access Group/Create Group
- Get Users

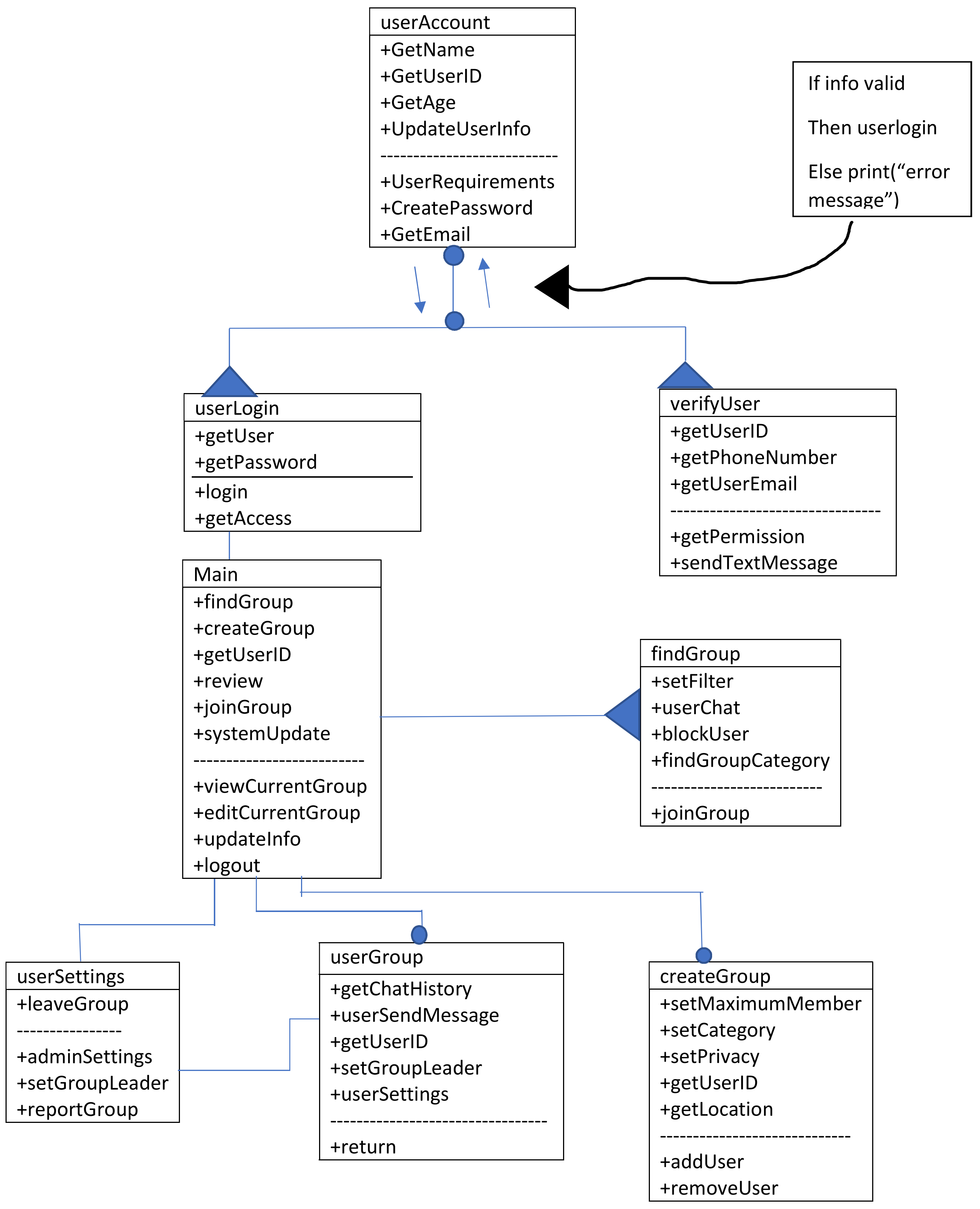
ProcessChat:

