# Android Application: Social Network App using Firebase

By Ng Yiu Wai, May 2019

**Modified from tutorial "Firebase Social Network App" offered by Coding Cafe.**
***https://www.youtube.com/playlist?list=PLxefhmF0pcPnTQ2oyMffo6QbWtztXu1W_***

This is a student project for building a social network app using Firebase. Our team study the source codes from Coding Cafe, then extract the useful parts to create a simplified app. Algorithm are visualized in this report using flowcharts.

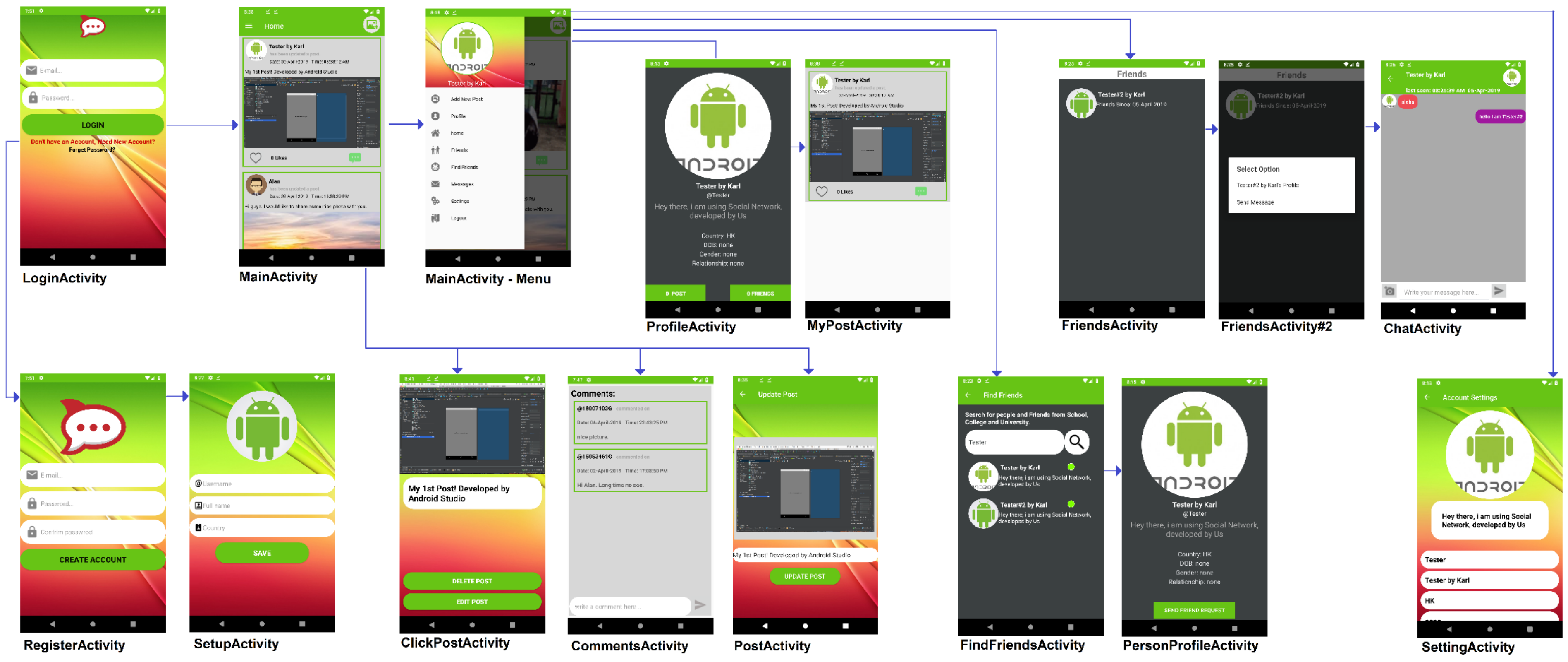
## Content

**Application Logic and Disconnection Management – Client Side**

Our application is an Instagram-like App which supports below functions

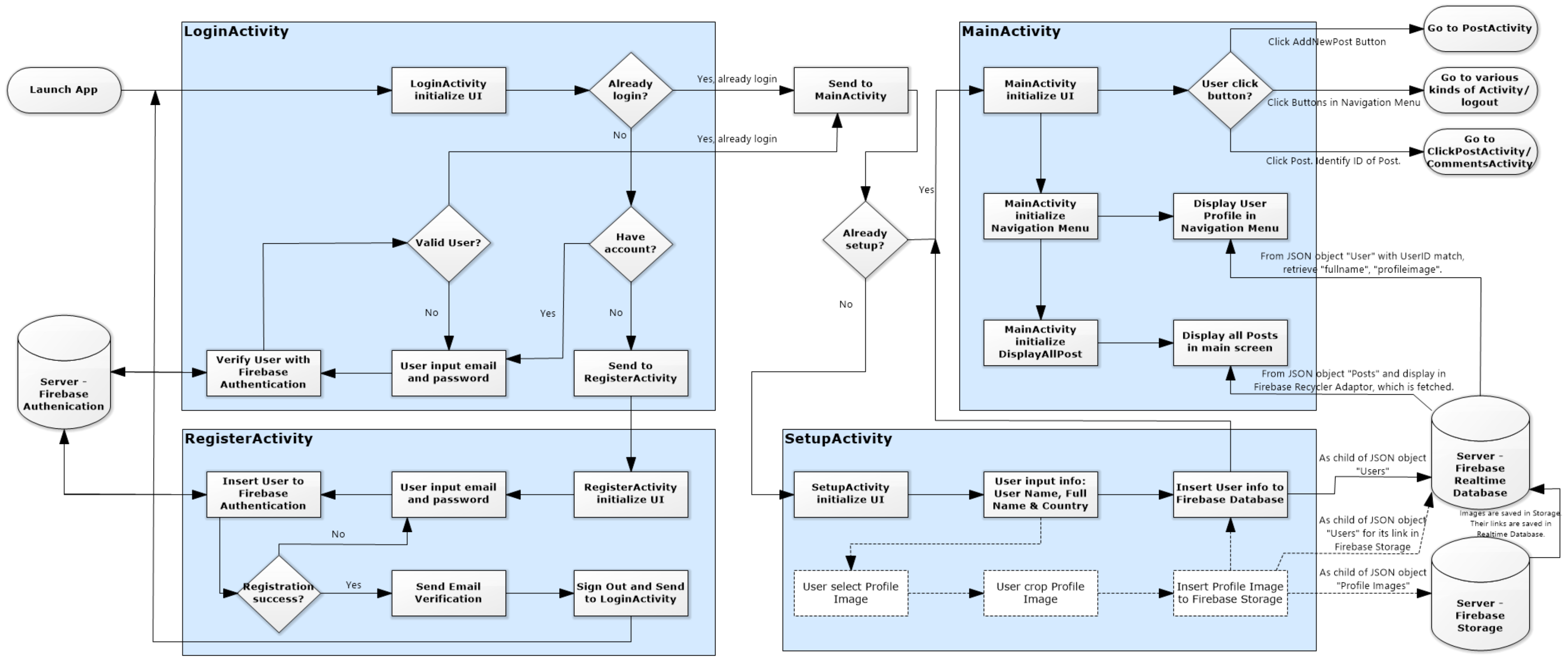
- User identification by login, then retrieving posts from server
- Customize user profile
- View, comment, like and create posts
- Sending messages to friends

