Synchronized Progress Management

Design Document

Di Xu*, Junyu Chen, Weihan Ding, Su Feng & Binhao Lin

Table of Content

Purpose

This document gives detailed descriptions on many aspects of the design of project "Synchronized Progress Management (SPM)" in implementation level. In particular, it gives a brief and high level summary about the functional requirements, outlines how SPM will be implemented, states the priorities of features that this project aims to, provides detailed introduction on objective and functional designs and addresses several design issues and logistics.

Logistics and Background

Constant data synchronization between clients, the server and the database will be maintained. All client information, data, profiles will be maintained by the server, which bridges the connection between clients and the database. The interaction between Clients and the server will be maintained in an online manner. Client may still function in an offline manner to ensure certain functionalities which requires no immediate communications to the server. Server will be responsible for handling client requests and bridging communications among multiple clients.

High-level Summary of Functional Requirements

Please refer to Product Backlog for detailed and low-level functional requirements categorized by different functional modules.

- **Registration**: First time users are required to register before they can access all functionalities that SPM provides. Registration information will be inspected by the server and stored to database upon successful registration or return with error instructions. Users' login information will be encrypted.
- **Login**: User can sign in with a valid pair of user ID and password. Server will maintain the status of a client. User may choose to cache login information.
- **Logout**: Clients are logged out automatically after a certain period of inactivity or upon request.
- **Personal Task Management**: User can monitor his personal tasks, create a new Personal Task, modify and delete an existing task. Reminders will be popped if a Personal Task expires.
- **Weather Information Display (Upon User's agreement to use Google Location Service; Tentative)**: User can view auxiliary information such as weather and location.
- **Contact Book**: All other members in the same group will be added to the contact list and labeled. User can manually add or delete additional contacts. An instant messenger system allows live communication among clients.
- **Notification Center**: Clients will notify the users upon receiving any important updates from the server and direct users to designated places regarding the notification.
- **Group Management**: A Team Leader can create a new group, invite or remove members, assign tasks, post announcements, etc. A task and its sub-tasks are logically represented by a tree.
- **Task Visualization**: All Task progresses can visualized with various charts and diagrams. Relations among subtasks and parent tasks will be represented by a tree.
- **Task Progress Report**: User can report current status regarding each sub-task. An overall progress of the main task will be calculated and post to the group.
- **Cloud File Storage and Access (Tentative)**: A designated place will be used to allow users to store private files. A simple file system will be maintained and user may upload and download file.
- **User Location Report (Once User uses Google Location Service; tentative)**: User may choose to share his location among team members of a group. Locations will be visualized geographically.

General Priorities

Please refer to the Product Backlog for information about how specific items that this project should achieve regarding each non-functional requirements.

With respect to the constraint of time, the following non-functional requirements are ranked according to their short term significance. Please note that such an ordering is not valid in the long run.

The primary short term objective is to implement major functionalities of SPM and ensure the robustness of the application. Specific rationales are given under each non-functional requirement, explaining why this requirement is ranked in such a way.

1. Accessibility & Availability

Although it does not require too much effort to ensure such requirements, these are of the most importance because the intuition of SPM relies on a constant attention from the user, which means the user need to constantly check with the application and obtain updates from the group.

2. Performance

It is essential to have robust application to fulfill the purpose of SPM. A buggy application which constantly runs into problems is certainly unacceptable. The server must be well designed and implemented for reliable performance and future extensions in functionalities. The nature of this application lie in a constant communication between clients and the server, and in turns the server and the database. A disruption in the server will disrupt this chain. Unfortunately, with the limitation of time and efforts, this may not be perfect in the short term.

3. Usability

Although it is important to provide an easy to use UI, with the restriction of time and efforts, in the short term it is difficult to design a perfect UI for the ease of use. Certain amount efforts will still be invested.

4. Security

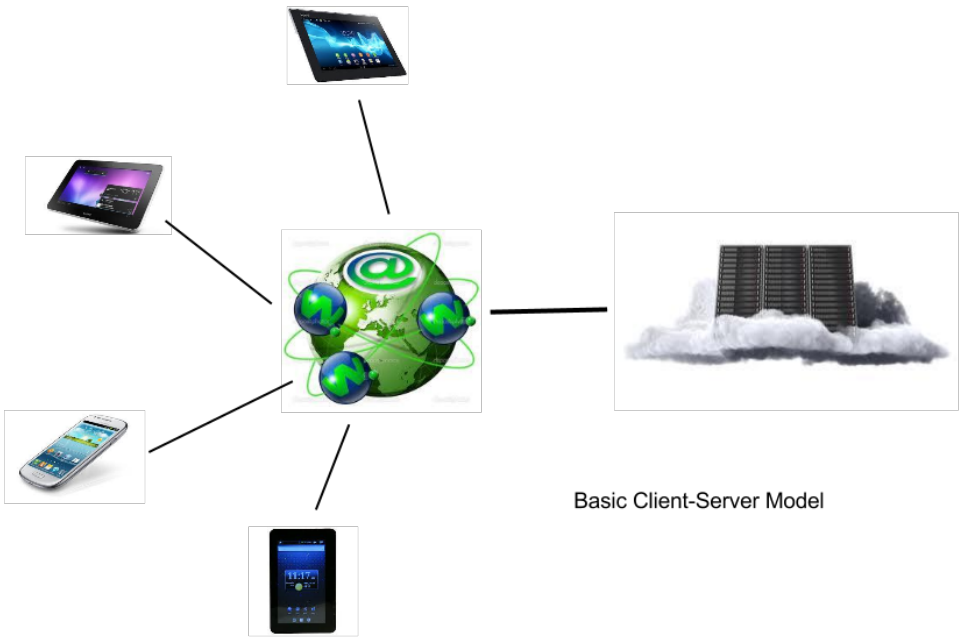
Although it is of absolute importance for such a social application, personnel of this team are generally not specialized in security. With respect to time and efforts, in the short term, the efforts to be invested in fulfilling this requirement are limited. In the long run, this will be taken good care of.