- Get member names from account table
- Read/display chat history
- Update chat

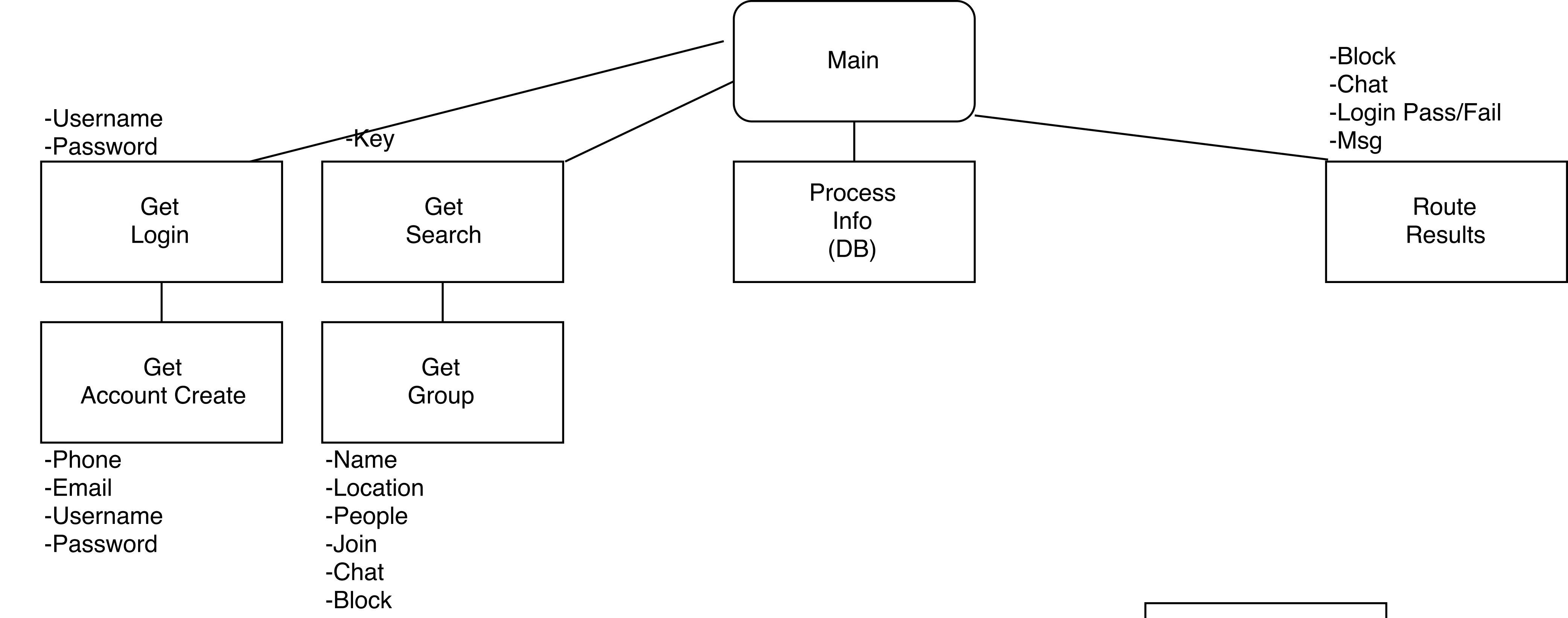
UpdateGroupRecord:

- Update Members(Add/delete)
- Update Group record(delete/update info)

