Design Document

Parekh Dhairya	200050097
Naman Singh Ran	200050083
Shikhar Mundra	200050131
Utkarsh Ranjan	200050147

LinkUs

April 04, 2023

Overview of application

Do you often get frustrated looking for a link shared by someone? Do you like reading and sharing content with your friends but cannot keep track of the exchange? Fret not, for we have something just for you!!

We intend to create an app with WhatsApp like user interface meant to share links. True to the vision, there will be provisions for making groups of users or sharing with individuals. Along with the links, you can add a small description and add relevant tags which describe the contents of the link shared. Just like WhatsApp, you will have provisions for logging in/ signing up, with username, password functionality and email ID integration (add google id integration*). There will be a unique username (by you and for you) and a profile visible only to you. You can also bookmark links which then will be shown in the profile helping you manage your interests (and a brief history of links of your interests*).

You will be surprised how simple it can make things for you. You can see what books you have shared just by searching for the tag ‘books’ and you will be shown filtered data. You will also have the freedom to like and dislike the shared links and can sort the links according to their popularity.

Creating groups with your peers is just a click away, all you need is their username, and you are done, it does not get simpler. You will be the admin of the group you create (I know you like it) and will have the authority to add/remove the members of a group (or make another user admin*). If you are just a member, then you can leave the group at your will.

* Indicates the features which we will integrate once the other features are up and running smoothly

Users

- After signing up, which will involve adding the personal information of the user to the Users table, a user can login/log out by using their password. They can also change their password. They will also be able to sort/filter links for particular tags in a specific group or in their entire chats.
- There will be two types of users in a group, an admin and the members.
    - Any user will have the following features,
        - They can send a link, along with the tags attached and a short description. This will make a simple call to the backend to add the relevant details to the Conversation database
        - They can react to (like/dislike) to a message. This will update the count stored in the React table
        - They can bookmark a tag. This will add entries to the Bookmark table at the backend
        - They can create a new group. This action will create a new entry in the Groups table. They also will be (by default) the admin of the group
        - They can leave a group. This will involve changes in the Participant's table
    - The admin will have the additional privilege of adding/removing any member. This too, involves changes in the Participant's table.

Overview of design

The development will occur in flutter for the front end and Node.JS for the back end. The app will use two databases. One local database is kept on the user's phone, and one database is hosted on the server. Both databases are SQL databases. The data local to a user is stored on the local database. The local database used will be SQLite, provided by flutter. We will use sqflite package/library of flutter for handling the local database. In contrast, the data that concerns other users is stored on the server database. The server database used will be PostgreSQL. We will use the pg package/library of Node.JS for handling the server database. The backend will use the express library for API call management.

Goals

1. **Reduce the number of API calls to the server:** As we use a free hosting service, we have to limit the number of API calls to the server. Also, doing this will ensure a much more responsive application.

2. **Reduce the storage on the server database:** The amount of data that we can store on the server is restricted to 1GB. At the same time, our data could get larger than 1GB. So we choose to put the data on the local databases of the user.

3. **Ensure the privacy of users:** To be thought and implemented later. Maybe we can use WhatsApp like end-to-end encryption.

4. **Preserve user state across devices:** If a user logs out and logs in from another device, he should be able to see at least a certain subset of links that he would see if it were the same device.

5. **The app should work even if the user is offline:** If a user logs in once, the login must persist until he explicitly logs out from the app, and the app should work even when the user is offline.

Features

1. Sign up and log in. Use a unique username for each user. Change password functionality to be added later.
2. Able to create groups and be a part of multiple groups. Personal sharing and sharing with myself are to be added later.
3. Able to send Links (Link Objects) to a group. Forwarding to be added later.
4. Bookmark a Link and view it from the profile page. Search and sort are available here as well.
5. Search and sort through Links of a particular group via tags, time, and sender. Global search through all groups is also allowed.
6. Search through groups by their names.
7. Able to react to Links sent in a group. And view the reactions of other users in the group.

8. Able to leave a group if not an Admin.
9. Delete messages for me available to all members of the group. Delete messages for everyone available to the sender as well as admins of the group.
10. Add and remove members to a group available to admin.
11. Leaving of Admin, Deletion of the group for myself as well as for everyone to be implemented later.