5. Recoverability

Due to the fact that this project is in its implementation and experimental period, the data are usually mock-up for testing purposes. A recovery system is not significant in short term. However, in the long run, this will be taken good care of, especially for marketing purposes.

Design Outline

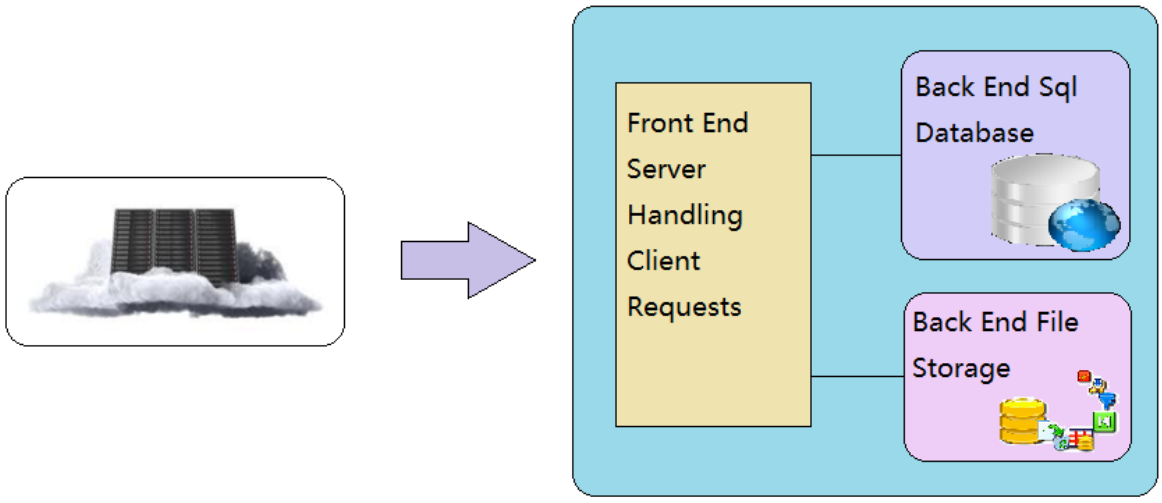
In the System's level, as introduced in the Product Backlog, SPM adopts the classical Client-Server model, as illustrated in the following diagram.



Basic Client-Server Model

To be more specific, SPM requires the engagement of Multiple Clients which serve as interfaces for users, one logical Frontend Server which bridges the communication among clients, and one logical database storing all user data. In the scope of this project, the front end server consists of a single Java program with multiple threads handling Clients' requests. In future, this abstraction can be implemented with a Cluster of Servers sharing the same SQL database to cope with large number of requests per second. A particular protocol will be designed for the purpose of standardizing communications between clients and the server. Clients will state their requests according to the protocol and the server will parse the requests, issue SQL queries and retrieve data for the client, or issue requests to other Clients with this protocol.

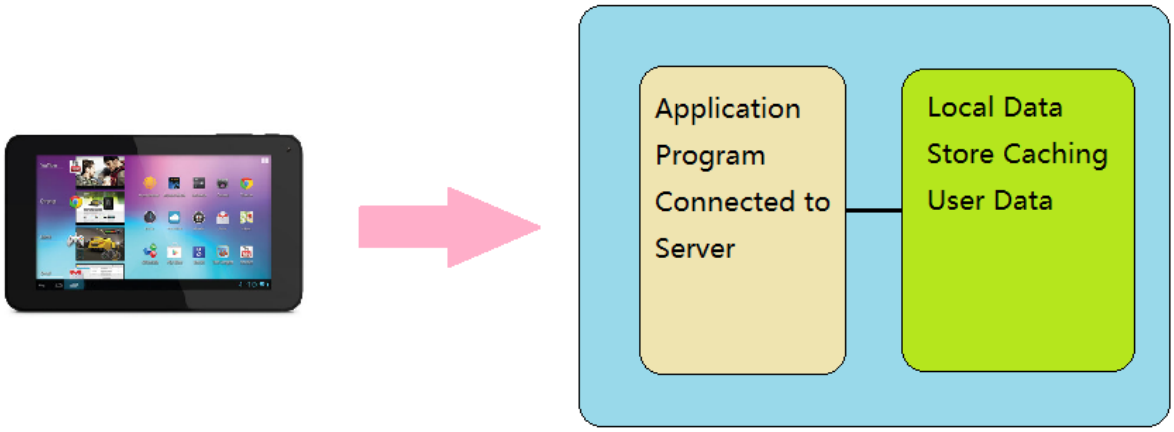
The logical database can be further identified as an SQL Database storing all textual information such as user login info, tasks, group information, contacts, etc., and a File Storage System running on dedicated machines to serve files such as documents, pictures, videos, etc. A detailed illustration is as follows.



This makes sense because non-textual user data cannot be easily and efficiently stored with SQL. A separated cluster of databases handling user-committed files is necessary because it allows future extension of storage