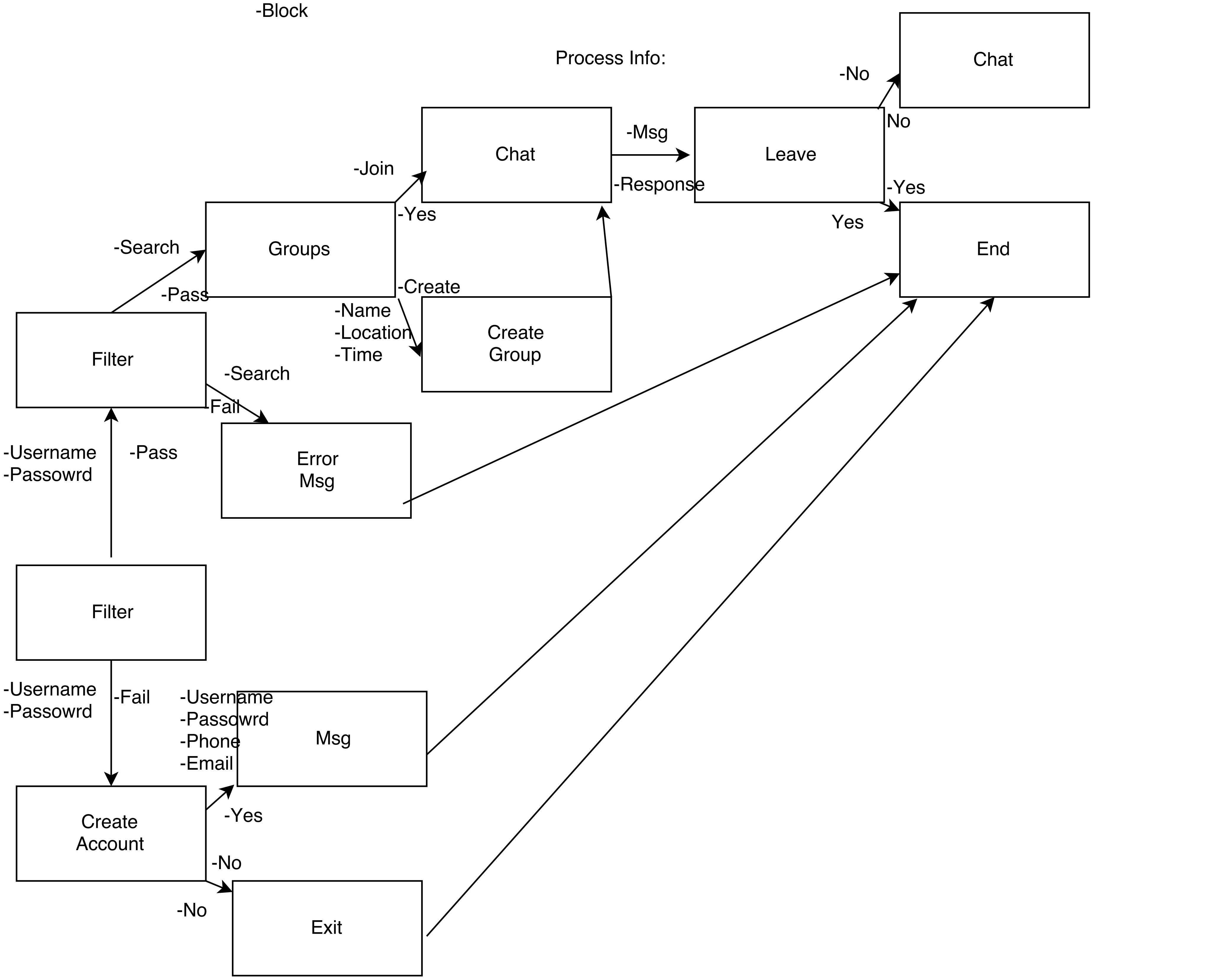
Object Oriented Design


userAccount
+GetName
+GetUserID
+GetAge
+UpdateUserInfo
+UserRequirements
+CreatePassword
+GetEmail
If info valid
Then userlogin
Else print("error message")
userLogin
+getUser
+getPassword
+login
+getAccess
verifyUser
+getUserID
+getPhoneNumber
+getUserEmail
+getPermission
+sendTextMessage
Main
+findGroup
+createGroup
+getUserID
+review
+joinGroup
+systemUpdate
+viewCurrentGroup
+editCurrentGroup
+updateInfo
+logout
findGroup
+setFilter
+userChat
+blockUser
+findGroupCategory
+joinGroup
userSettings
+leaveGroup
+adminSettings
+setGroupLeader
+reportGroup
userGroup
+getChatHistory
+userSendMessage
+getUserID
+setGroupLeader
+userSettings
+return
createGroup
+setMaximumMember
+setCategory
+setPrivacy
+getUserID
+getLocation
+addUser
+removeUser