Database Schema

❖ Server Side Database

1. Users
    a. User ID: Primary Key, Non-nullable, Numeric
    b. User Name: String, Non-nullable
    c. Password: String, Non-nullable
    d. Email-ID: String, Non-nullable
2. Groups
    a. Group ID: Primary Key, Non-nullable, Numeric
    b. Group name: String, Non-nullable
    c. Group information: String, nullable
3. Participants
    a. Group ID: Primary Key, Foreign Key Groups on Delete Cascade, Numeric
    b. User ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
    c. Role: String Admin/Member
4. Links
    a. LinkID: Primary Key, Non-nullable, Numeric
    b. GroupID: Foreign Key Groups on Delete Cascade, Numeric
    c. SenderID: Foreign Key Users on Delete Cascade, Numeric
    d. Title: String default value “New Book”
    e. Link: String Non-nullable
    f. Time: Timestamp
    g. Description: String
5. Reacts

   a. LinkID: Primary Key, Foreign Key Links, Non-nullable, Numeric
   b. SenderID: Foreign Key Users on Delete Cascade, Numeric
   c. React: String Like/Dislike/Null default null
6. Message Actions
   a. Receiver ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   b. Sender ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   c. Link ID: Primary Key, Foreign Key Links on Delete Cascade, Non-nullable, Numeric
   d. Time: Timestamp
   e. Action: Receive | React
7. Delete Actions
   a. Receiver ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   b. Sender ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   c. Link ID: Primary Key, Non-nullable, Numeric
   d. Time: Timestamp
8. Group Actions
   a. Receiver ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   b. Group ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   c. Affected ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
   d. Role: String Admin | Member
   e. Time: Primary Key, Timestamp
   f. Action: Add to group | Get added to group | Remove | Change
9. Tags
   a. LinkID: Primary Key, Foreign Key Links, Non-nullable, Numeric
   b. Tag: Primary Key, Non-nullable, String

❖ Client Side Database

1. Users
   a. User ID: Primary Key, Non-nullable, Numeric
   b. User Name: String, Non-nullable
2. Groups
   a. Group ID: Primary Key, Non-nullable, Numeric
   b. Group name: String, Non-nullable

    c. Group information: String, nullable
3. Participants
    a. Group ID: Primary Key, Foreign Key Groups on Delete Cascade, Numeric
    b. User ID: Primary Key, Foreign Key Users on Delete Cascade, Numeric
    c. Role: String Admin/Member
4. Links
    a. LinkID: Primary Key, Non-nullable, Numeric
    b. GroupID: Foreign Key Groups on Delete Cascade, Numeric
    c. SenderID: Foreign Key Users on Delete Set Default, Numeric, default null
    d. Title: String default value “New Book”
    e. Link: String Non-nullable
    f. Time: Timestamp
    g. Description: String
5. Reacts
    a. LinkID: Primary Key, Foreign Key Links, Non-nullable, Numeric
    b. SenderID: Foreign Key Users on Delete Cascade, Numeric
    c. React: String Like/Dislike/Null default null
6. Tags
    a. LinkID: Primary Key, Foreign Key Links, Non-nullable, Numeric
    b. Tag: Primary Key, Non-nullable, String
7. Bookmarks
    a. LinkID: Primary Key, Foreign Key Links Non-nullable, Numeric

UI Implementation

1. On Startup

    Read from local storage whether there are any stored credentials.

    If credentials are present:

    - Redirect to the App
    - Make a fetch data request to the server
    - Update the local storage timestamp on successful fetch

- Update the local database correctly //Defined ahead in detail

If credentials are not present:

- Redirect to the Login/SignUp page
- Set local timestamp to 0

2. Login/Signup

The first screen to a new application instance will be an option to either login or signup. If the user clicks on login:

- The user is prompted to enter his user id and password.
- On clicking submit, we make an API call to the server to check the credentials.
- If the credentials are correct
    - Store the credentials in the local storage of the app.
    - Redirect to the App
- If the credentials are incorrect
    - The user is prompted again.

If the user clicks on register:

- The user is prompted to enter credentials
- On clicking submit, we make an API call to the server to check that user with such credentials does not exists already.
- If the credentials are correct
    - Store the credentials in the local storage of the app.
    - Redirect to the App
- If the credentials are incorrect
    - The user is prompted again.