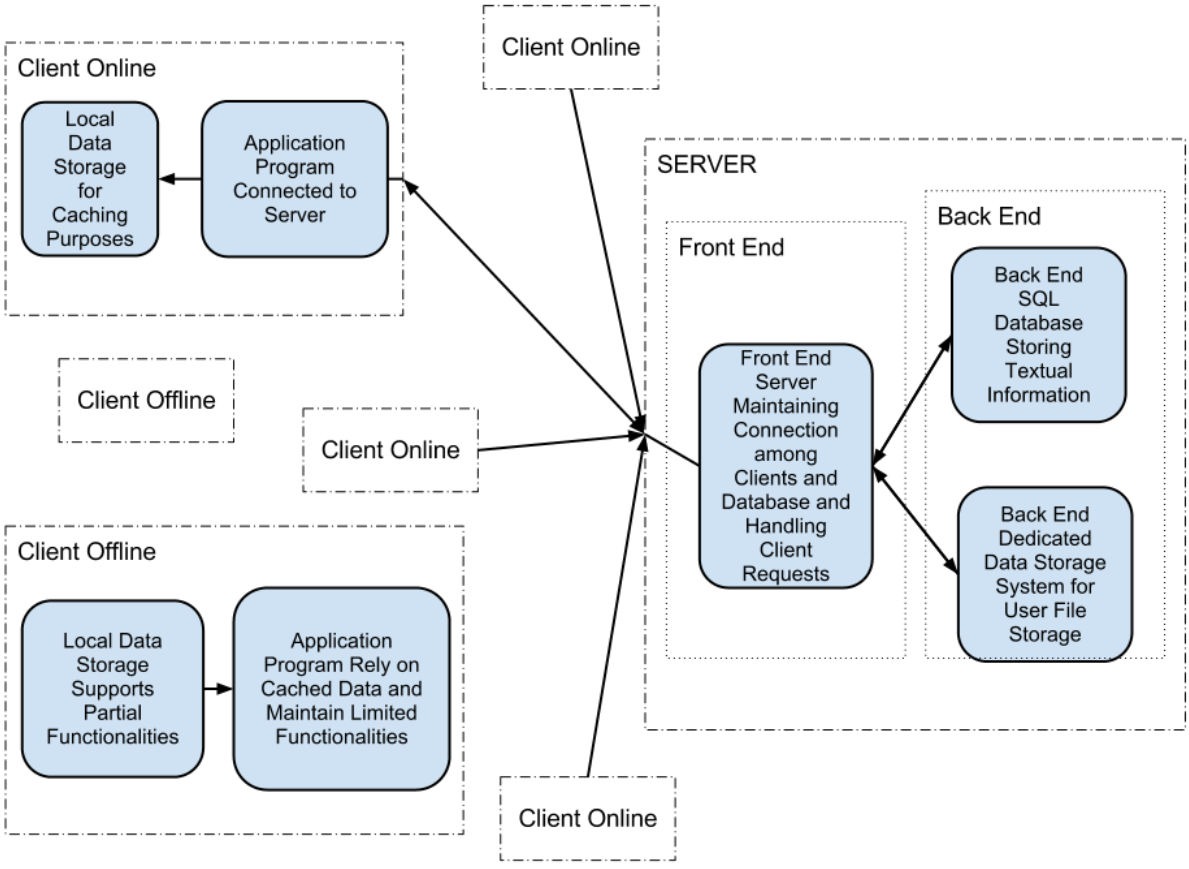
space, as we expect the space required to store user-committed files will increase substantially in the long run.

All Clients maintain a local copy of all relevant information to the User for caching and recovery purposes. A different and ad-hoc data storage schema will be used specifically for local data storage. A detailed illustration is as follows.



A major advantage of this design is that it allows Client to continue to function even if its connection to Server is terminated. This makes sense in the scenario of SPM because once the Client records the due date of a task, it will continue to notify users about the task status. Moreover, Personal Progress Management System is still fully functional even if the Client is offline. Another advantage of maintaining a local copy of User Data is that in the event of disruption of database, the copies of data from the Client side can naturally recommit all necessary data for database to recover smoothly.

The complete and annotated architectural design is illustrated by the following diagram.



Design Issues

1. **Server: Implementation**

   Option 1: Build an ad-hoc all-in-one server which handles data storage with OS' own file system.

   Option 2: Consider the server as an abstraction of 3 components: A front end server which handles communications between clients and server, processes clients' requests, receiving, forwarding information, transferring data for storage, etc., a dedicated SQL database, such that the front end store and retrieve data through SQL, and another dedicated database for user file storage.

   Decision: Although it's cheaper to go with option 1 to get the project functioning, from the prospective of a course project, having the experience with SQL is certainly worthy. More importantly, as an application which is designated to bring convenience to users, we do care about robustness and the potential of further improvements. Option 2 certainly beats Option 1 in terms scalability and robustness.

2. **Client: Thin Client vs. Fat Client**

   Option 1: Thin Client

   Option 2: Fat Client

   Decision: It is beneficial to implement a thin client. It is not just because it is easier to implement. Due to the fact that most Android devices do not have very rich computational resources for applications, it is better to reduce the computational costs of the application. From user's prospective, the most important thing is efficiency and performance. In the scope of SPM, the major functionalities do not require much computation. So, Option 1 is more appropriate.

3. **Database: User identification**

   Option 1: Assign unique UUID for each user for identification

   Option 2: Using user email as ID

   Decision: After inspecting the relation diagram between objects, we have identified that using email address as ID is sufficient, as long as the integrity of primary key is preserved. So, Option 2 is enough.

4. **Database: Task Management**

   Option 1: Label Personal and Group Tasks separately

   Option 2: Do not assign explicit labels to Personal Tasks or Group Tasks

   Decision: It might be easy to not explicitly distinguish Personal and Group Tasks, but doing so will require extra data structures to store the information such that different tasks will be associated to different groups or persons. However, it is much cheaper and efficient to explicitly label Personal and Group Tasks, by adding one more attribute to each Task, and we can still maintain all tasks in one table. So, Option 1 is a better approach.

5. **Data structure: Task Representation**

   Option 1: For each task, maintain a list of tasks as its sub-tasks, recursively. This allows levels of abstractions and maintains a hierarchy of tasks.

   Option 2: Do not maintain sub-task, all tasks are specific.

   Decision: In the paradigm of SPM, it is of great importance to provide users with better experience in managing task progress. From a group's perspective, it is natural and efficient to arrange tasks in a hierarchical order which reflects the structure of a major task. In addition, by representing a major task with sub-tasks in a tree structure, we can aggregate task progress in each level in a bottom-up manner. So, option 1 will be ideal and worthy.

6. **User Interface: Separate Views for Personal and Group Progress**

   Option 1: Display all personal tasks and group tasks in the same view.

