Design Document for Mini-Project 1 (by Jejoon Ryu and Moe Numasawa)

< General Overview of Our System >

This is a platform where users can make posts and perform various actions to posts by managing the enterprise data in the database which works like social media. Each user is either an ordinary user or a privileged user. While the actions that an ordinary user can do are limited, a privileged user is able to perform all of the actions available in the system.

First Screen:

- ➔ [si] Sign in
- ➔ [su] Sign up
- ➔ [q] Quit

Main Menu:

- ➔ [pq] Post question
- ➔ [sp] Search posts
- ➔ [so] Sign out
- ➔ [q] Quit

Sub Menu:

- ➔ [vp] Vote post
- ➔ [wa] Write an answer
- ➔ [gb] Give a badge
- ➔ [t] Add tags
- ➔ [ep] Edit post
- ➔ [ma] Mark as accepted
- ➔ [bm] Back to menu

The interface of sub-menu changes depending on the type of users and the type of posts selected.
Color code: the options are available for
[all users] [users who select an answer] [privileged user] [privileged users who select an answer]

< The Detailed Design of Our System >

First Screen:

page.**signIn**(*cursor*)

- Prompts the user to enter their user ID and password. Checks if they exist in the database. If it exists, the user is signed in. Returns the same user ID.
- Password is case-sensitive. It is not visible at the time of typing.
- User ID is case-insensitive.

page.**signUp**(*connection, cursor*)

- Prompts the user to sign up by asking for the following information: user ID, first name, last name, city, and password.
- User ID should be unique, alphanumeric, and at most 4 characters. If the user enters more than 4 characters, s/he is prompted to either select the first 4 characters of the entry as his/her user id or re-enter it.
- Password is alphanumeric and case-sensitive. The user is prompted to enter the same password twice for the double checking. It is not visible at the time of typing.
- The new user information is recorded in the database with the date created set to the current date.

After a successful sign up, the user will go back to the first screen.
*Users are able to quit from the first screen and the main menu. They can sign out from the main menu to go back to the first screen. Users are also able to go back to the main menu from the sub menu.

After successful sign in, the user can enter a command to perform a system functionality from the main menu.

Main Menu:

action.**postQ**(*connection, cursor, poster*)

- Prompts the user to create a question post with a title and a body text and saves in the database.
- pid is generated by the system with the format of p'x', where x ranges 001 <= x <= 999.
- The post information includes: the generated pid, the post date set to the current date, the title and body, the current user ID as the poster, and the accepted answer (set as NULL by default)

action.**searchPosts**(*cursor, isPrivilege*)

- Prompts the user for one or more search keywords, each separated by a comma.
- The search looks for the matching keywords from the three fields of the table *posts*: title, body, and tags.
- Keywords are case-insensitive.
- If there exists one or more matching posts, the system displays at most 5 posts at a time. The user can either choose to view more, or to perform a post action.
- Returns the selected post and action when the user finishes choosing them.

Sub Menu:

action.**castVote**(*connection, cursor, pid, uid*)

- Prompts the user whether to cast a vote on the selected post. The user is not allowed to vote on the same post more than once.
- The vote information is recorded in the database, with the pid of the selected post, the vno generated by the system, the vote date set to the current date, and the user ID of the signed in user.

action.**postAns**(*connection, cursor, poster, qid*)

- Prompts the user to create an answer post to the selected question post with a title and a body text.
- pid is generated by the system with the format of p'x', where x ranges 001 <= x <= 999.
- The post information is recorded in the database, with the generated pid, the post date set to the current date, the tile and body, the current user ID as the poster, and the pid of the selected question post.

paction.**giveBadge**(*connection, cursor, uid*)

- Displays a table of badges available with their type and badge name, and prompts the privileged user to select one badge from the table the s/he wishes to assign to the poster of the selected post.
- The user cannot give a badge that is not available in the table.
- Each user can only receive one badge per day, which means the privileged user is not able to give a badge to the user who has already received a badge on that day. The privileged user is also not able to give a badge if there is no badge available.
- The badge information is recorded in the database with the user ID of the poster, the badge date set to the current date, and the badge name.