*Overview of client-side application*



Application Logic – Client Side – User Identification and Retrieving Posts

*Flowchart*


LoginActivity
Launch App
LoginActivity initialize UI
Already login?
Yes, already login
Send to MainActivity
No
Yes, already login
Valid User?
Have account?
No
Yes
No
Verify User with Firebase Authentication
User input email and password
Send to RegisterActivity
Server - Firebase Authenication
RegisterActivity
Insert User to Firebase Authentication
User input email and password
RegisterActivity initialize UI
No
Registration success?
Yes
Send Email Verification
Sign Out and Send to LoginActivity
Already setup?
Yes
No
MainActivity
MainActivity initialize UI
User click button?
Click AddNewPost Button
Go to PostActivity
Click Buttons in Navigation Menu
Go to various kinds of Activity/ logout
Click Post. Identify ID of Post.
Go to ClickPostActivity/ CommentsActivity
MainActivity initialize Navigation Menu
Display User Profile in Navigation Menu
From JSON object "User" with UserID match, retrieve "fullname", "profileimage".
MainActivity initialize DisplayAllPost
Display all Posts in main screen
From JSON object "Posts" and display in Firebase Recycler Adaptor, which is fetched.
SetupActivity
SetupActivity initialize UI
User input info: User Name, Full Name & Country
Insert User info to Firebase Database
As child of JSON object "Users"
Server - Firebase Realtime Database
User select Profile Image
User crop Profile Image
Insert Profile Image to Firebase Storage
As child of JSON object "Users" for its link in Firebase Storage
As child of JSON object "Profile Images"
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Storage

*LoginActivity, RegisterActivity and SetupActivity*

User should provide his/ her identification information to application. The application will submit the information to server for login verification/ creating account. Once user is login, and unique userID would be downloaded and cached in the mobile device. No login is required next time.

*MainActivity*

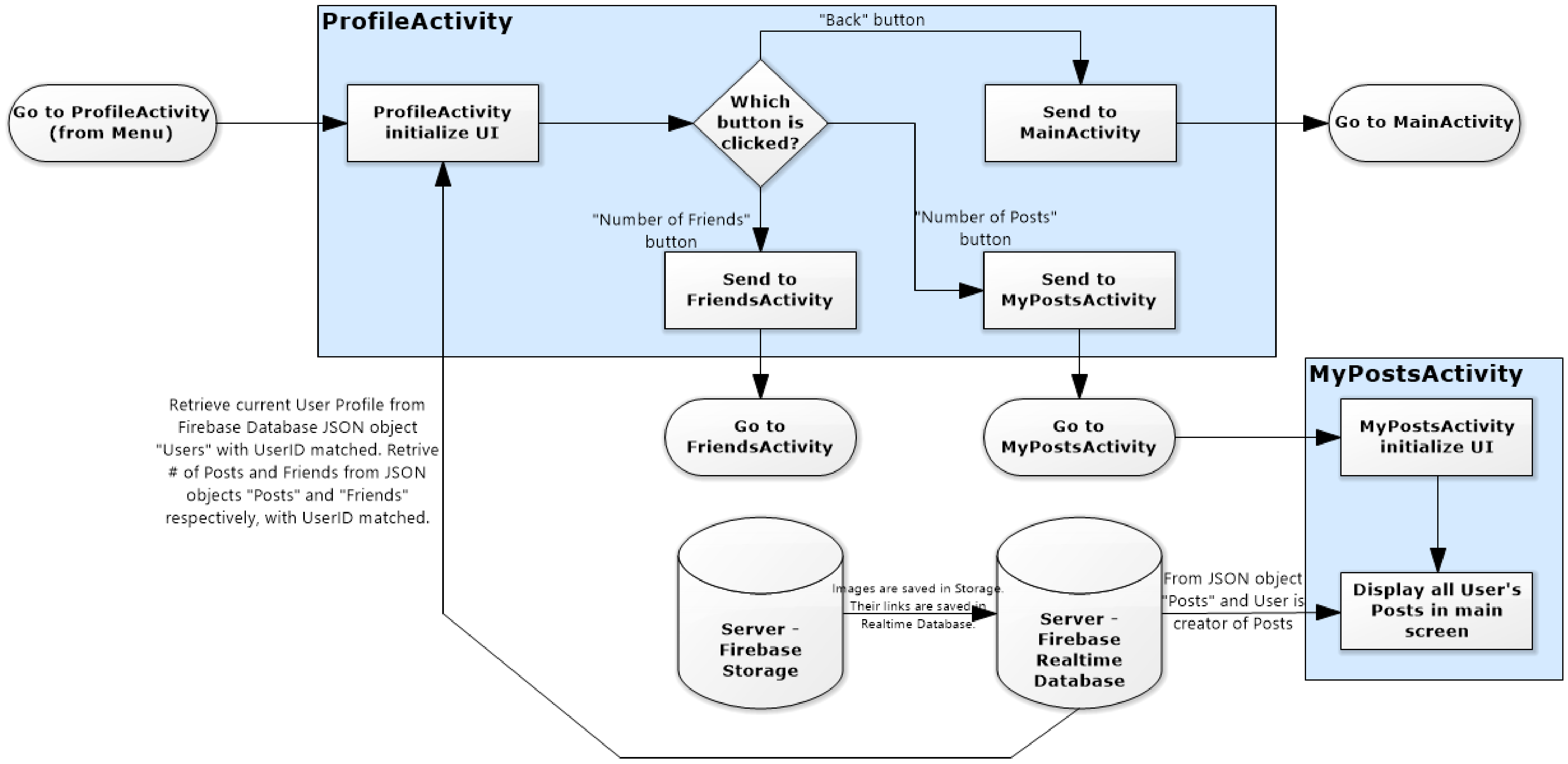
Application would download all posts if users submit a valid userID to server, i.e. user successfully login. The posts are fetched and showed using RecyclerView, which allows user to view posts even if the connection with server is lost.