   Option 2: Display personal tasks and group tasks in different views

   Decision: As a part of the design, we want SPM to continue functioning in an offline manner, which means that group progresses will no longer be updated, while person tasks remain intact because it is usually the client which commits data for personal tasks to the server and not the other way around. Maintaining separate views for personal and group matters makes this feature possible, while mixing them together brings implementation issues and increases complexities. So, Option 2 is preferred.

7. **Feature: Location Service**

   Option 1: Do not provide location service

   Option 2: Provide location service at user's will and specifications

   Decision: This brings more privacy issues than implementation ones. Surely, no location services make the implementation of SPM easier, but from the group's perspective, it is sometimes convenient and necessary to be aware of the locations of all team members. We can leave the privacy issues to be resolved by the users instead of us, by allowing users to specify its location accessibility to certain users. So option 2 is preferred.

8. **Feature: Online Cloud Storage**

   Option 1: Do not provide online cloud storage service

   Option 2: Provide online storage service at user's will.

   Decision: Although it is costly to assign designated spaces for each user to create an online could storage. From the group's perspective, it is sometimes necessary to share documents, pictures or videos among the team mates. Incorporating cloud storage to SPM can better facilitate team work and reduce the overhead of finding another way of file sharing. So, Option 2 is preferred.

9. **Feature: Instant Message System (IMS)**

   Option 1: Do not provide real time IMS. Instead, allow users to leave a message

   Option 2: Provide a real time IMS

   Decision: Although there already exists a number of IMS systems that user can use to communicate, switching from one application to anther application still incurs a significant overhead. It is beneficial to allow users to directly contact each other when using the application with zero overhead. So Option 2 is preferred. So, Option 2 is preferred.

10. **Feature: Auto-filling User Login Information**

    Option 1: Do not offer to auto-fill user's login and password

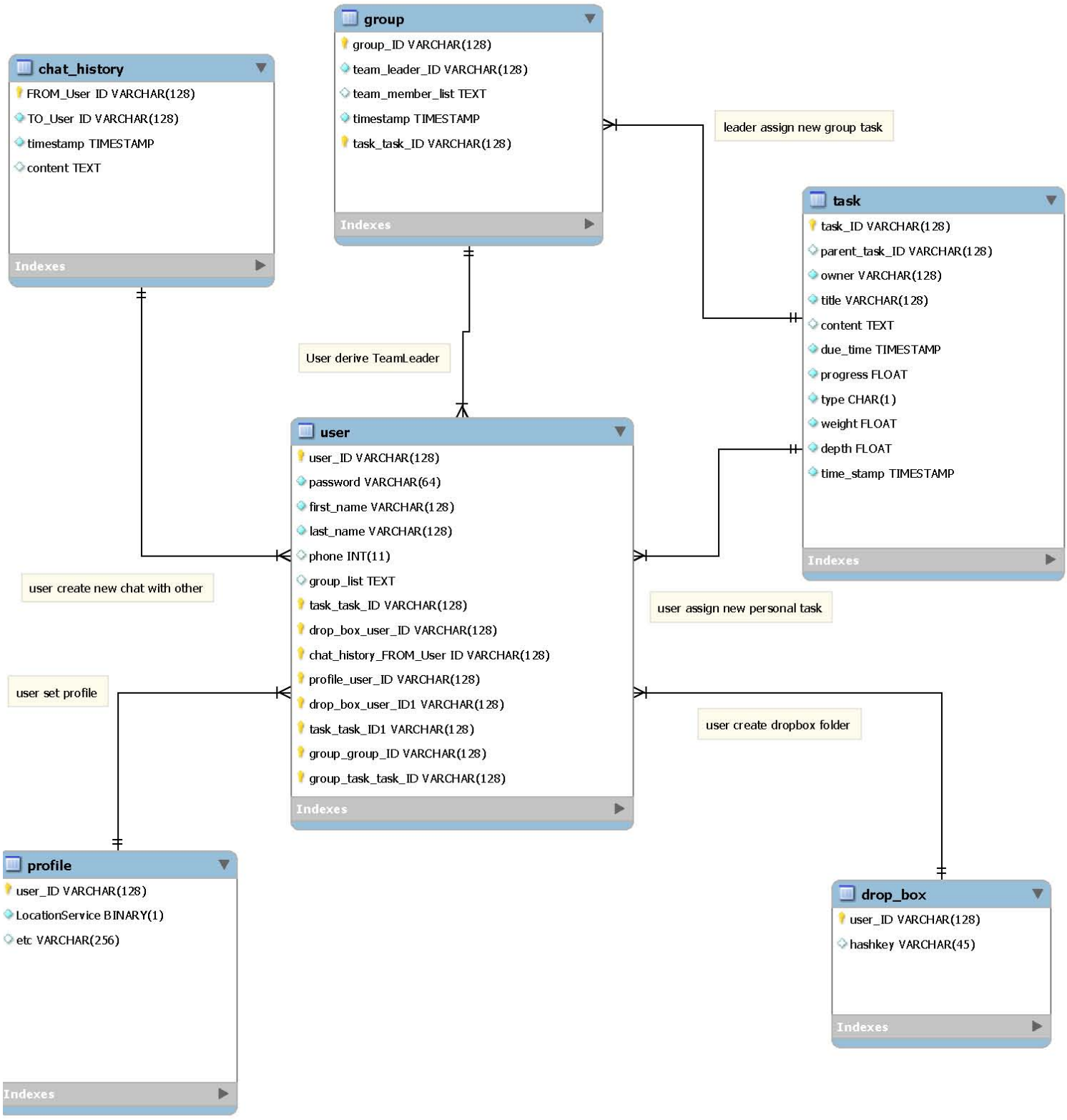
    Option 2: Offer an option for user to auto-fill user's login and password

    Decision: Although from the safety's perspective it is better to have user remember his password upon login, overhead is still overhead. Again, we will provide such feature, but leave it to the user to decide whether to auto-fill or not. We may design smart algorithms which detects abnormally of user's locations and automatically disengage the auto-fill service. So, Option 2 is preferred.

Design Details

CLASS DIAGRAM

The following is the class diagram automatically generated by MySQL.