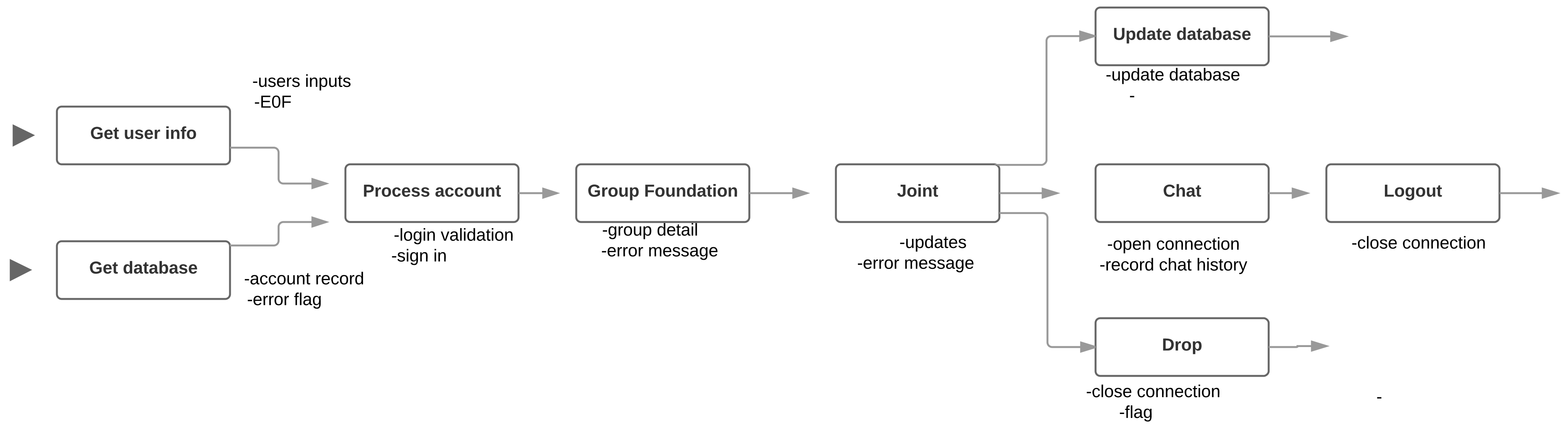
Top Down Design



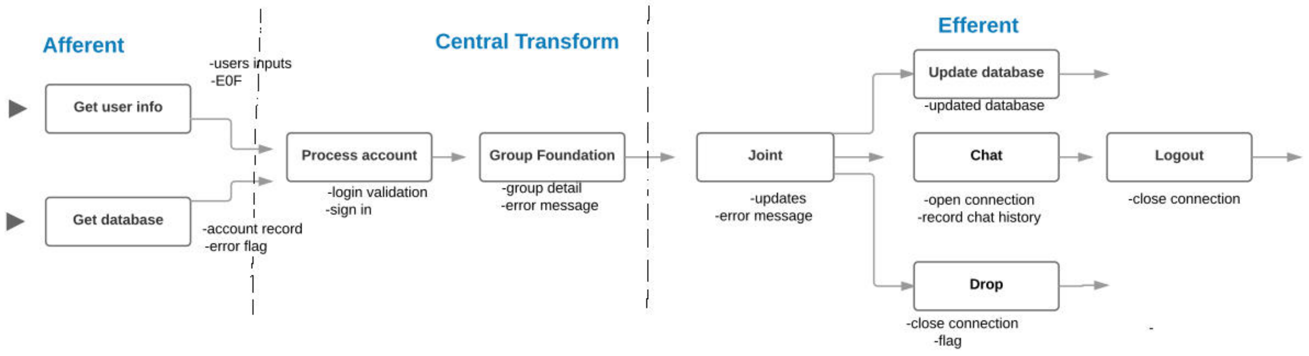
Process Info:



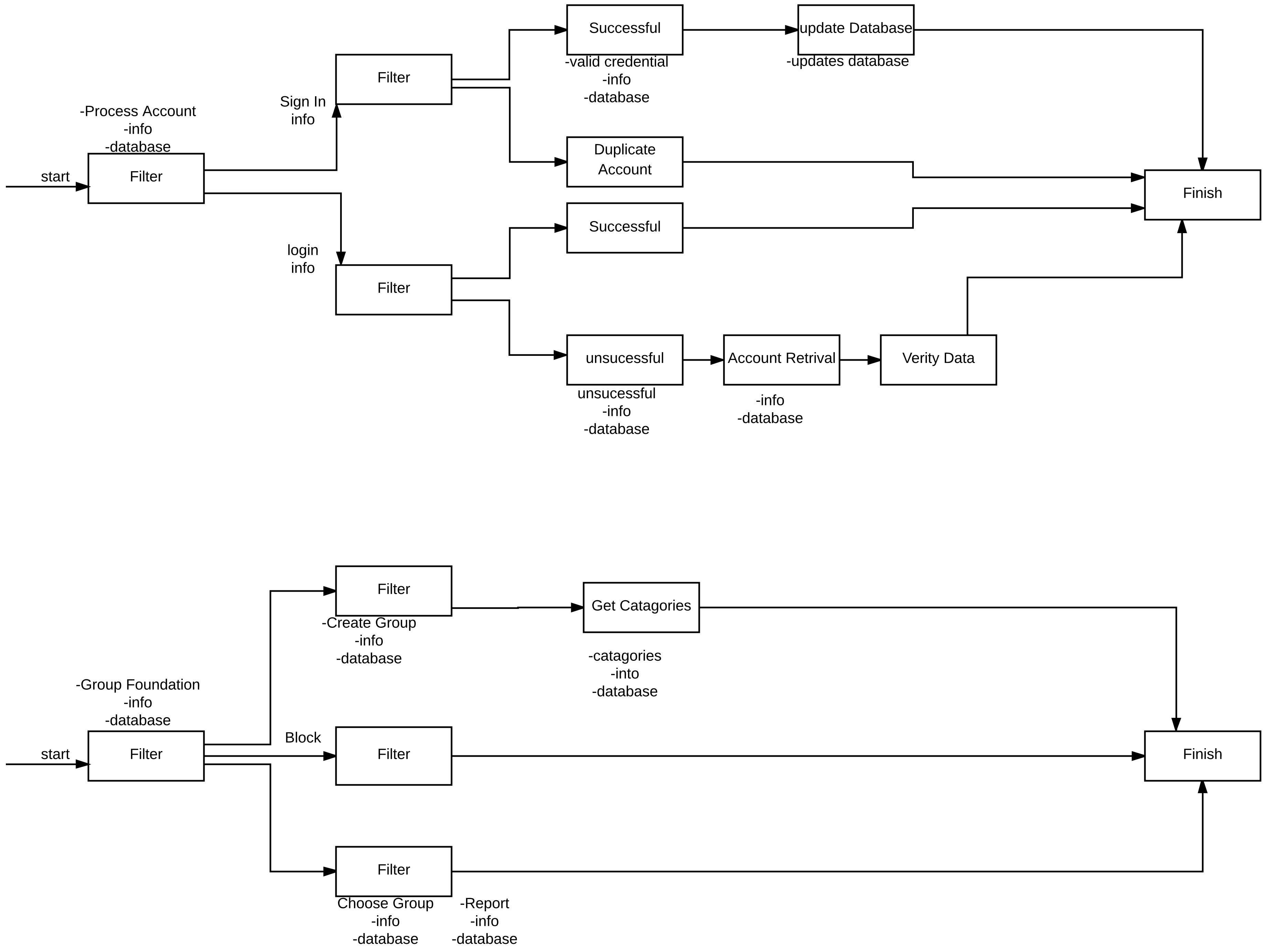
Dynamic Dataflow Analysis


Get user info
-users inputs
-E0F
Get database
-account record
-error flag
Process account
-login validation
-sign in
Group Foundation
-group detail
-error message
Joint
-updates
-error message
Update database
-update database
-
Chat
-open connection
-record chat history
Logout
-close connection
Drop
-close connection
-flag
-