*MainActivity – Navigation Menu*

User could access other functions by clicking the menu.

Application Logic – Client Side – Customize Profile

*Flowchart*


ProfileActivity
"Back" button
Go to ProfileActivity (from Menu)
ProfileActivity initialize UI
Which button is clicked?
Send to MainActivity
Go to MainActivity
"Number of Friends" button
"Number of Posts" button
Send to FriendsActivity
Send to MyPostsActivity
MyPostsActivity
Retrieve current User Profile from Firebase Database JSON object "Users" with UserID matched. Retrive # of Posts and Friends from JSON objects "Posts" and "Friends" respectively, with UserID matched.
Go to FriendsActivity
Go to MyPostsActivity
MyPostsActivity initialize UI
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database
From JSON object "Posts" and User is creator of Posts
Display all User's Posts in main screen

*ProfileActivity*

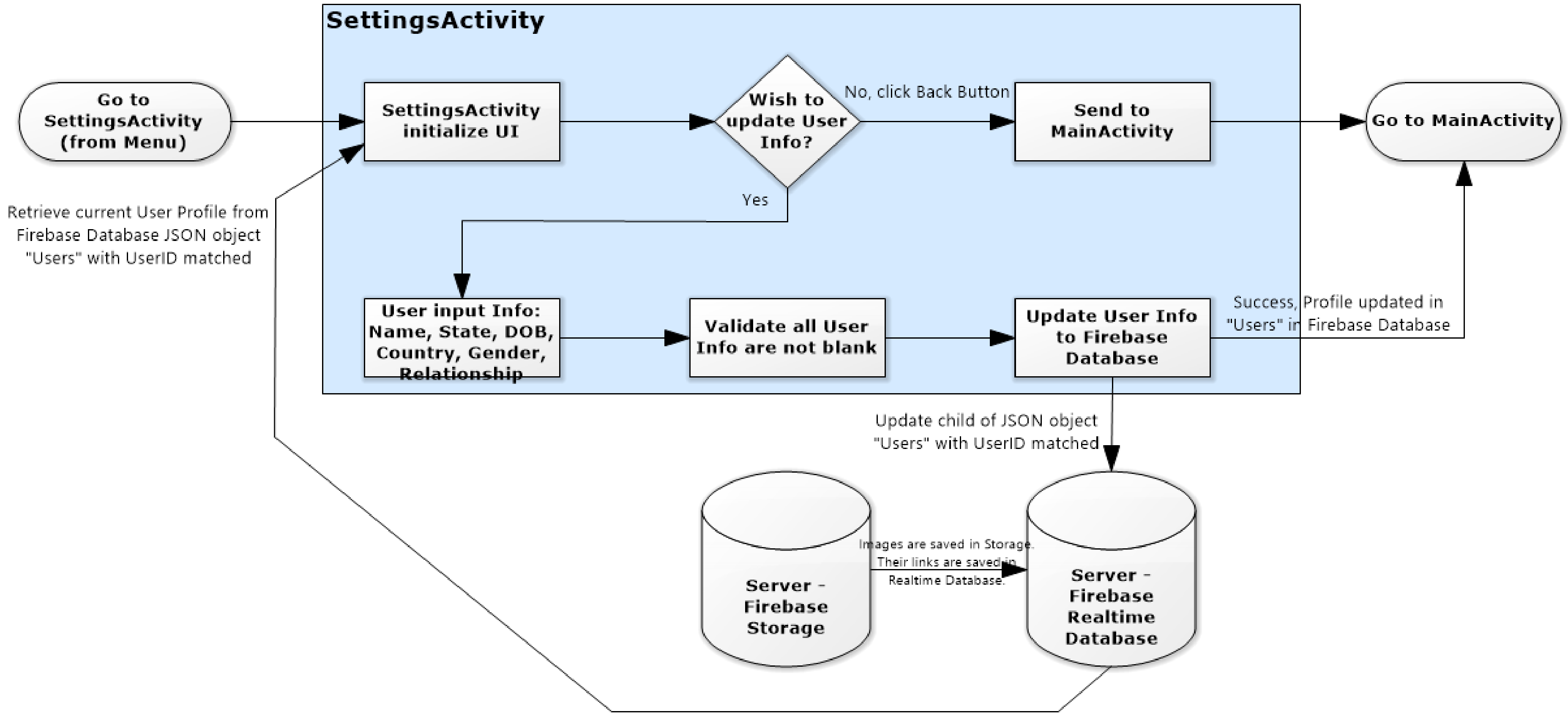
Application retrieves current user information and statistics from server and displays on screen.

*MyPostsActivity*

Application requests information of post created by current user by query (supported by Firebase Server) and displays on screen.

Application Logic – Client Side – Customize Profile (Continuous)