Description of Classes and Models

- **User**

  This is the tuple encapsulating a user. It stores basic user information, such as User login ID, password, contact information, etc. User ID is the primary key which identifies each user from another. In addition, it is also a foreign key of several other classes, such as chat history, profile and drop box. It can be regarded as the hub of all information that goes around in the system. Please note that we do not directly label if a user is a team leader or not, because we allow a user to be involved in multiple groups simultaneously, such role binding will not be efficient.

- **Group**

  This is the tuple encapsulating a group. It stores basic information about a group. Group ID uniquely identifies a group, which is also a foreign key of a task which is labeled as a group task. It maintains a specific User ID for the team leader, and a list of User ID for regular team members. This structure frees the burden of encoding group information into User Class directly. In addition, a timestamp is recorded at the creation of the group. Please note that all timestamps related to group tasks are generated by the server.

- **Task**

  This is the tuple encapsulating a task. As mentioned, we have determined to store tasks using tree structures. Specifically, we allow defining a complex task, which constitutes several sub-tasks. The tree structure can be implemented gracefully with multiple level of indexing. That is, a Task tuple stores not only the Task ID or itself, which uniquely identifies a task. It also contains a Task ID of its parent task. Such "Inverted Tree Structure" makes the construction of a task tree possible for Clients. Moreover, it also contains the owner ID, which is simply a foreign key referencing a specific user or a group. In addition, it also contains other general information about the task such as content, title, due date, etc. Please note that the timestamp here is used for identifying duplicated task creations.

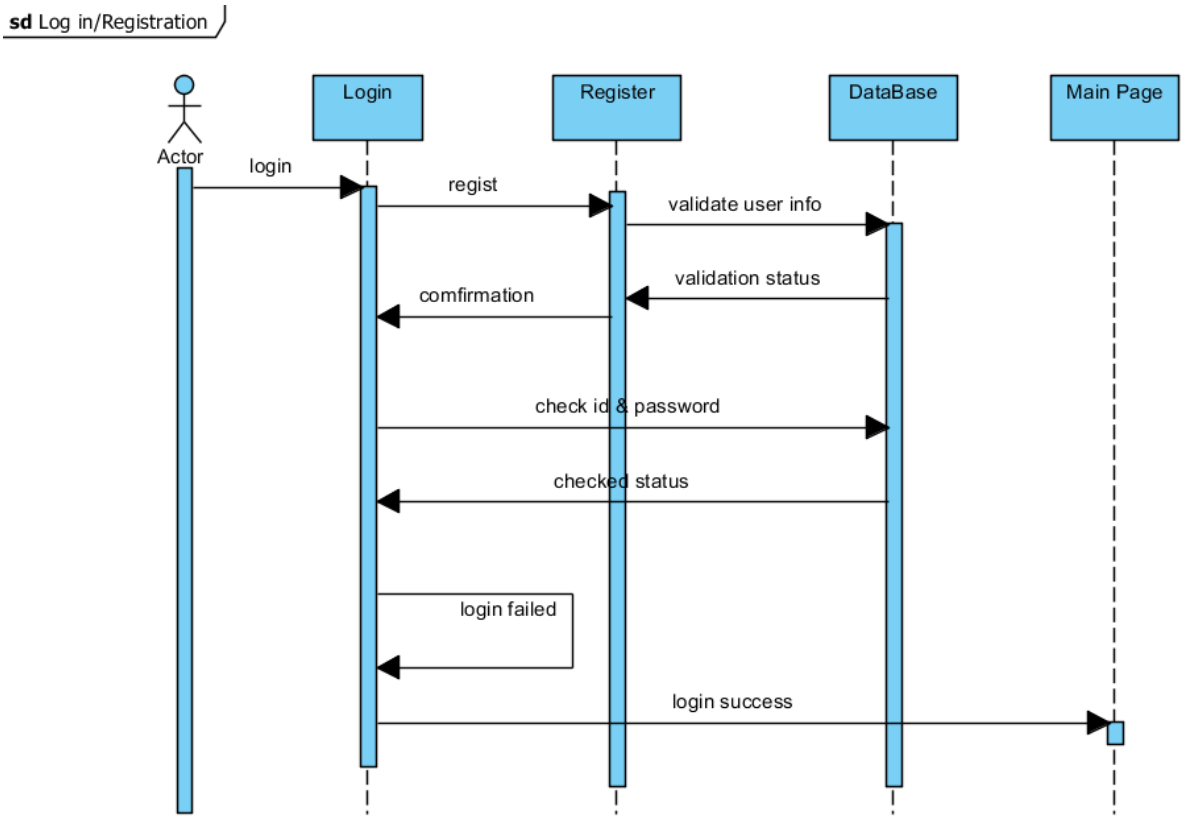
- **Chat History**

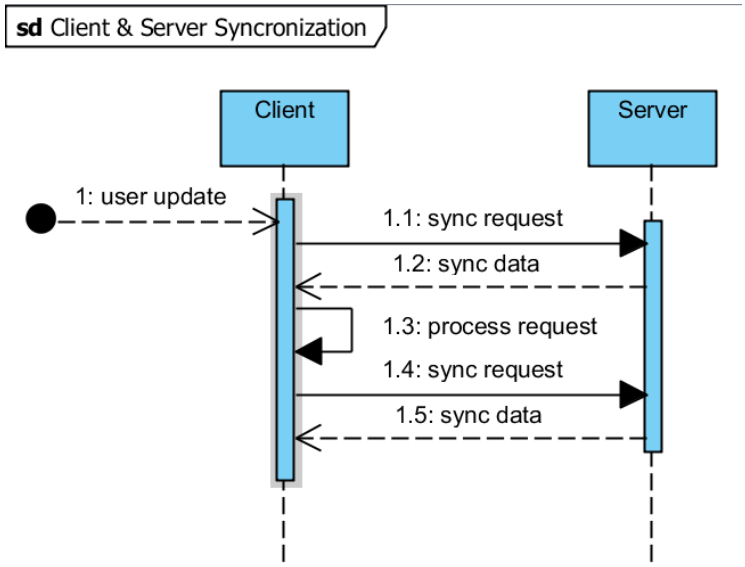
  This is the tuple encapsulating a piece of chatting history between two specific users. It records the content of the message, the timestamp of this message, which user sent this message and which user received this message. The Chat History table is expected to be huge, since we store messages one by one.