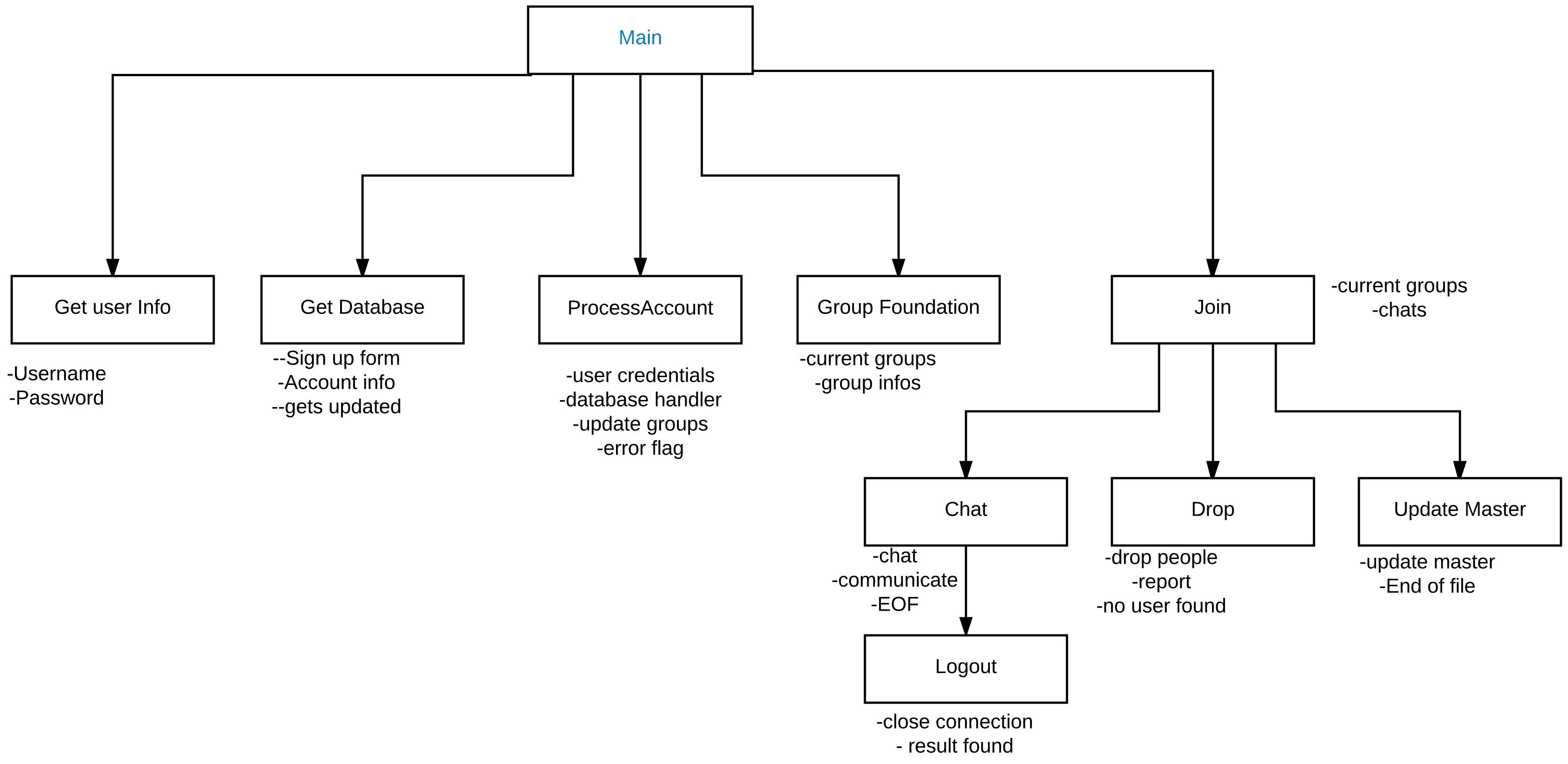
The Afferent, Efferent and Central Transforms