*Flowchart*

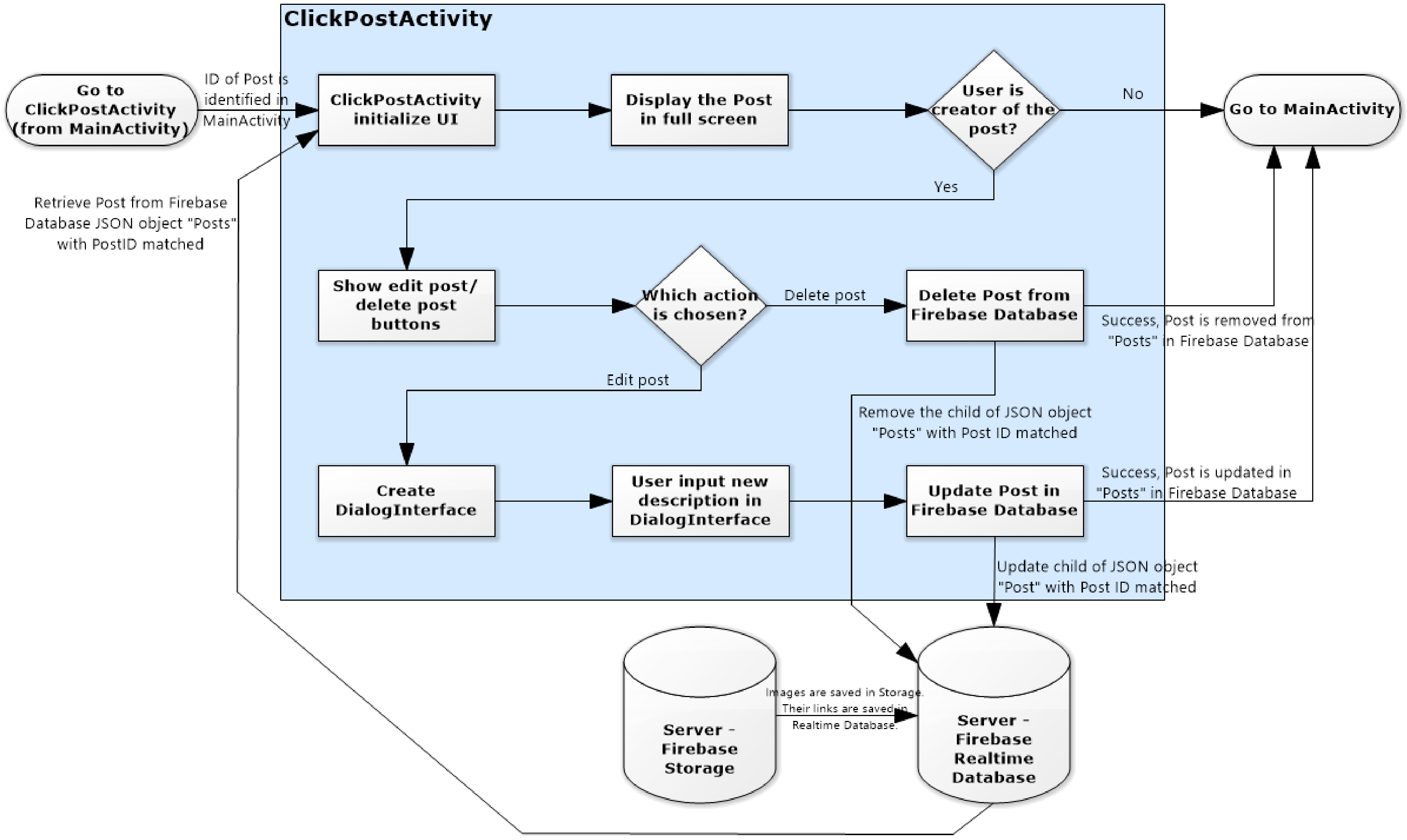

SettingsActivity
Go to SettingsActivity (from Menu)
SettingsActivity initialize UI
Wish to update User Info?
No, click Back Button
Send to MainActivity
Go to MainActivity
Yes
Retrieve current User Profile from Firebase Database JSON object "Users" with UserID matched
User input Info: Name, State, DOB, Country, Gender, Relationship
Validate all User Info are not blank
Update User Info to Firebase Database
Success, Profile updated in "Users" in Firebase Database
Update child of JSON object "Users" with UserID matched
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database

*SettingsActivity*

Application displays current user information on screen. It allows user to enter new information and replace the existing one in server.

## Application Logic – Client Side – View, Comment, Like and Create Posts

*Flowchart*



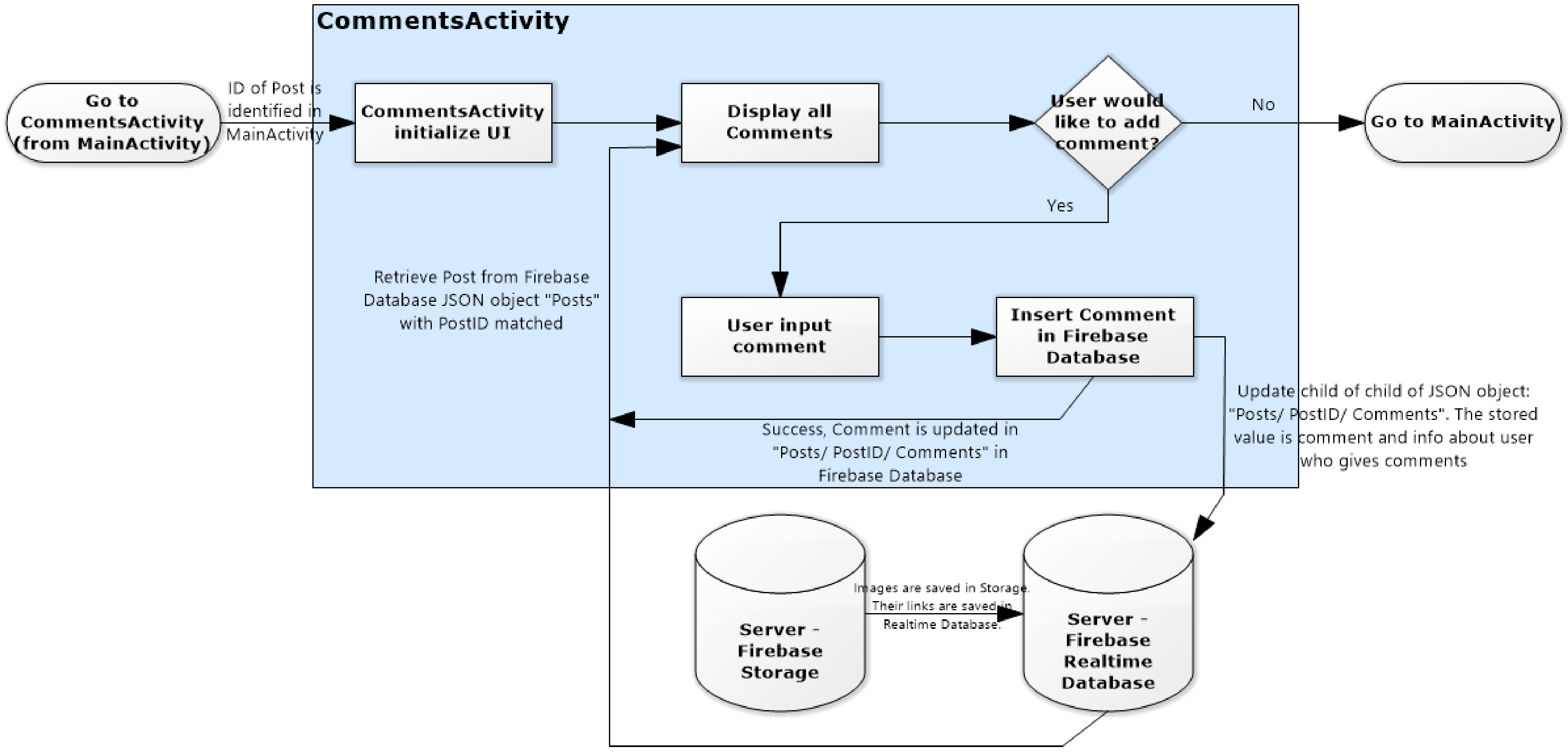
ClickPostActivity
Go to ClickPostActivity (from MainActivity)
ID of Post is identified in MainActivity
ClickPostActivity initialize UI
Display the Post in full screen
User is creator of the post?
No
Go to MainActivity
Yes
Retrieve Post from Firebase Database JSON object "Posts" with PostID matched
Show edit post/ delete post buttons
Which action is chosen?
Delete post
Delete Post from Firebase Database
Success, Post is removed from "Posts" in Firebase Database
Edit post
Remove the child of JSON object "Posts" with Post ID matched
Create DialogInterface
User input new description in DialogInterface
Update Post in Firebase Database
Success, Post is updated in "Posts" in Firebase Database
Update child of JSON object "Post" with Post ID matched
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database