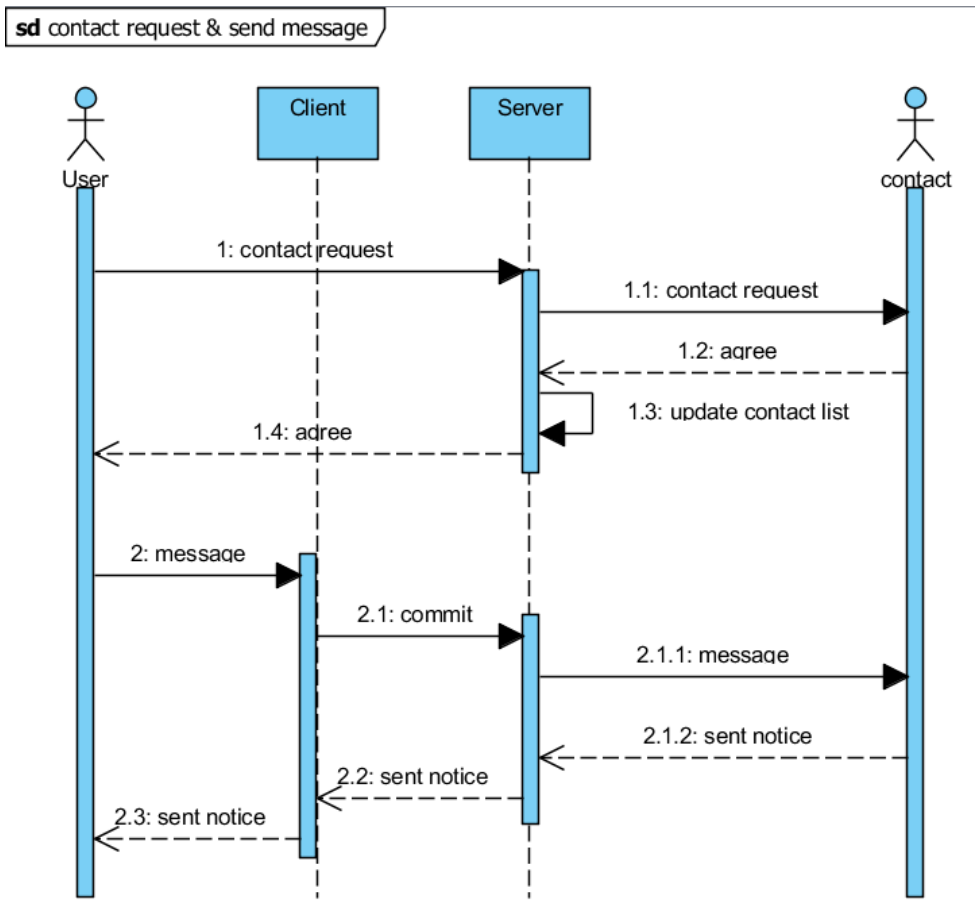
- **Profile**

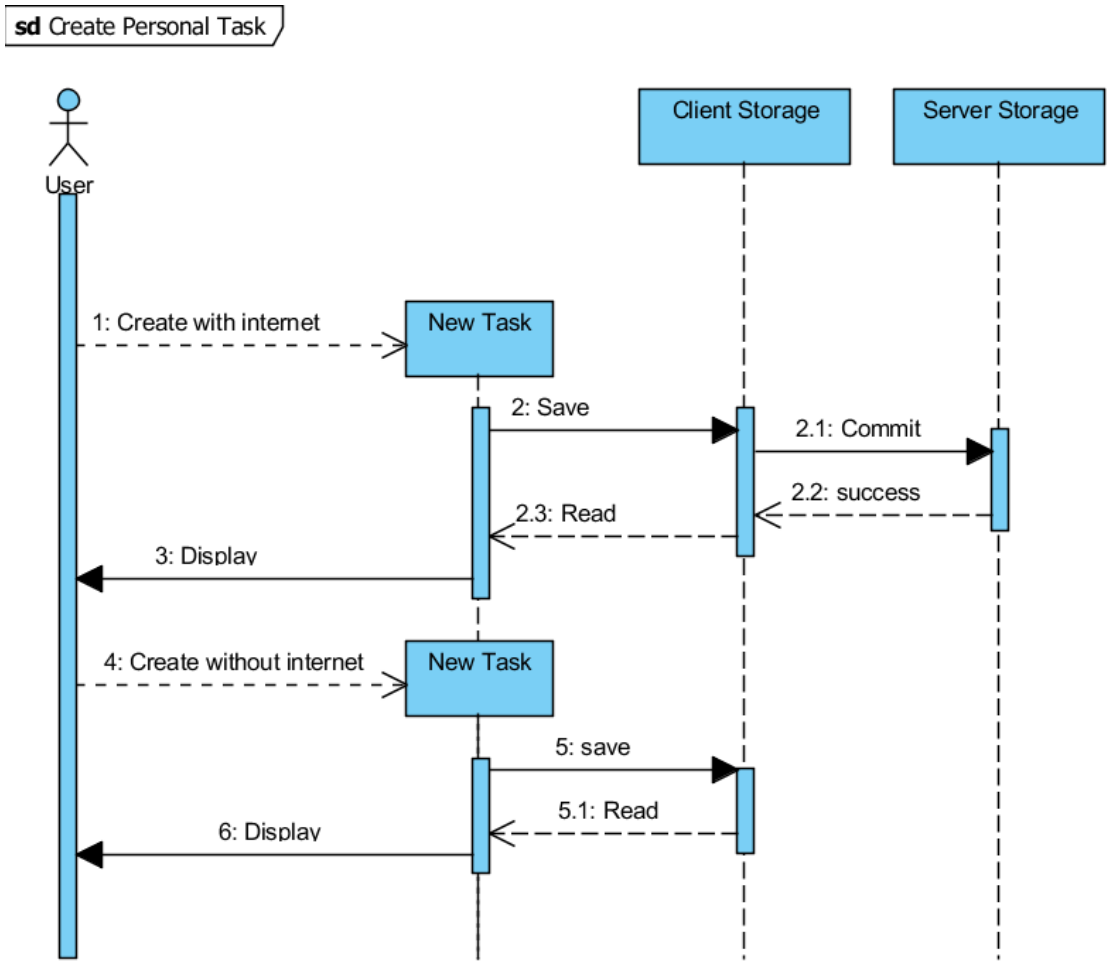
  This is the tuple encapsulating a user's customized profile. Please note that due to the complexity of this project we have not determined specific options for users to customize. One example given is "Location Service", which is a Boolean value specifying whether the user enables or disables this option.