paction.**addTag**(*connection, cursor, pid*)

- Displays the tags currently attached to the selected post, and prompts the user to enter one or more tags to add, each separated by a comma.
- The user is not allowed to add the tags that are already attached to the post.
- The names of the tags are case-insensitive.
- The tag information is recorded in the database with the pid of the selected post and the tag.

paction.**editPost**(*connection, cursor, pid*)

- A privileged user can edit the selected post by providing a new title and/or body text.
- Pressing an enter with nothing typed will keep the content unchanged.

paction.**markAnswer**(*connection, cursor, aid*)

- A privileged user can mark the selected answer post as accepted.
- If a question already has an accepted answer, the user is prompted whether to update or remain unchanged.

< Testing Strategy >

There were two main types of tests to perform:

1. functionalities that updates the database, and
2. functionalities that can be directly checked from the program output.

For the system functionalities that update the database, but do not output anything, we checked whether the database was correctly changed or not by looking at the rows directly queried using sqlite3 in the shell.

For the functionalities that can be visibly checked from the output, we compared the actual output with the desired output.

<u>Testing Scenarios:</u>

Initially, the database contained:

- One privileged account for each group member,
- 10 question posts, 7 answer posts, and 3 accepted answer posts,
- 2 badge names for each badge type (there were 3 types in total),
- 16 votes evenly distributed to the 4 posts.

Since the contents of the title and body for the posts do not matter, they were recorded in the way so that action.searchPosts() can be tested easily. The first letter of the body contains the number of total keywords in the title, body, and tags. Only the repeated keywords are written in those three fields to simplify the tests. For example, if we choose the keyword "database" for the testing, then a post will have a similar format to as following:

Title: "Database"
Body: "6 Database, daTabase, database, dataBase"
Tag: "DatabASE"

Even if we are testing on multiple keywords, the format stays the same.

We then checked each functionality by the following four cases: an ordinary user selects a question, an ordinary user selects an answer, a privileged user selects a question, and a privileged user selects an answer. We came up with some specific scenarios for votes, badges, tags, and accepted answers.

For **votes**, we considered the following two scenarios: 1) the user tries to vote on the post that h/she hasn't voted and 2) the user tries to vote on the post that h/she has already voted.

For **badges**, there are three main scenarios: 1) there is no badge available in the badges table, 2) the user tries to give a badge to the user who has already received a badge on that day, and 3) there are some badges available and the badge receiver hasn't gotten any badge on that day. In scenario 3, we came up with three sub-scenarios: A) the user enters a wrong badge name or a badge name that is not available, B) the user enters a correct badge name, and C) the user enters the correct badge name but its case is different.

For **tags**, there are two main scenarios. First, when there is no tag attached to the post, we considered two sub-scenarios: A) the user enters one tag and B) the user enters more than one tag. Second, when there are some tags already attached to the post, we tested two sub-scenarios: A) the user enters only the tags that have already been added to the post and B) the user enters the tag(s) that have already been added and at least one new tag.

Lastly, for **marking an answer as accepted**, we evaluated the functionality by two scenarios: 1) there is no accepted answer to the question and 2) there exists an accepted answer to the question.

We tested each scenario using the testing strategies and ensured that we got the desired results.

Statistics:

Regardless of the complexity, there were around 1-3 logical errors and 2-5 syntax errors for each functionality. Although the number of bugs and the complexity did not correlate, the number of time spent and the complexity did seem to correlate. It took more time to locate and fix the bugs that arose from the complex functionalities than the bugs from the simple ones.

< Group Work Breakdown Strategy >

We equally distributed the workload by breaking down the system functionalities that we had to work on in this project. We then set deadlines and had meetings to check on each other's progress. After completing all of the system functionalities, we revised the code several times and checked the program together by using the testing strategies discussed above.

Work items breakdown by each member:

Jejoon Ryu (estimated time spent: 15-20 hrs)	Moe Numasawa (estimated time spent: 18 hrs)
- Main menu - Sign up - Search posts - Mark as accepted - Edit post	- Sign in - Post a question - Write an answer - Vote post - Give a badge - Add tags