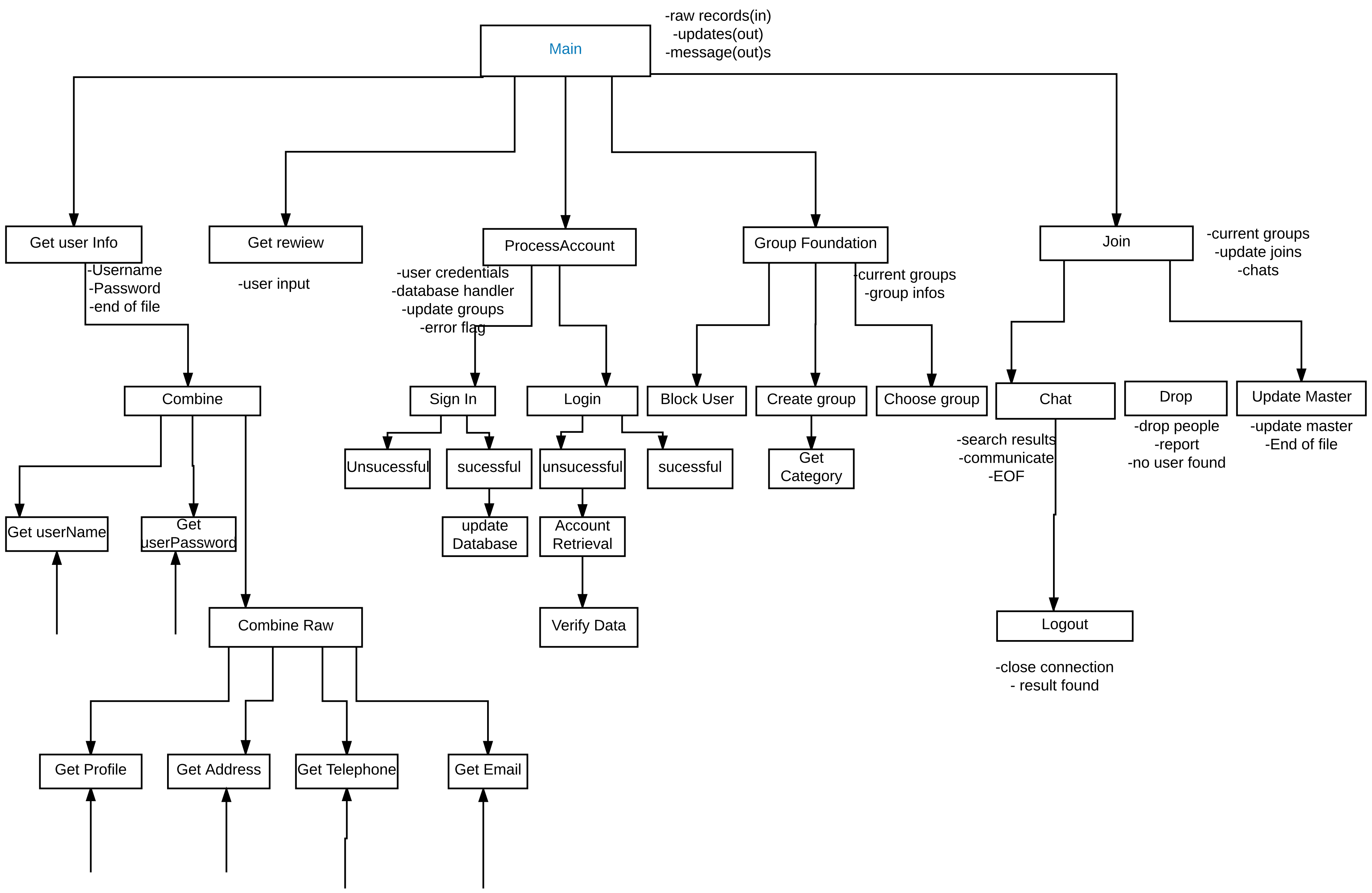
Expand Process Modules



First Level Factoring



Factored System



Main
-raw records(in)
-updates(out)
-message(out)s
Get user Info
-Username
-Password
-end of file
Get rewiew
-user input
ProcessAccount
-user credentials
-database handler
-update groups
-error flag
Group Foundation
-current groups
-group infos
Join
-current groups
-update joins
-chats
Combine
Get userName
Get userPassword
Combine Raw
Get Profile
Get Address
Get Telephone
Get Email
Sign In
Unsucessful
sucessful
update Database
Login
unsucessful
sucessful
Account Retrieval
Verify Data
Block User
Create group
Get Category
Choose group
Chat
-search results
-communicate
-EOF
Logout
-close connection
- result found
Drop
-drop people
-report
-no user found
Update Master
-update master
-End of file