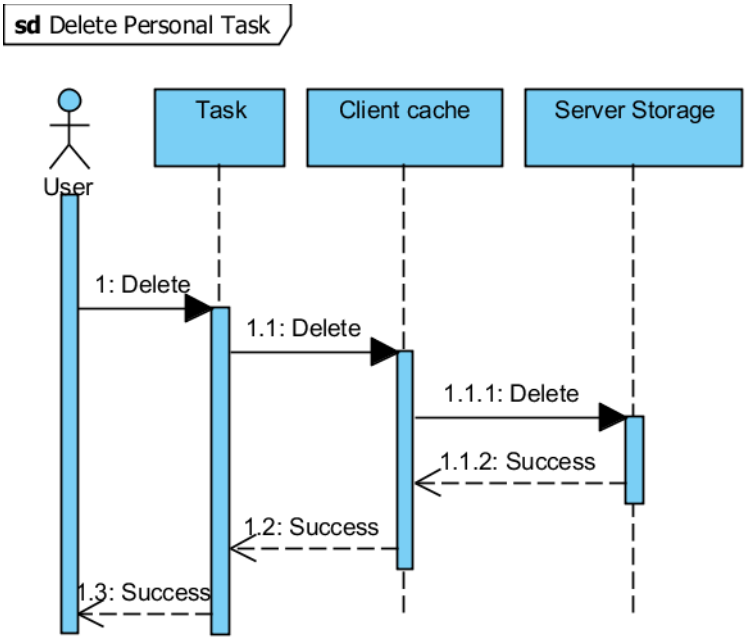
SEQUENCE DIAGRAM





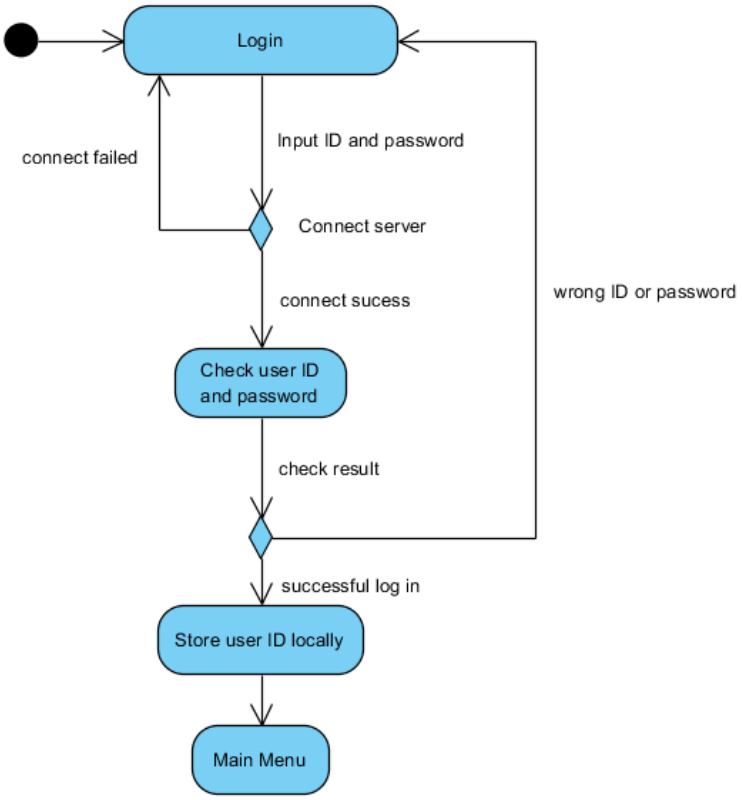

sd contact request & send message
User
Client
Server
contact
1: contact request
1.1: contact request
1.2: agree
1.3: update contact list
1.4: adree
2: message
2.1: commit
2.1.1: message
2.1.2: sent notice
2.2: sent notice
2.3: sent notice


sd Create Personal Task
User
Client Storage
Server Storage
1: Create with internet
New Task
2: Save
2.1: Commit
2.2: success
2.3: Read
3: Display
4: Create without internet
New Task
5: save
5.1: Read
6: Display