*ClickPostsActivity*

All posts are show in *MainActivity*. By clicking a post, creator of the post could send request to server for edit or delete the it. If other users click a post, they could see the post in full screen. Edit and deleted functions are invisible for them.

Application Logic – Client Side – View, Comment, Like and Create Posts (Continuous)

*Flowchart*

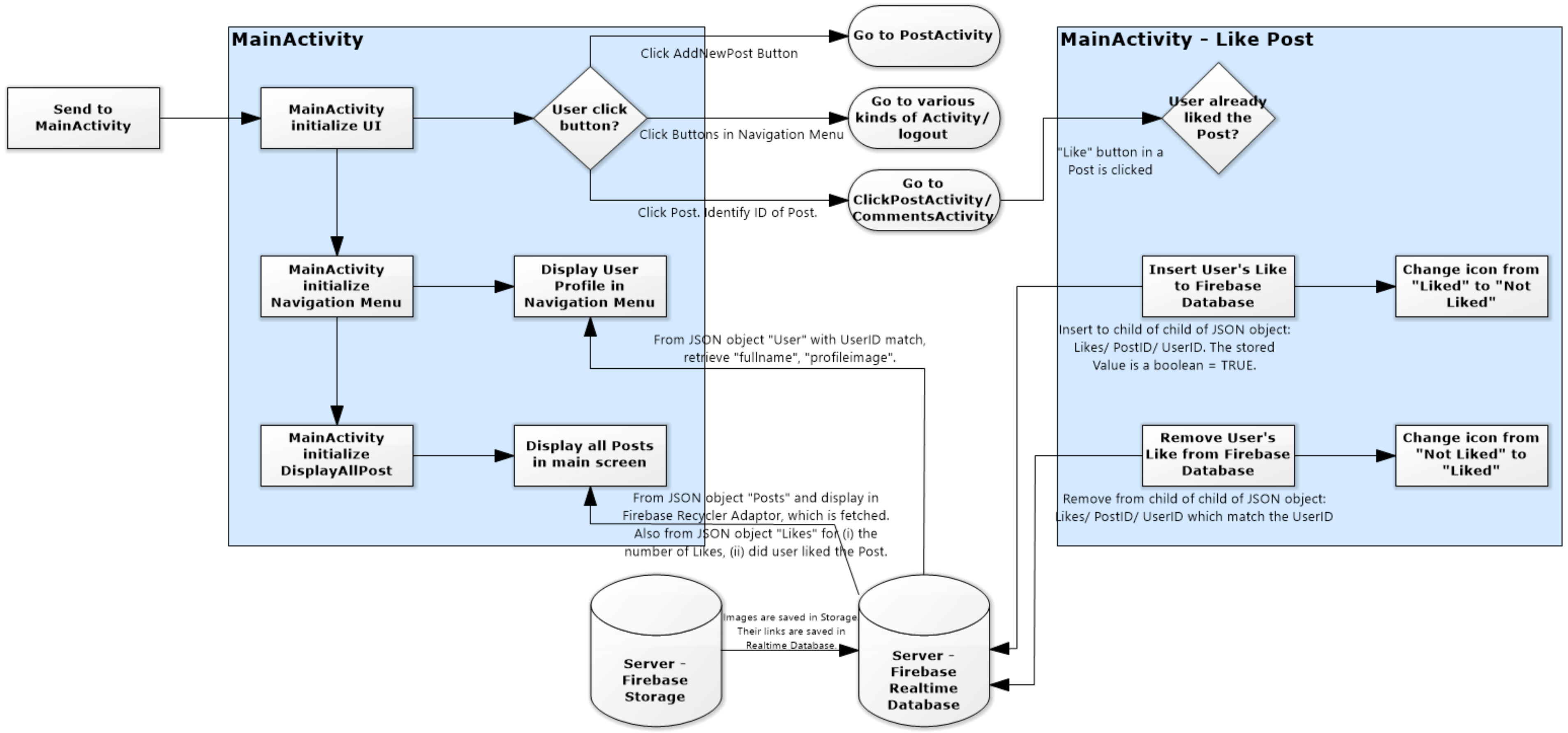

CommentsActivity
Go to CommentsActivity (from MainActivity)
ID of Post is identified in MainActivity
CommentsActivity initialize UI
Display all Comments
User would like to add comment?
No
Go to MainActivity
Yes
Retrieve Post from Firebase Database JSON object "Posts" with PostID matched
User input comment
Insert Comment in Firebase Database
Update child of child of JSON object: "Posts/ PostID/ Comments". The stored value is comment and info about user who gives comments
Success, Comment is updated in "Posts/ PostID/ Comments" in Firebase Database
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database

*CommentsActivity*

User could leave comments in a post. Comments are child of posts, which are in text format. Comments are showed only if user clicks the comment button.