3. App

Has Navbar already navigated to Home by default and an Option to force a sync. On clicking it:

- The fetch data request is made to the server
- Update the local storage timestamp on successful fetch

- Update the local database correctly //Defined ahead in detail
- Redirect to Home again

4. Home Page

Read from the local database all the groups and display them.
Give a search bar

- Search from the local database and return the results

On clicking on a group,

- Redirect to the Group Page

Give the option to create a group. On clicking it:

- Pop Up to create a group by entering user names and roles.
- On submitting, make a POST API call to the server to broadcast the action
- On getting an ACK from the server, add it to the local database (**Groups Table, Users Table, and Participants Table.)**

5. Group Page

Read from the local database all the Links and display them.
Give a search bar and sort option

- Search from the local database and return the results

On clicking on a link,

- Redirect to the Link Page

Option to send a link only if I am a Participant. On clicking it:

- Pop Up to create a message along with a description, title, tags, and link.
- On submitting, make a POST API call to the server to broadcast the message
- On getting an ACK from the server, add it to the local database (**Links Table, Tags Table.)**

Option to kick members if you're an admin. On clicking it:

- Make a POST API call to the server to broadcast the action

- On getting an ACK from the server, delete it from the local database (**Participants Table).**

Option to Change the role of members if you're an admin. On clicking it:

- Make a POST API call to the server to broadcast the action
- On getting an ACK from the server, update it from the local database (**Participants Table).**

Option to add members if you're an admin. On clicking it:

- Make a POST API call to the server to broadcast the action
- On getting an ACK from the server, add it to the local database (**Participants Table, Users Table).**

Option to leave the group. On clicking it:

- Make a POST API call to the server to broadcast the action
- On getting an ACK from the server, delete it from the local database (**Participants Table).**

Additional things to display

Group members list, Change group name, group description, etc.

6. Profile Page

Read from the local database all the Bookmarked Links and display them.
Give a search bar and sort option

- Search from the local database and return the results

On clicking on a link,

- Redirect to the Link Page

Additional things to display:

Give the option to Bookmark/UnBookmark it. On clicking it: Update the client-side **Links** table accordingly.

Option to send a reaction. On clicking it:

- On submitting, make a POST API call to the server to broadcast the message
- On getting an ACK from the server, add it to the local database (**Links Table, Reacts Table.)**

Give the option to “delete for everyone” if I am the sender or admin.

- On submitting, make a POST API call to the server to broadcast the message
- On getting an ACK from the server, delete it from the local database (**Links Table)**

Give the option to “delete for me”. On clicking it: delete it from the local database (**Links Table)**

Total likes received, Option to change password or username.

Total likes received, Option to change password or username.

1. Link Page

   Read from the local database all the Reacts descriptions etc., and display them.

APIs

String Size constraints:

```
user_id, link_id, group_id = 36
user_name, group_name <= 15
passcode <= 15
email_id <= 40
group_info <= 150
roles = adm/mem
title <= 50
link <= 256
info <= 100
tags <= 15
react = l/d
```

Local Storage:

```
last updated time_stamp
user_id
username
password
user Email
```

Client Side

1. **Update new message actions**

```
Type : void
   Name : updateMessages
   Params : <list<{
                sender_id: <uuid type char 16>,
                group_id: <uuid type char 16>,
                link: {
                        link_id : <uuid type char 16>,
                        title: <string>,
                        link: <string>,
                        info: <string>,
                        tags: <list<strings>>,
                        time_stamp: <time_stamp>
                }
        }>>
```

2. **Update delete message actions**

```
Type : void
   Name : deleteMessages
   Params : <list<{
                link_id: <uuid type char 16>
        }>>
```

3. **Update react message actions**

```
Type : void
   Name : updateReactions
   Params : <list<{
                  user_id: <uuid type char 16>,
                  link_id: <uuid type char 16>,
                  react: <string>
          }>>
```

4. **Update get Added**

```
Type : void
   Name : getAdded
   Params : {
         <list<{
                  group_id: <uuid type char 16>,
                  group_name: <string>,
                  group_info: <string>,
                  members: <list<{
                           user_id: <uuid type char 16>,
                           user_name: <string>,
                           roles: <string>
                  }>>
         }>>
   }
```

5. **Update change role actions**

```
Type : void
   Name : updateRoles
   Params : <list<{
          group_id: <uuid type char 16>,
          user_id: <uuid type char 16>,
          role: <Enum in flutter>
   }>>
```