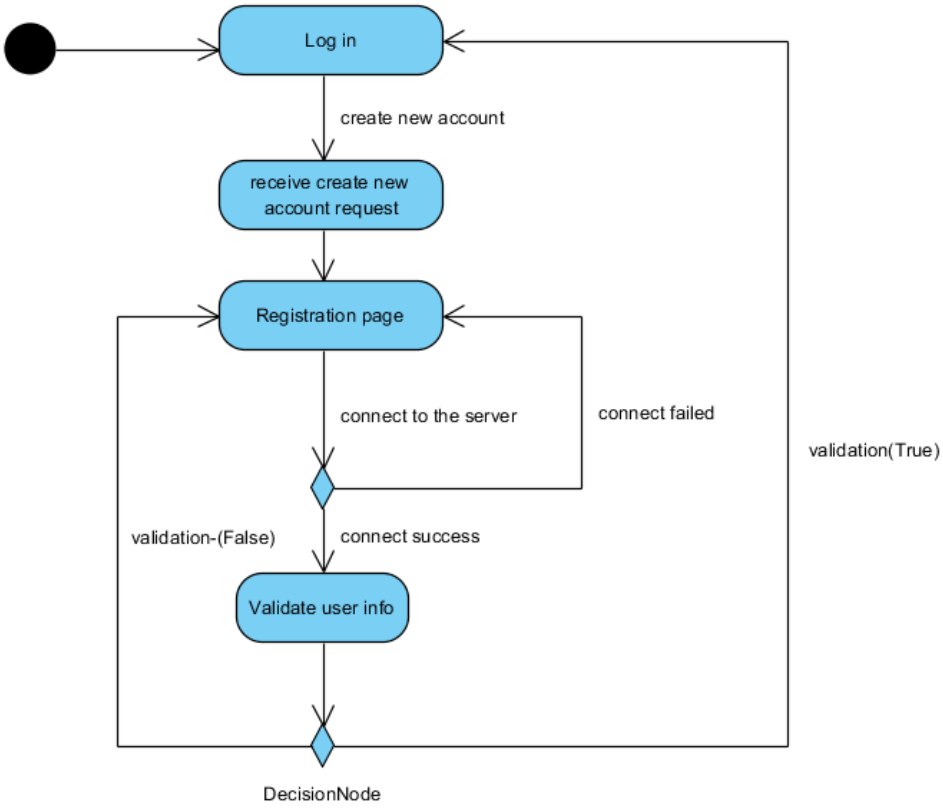

sd Delete Personal Task
User
Task
Client cache
Server Storage
1: Delete
1.1: Delete
1.1.1: Delete
1.1.2: Success
1.2: Success
1.3: Success

ACTIVITY DIAGRAM

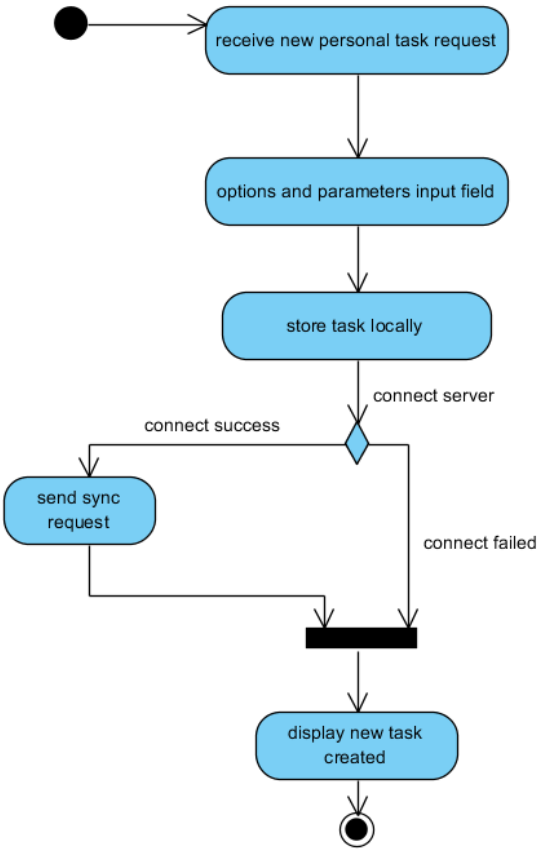
Login



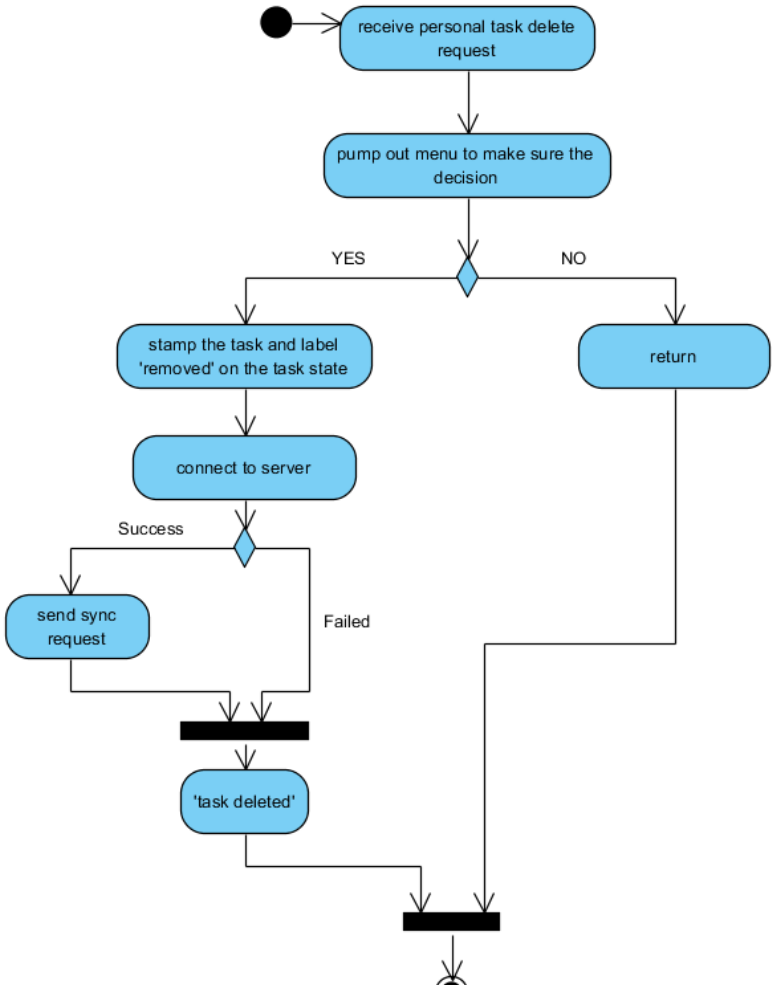
Registration



Personal Task Creation



Personal Task Deletion



Client Server Synchronization