Application Logic – Client Side – View, Comment, Like and Create Posts (Continuous)

*Flowchart*

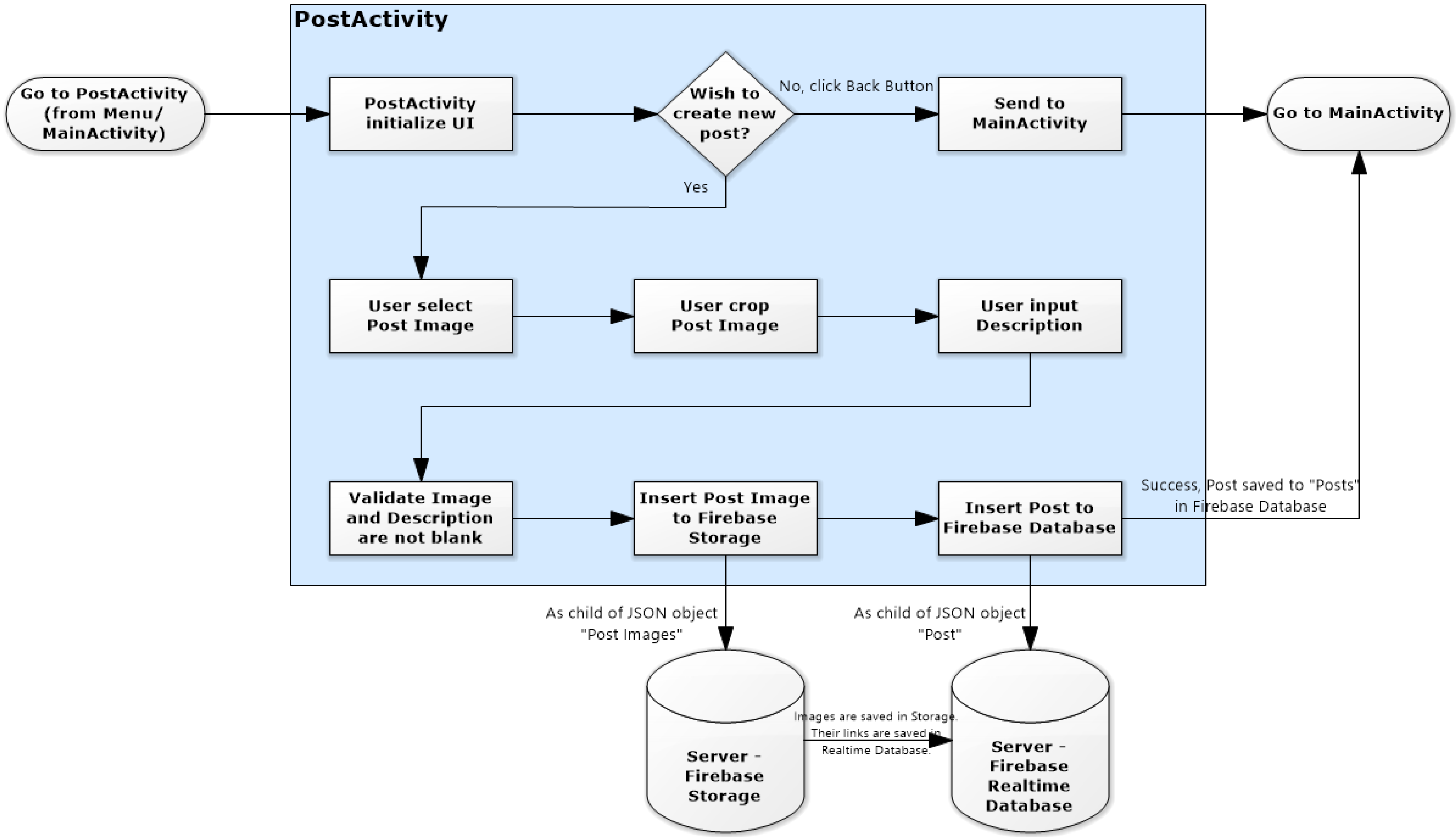

MainActivity
Send to MainActivity
MainActivity initialize UI
User click button?
Click AddNewPost Button
Go to PostActivity
Click Buttons in Navigation Menu
Go to various kinds of Activity/ logout
Click Post. Identify ID of Post.
Go to ClickPostActivity/ CommentsActivity
MainActivity initialize Navigation Menu
Display User Profile in Navigation Menu
From JSON object "User" with UserID match, retrieve "fullname", "profileimage".
MainActivity initialize DisplayAllPost
Display all Posts in main screen
From JSON object "Posts" and display in Firebase Recycler Adaptor, which is fetched. Also from JSON object "Likes" for (i) the number of Likes, (ii) did user liked the Post.
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database
MainActivity - Like Post
"Like" button in a Post is clicked
User already liked the Post?
Insert User's Like to Firebase Database
Change icon from "Liked" to "Not Liked"
Insert to child of child of JSON object: Likes/ PostID/ UserID. The stored Value is a boolean = TRUE.
Remove User's Like from Firebase Database
Change icon from "Not Liked" to "Liked"
Remove from child of child of JSON object: Likes/ PostID/ UserID which match the UserID

*Like Post*

Like post is a function in *MainActivity*. If user click like button, a record in inserted in the “Likes” table of database. Application counts the number of records and shows the number of Like for a post. Different from comments, “Likes” is not child of post. It is accessed more frequently because number of Like is counted when *MainActivity* is loaded.

<u>Application Logic – Client Side – View, Comment, Like and Create Posts (Continuous)</u>