6. **Update kick actions**

```
Type : void
   Name : removeMembers
   Params : <list<{
        user_id : <uuid type char 16>,
        group_id : <uuid type char 16>
   }>>
```

7. **Update add actions**

```
Type : void
   Name : addUsers
   Params : {
        <list<{
                user_id: <uuid type char 16>,
                group_id: <uuid type char 16>,
                user_name: <string>,
                role: <string>
        }>>
   }
```

8. **Read groups**

```
Type : list<json>
   Name : readGroups
   Params : {
        user_id : <uuid type char 16>,
        group_id : <uuid type char 16>
   }
   Returns : {
        groups : <list<{
                        group_id : <int>,
                        group_name: <string>,
                        group_info: <string>
                }>>
   }
```

9. Search groups

```
Type : list<json>
   Name : searchGroup
   Params : {
        user_id : <uuid type char 16>,
        group_id : <uuid type char 16>
   }
   Returns : {
        groups : <list<{
                        group_id : <int>,
                        group_name: <string>,
                        group_info: <string>
                }>>
   }
```

10. Search links global

```
Type : list<json>
   Name : searchAll
   Params : {
   }
   Returns : {
        links : <list<{
                        link_id : <uuid type char 16>,
                        group_id : <uuid type char 16>,
                        title: <string>,
                        link: <string>,
                        info: <string>
                }>>
   }
```

11. Search links for group

```
Type : list<json>
   Name : searchAll
   Params : {
   }
   Returns : {
        links : <list<{
                        link_id : <uuid type char 16>,
                        group_id : <uuid type char 16>,
                        title: <string>,
                        link: <string>,
                        info: <string>
                }>>
   }
```

12. Sort the links based on a parameter and a filter

```
Type : list<json>
   Name : sortLinks
   Params : {
        sort_by : <string>,
        tags: <list<strings>>
   }
   Returns : {
        links : <list<{
                        link_id : <uuid type char 16>,
                        title: <string>,
                        link: <string>,
                        info: <string>
                }>>
   }
```

13. Get all groups

```
Type : list<json>
   Name : fetchGroups
   Returns :
        <list<{
                group_id : <uuid type char 16>,
                group_name: <string>
        }>>
```

14. Get links for a particular group

```
Type : list<json>
   Name : fetchLinks
   Params : {
        group_id : <uuid type char 16>
   }
   Returns :
        <list<{
                link_id : <uuid type char 16>,
                title: <string>,
                sender_name: <string>,
                time_stamp: <time_stamp_dart>
        }>>
```

15. Get bookmarks

```
Type : list<json>
   Name : fetchBookmarks
   Returns :
        <list<{
                link_id : <uuid type char 16>,
                title: <string>,
                sender_name: <string>,
                time_stamp: <time_stamp_dart>
        }>>
```

16. Update bookmarks

```
Type : void
   Name : updateBookmark
   Params : {
        link_id : <uuid type char 16>
        action : <string>
   }
```

17. Get group info along with members and their roles

```
Type : list<json>
   Name : getGroupInfo
   Params : {
        group_id : <uuid type char 16>
   }
   Returns : {
        group_name : <string>,
        group_info : <string>,
        members : <list<{
                user_id: <uuid type char 16>,
                user_name: <string>,
                role: <Enum in flutter>
        }>>
   }
```

18. Get group specific user information

```
Type : list<json>
   Name : getGroupSpecificUserInfo
   Params : {
        group_id : <uuid type char 16>,
        user_id : <uuid type char 16>
   }
   Returns : {
        user_name : <string>,
        role : <Enum in flutter>
   }
```

19. Get link info

```
Type : list<json>
   Name : getLinkInfo
   Params : {
        link_id : <uuid type char 16>
   }
   Returns : {
        title : <string>,
        link : <string>,
        info : <string>,
        sender_name : <string>,
        time_stamp : <time_stamp_dart>,
        likes : <int>,
        dislikes : <int>,
        tags: <list<string>>
   }
```

Server Side

1. Login

```
Type: post
   URL: /login
   Body: {
        user_name : <string>,
        password : <string>
   }
   Return: {
        success: <bool>,
        user_id: <uuid type char 16>,
        email: <string>,
       message: <string>
   }
```

2. Sign Up

```
Type: post
   URL: /signup
   Body: {
        user_name : <string>,
        password : <string>,
        email : <string>
   }
   Return: {
        success: <bool>,
        user_id: <uuid type char 16>,
        email: <string>,
       message: <string>
   }
```

