Design Document

Parekh Dhairya
Naman Singh Rana
Shikhar Mundra
Utkarsh Ranjan

LinkUs

April 04, 2023

Overview

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.

4. **Preserve user state across devices:** If a user logs out and logs in from another device, he should be able to see whatever he would see if it were the same device. To be thought and implemented later.

5. **The app should work even if the user is offline:** If a user logs in once, then 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, 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, Non-nullable, Numeric
   d. Time: Timestamp
   e. Action: Receive | React | Delete
7. 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

❖ 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 Cascade, Numeric
    d. Title: String default value “New Book”
    e. Link: String Non-nullable
    f. Time: Timestamp
    g. Description: String
    h. Bookmarked: Bool
5. Reacts
    a. LinkID: Primary Key, 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, Non-nullable, Numeric
    b. Tag: Primary Key, Non-nullable, String

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 an 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:

Total likes received, Option to change password or username.

7. Link Page

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

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)**

APIs

Client Side

1. **Update new message actions**

2. Update delete message actions
3. Update react message actions
4. Update change role actions
5. Update kick actions
6. Update add actions
7. Read groups
8. Search groups
9. Read links for a particular group
10. Search links global
11. Search links for group
12. Sort the links based on a parameter and a filter
13. Read group members with their roles
14. Read bookmarks
15. Update bookmarks

Server Side

1. Login
2. Sign Up
3. Fetch Updates
4. Broadcast message
5. Broadcast react
6. Broadcast delete
7. Broadcast add
8. Broadcast change role
9. Broadcast kick