*Flowchart*

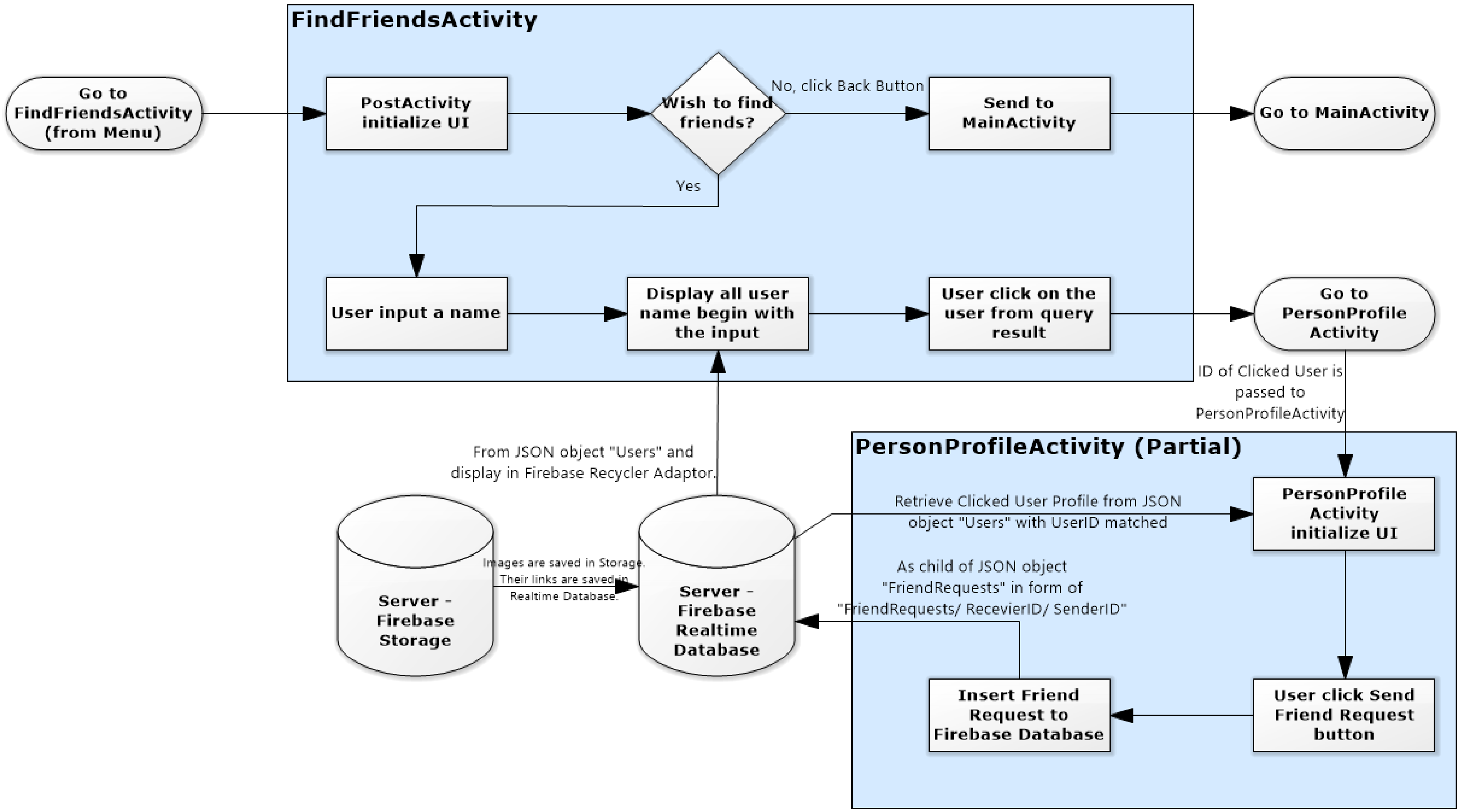

PostActivity
Go to PostActivity (from Menu/ MainActivity)
PostActivity initialize UI
Wish to create new post?
No, click Back Button
Send to MainActivity
Go to MainActivity
Yes
User select Post Image
User crop Post Image
User input Description
Validate Image and Description are not blank
Insert Post Image to Firebase Storage
Insert Post to Firebase Database
Success, Post saved to "Posts" in Firebase Database
As child of JSON object "Post Images"
As child of JSON object "Post"
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database

*PostActivity*

User could create a post by selecting image, entering description and inserting the post into server. If it is success, the post could be retrieved in *MainActivity*, edited in *ClickPostActivity*, commented in *CommentsActivity* and liked by *Like Post Function in MainActivity*.