3. New Group

```
Type: post
   URL: /create_group
   Body: {
        user_id : <uuid type char 16>,
        group_name : <string>,
        group_info : <string>,
        members : <list<{
                    participant_name: <string>,
                    role: <string adm|mem>
                }>>
   }
   Return: {
        success: <bool>,
        group_id : <uuid type char 16>,
        message: <string>,
        time_stamp: <time_stamp>,
        members : <list<{
                    user_id: <uuid type char 16>
                    user_name: <string>,
                    roles: <string adm|mem>,
                }>>
   }
```

4. Fetch Updates

```
Type: get
   URL: /get_updates
   Params: {
         user_id: <uuid type char 16>,
         time_stamp: <time_stamp>
   }
   Return: {
         time_stamp: <time_stamp>,
         new_messages: <list<{
                  sender_id: <uuid type char 16>,
                  group_id: <uuid type char 16>,
                  link_id : <uuid type char 16>,
                  title: <string>,
                  link: <string>,
                  info: <string>,
                  tags: <list<strings>>,
                  time_stamp: <time_stamp>
         }>>,
```

```
         delete_messages: <list<{
                  link_id: <uuid type char 16>
         }>>,
         react: <list<{
                  sender_id: <uuid type char 16>,
                  link_id: <uuid type char 16>,
                  react: <string>
         }>>,
         change_role: <list<{
                  affected_id: <uuid type char 16>,
                  group_id: <uuid type char 16>,
                  affected_role: <string adm|mem>
         }>>,
         remove_member: <list<{
                  affected_id: <uuid type char 16>,
                  group_id: <uuid type char 16>
         }>>,
         add_user: <list<{
                  user_id: <uuid type char 16>,
                  group_id: <uuid type char 16>,
                  user_name: <string>,
                  role: <string>
         }>>,
         get_added: <list<{
                  group_id: <uuid type char 16>,
                  group_name: <string>,
                  group_info: <string>,
                  role: <string>,
                  members: <list<{
                           user_id: <uuid type char 16>,
                           user_name: <string>,
                           roles: <string>
                  }>>
         }>>
   }
```

5. **Broadcast message**

```
Type: post
   URL: /send_message
   Body: {
        sender_id : <uuid type char 16>,
        group_id : <uuid type char 16>,
        link: {
                title: <string>,
                link: <string>,
                info: <string>,
                tags: <list<strings>>
        }
   }

   Return: {
        link_id: <uuid type char 16>,
        time_stamp: <time_stamp>
   }
```

6. **Broadcast react**

```
Type: post
   URL: /react
   Body: {
        sender_id : <uuid type char 16>,
        link_id : <uuid type char 16>,
        group_id : <uuid type char 16>,
        react : <string>
   }
   Return: {
        time_stamp : <time_stamp>
   }
```

7. **Broadcast delete**

```
Type: post
   URL: /delete_message
   Body: {
        user_id : <uuid type char 16>,
        link_id : <uuid type char 16>,
        group_id : <uuid type char 16>
   }
   Return: {
        success : <bool>,
        time_stamp : <time_stamp>
   }
```

8. **Broadcast add**

```
Type: post
   URL: /add_user
   Body: {
        user_id : <uuid type char 16>,
        group_id : <uuid type char 16>,
        new_member_name : <string>,
        new_member_role : <string>
   }
   Return: {
        success : <bool>,
        message : <string>,
        new_member_id : <uuid type char 16>,
        time_stamp : <time_stamp>
   }
```

9. **Broadcast change role**

```
Type: post
   URL: /change_role
   Body : {
        user_id : <uuid type char 16>,
        group_id : <uuid type char 16>,
        changer_id : <uuid type char 16>,
        role : <string adm|mem>
   }
   Return: {
        success : <bool>,
        message : <string>,
        time_stamp : <time_stamp>
   }
```

10. **Broadcast kick**

```
Type: post
   URL: /remove_member
   Body: {
        user_id : <uuid type char 16>,
        kicker_id : <uuid type char 16>,
        group_id : <uuid type char 16>
   }
   Return: {
        success : <bool>,
        message : <string>,
        time_stamp : <time_stamp>
   }
```

Other aspects

We will use express sessions (as used in lab 4) for session management of users. We intend to create the app on Flutter so that it will work on the android as well as ios mobile devices.