<u>Application Logic – Client Side – Friends and Messages</u>

*Flowchart*

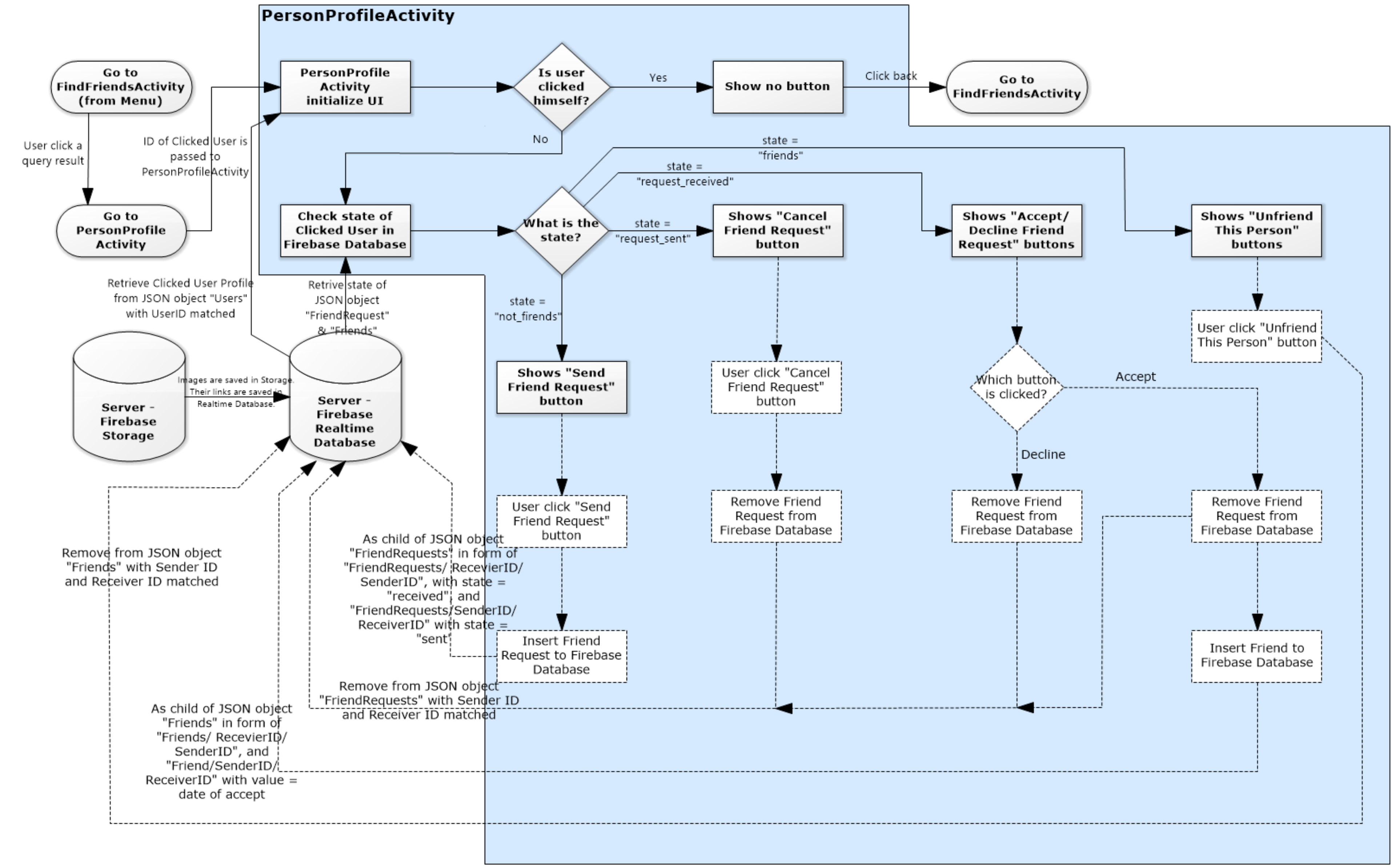

FindFriendsActivity
Go to FindFriendsActivity (from Menu)
PostActivity initialize UI
Wish to find friends?
No, click Back Button
Send to MainActivity
Go to MainActivity
Yes
User input a name
Display all user name begin with the input
User click on the user from query result
Go to PersonProfile Activity
ID of Clicked User is passed to PersonProfileActivity
From JSON object "Users" and display in Firebase Recycler Adaptor.
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database
PersonProfileActivity (Partial)
Retrieve Clicked User Profile from JSON object "Users" with UserID matched
PersonProfile Activity initialize UI
As child of JSON object "FriendRequests" in form of "FriendRequests/ RecevierID/ SenderID"
Insert Friend Request to Firebase Database
User click Send Friend Request button

*FindFriendActivity*

Sender have to be a friend of receiver for sending a message, which is done in *FindFriendActivity*. Application sends query to server to request the ID of intended receiver. Then application insert a record in “FriendRequests” table in *PersonProfileActivity*.

Application Logic – Client Side – Friends and Messages (Continuous)

*Flowchart*

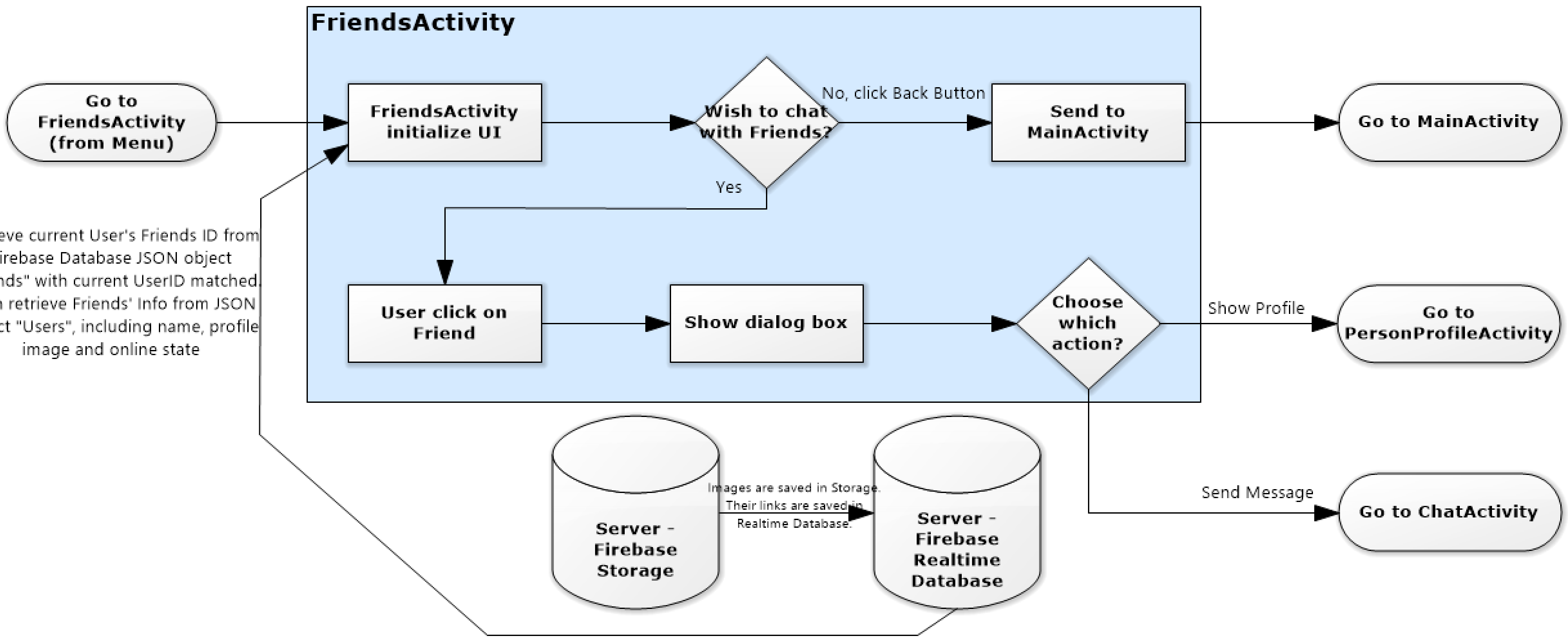

PersonProfileActivity
Go to FindFriendsActivity (from Menu)
User click a query result
Go to PersonProfile Activity
ID of Clicked User is passed to PersonProfileActivity
PersonProfile Activity initialize UI
Is user clicked himself?
Yes
Show no button
Click back
Go to FindFriendsActivity
No
Check state of Clicked User in Firebase Database
What is the state?
state = "friends"
state = "request_received"
state = "request_sent"
Shows "Cancel Friend Request" button
Shows "Accept/ Decline Friend Request" buttons
Shows "Unfriend This Person" buttons
state = "not_firends"
Shows "Send Friend Request" button
Retrieve Clicked User Profile from JSON object "Users" with UserID matched
Retrive state of JSON object "FriendRequest" & "Friends"
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database
User click "Cancel Friend Request" button
Which button is clicked?
Accept
Decline
User click "Unfriend This Person" button
User click "Send Friend Request" button
Remove Friend Request from Firebase Database
Remove Friend Request from Firebase Database
Remove Friend Request from Firebase Database
Insert Friend Request to Firebase Database
Insert Friend to Firebase Database
Remove from JSON object "Friends" with Sender ID and Receiver ID matched
As child of JSON object "FriendRequests" in form of "FriendRequests/ RecevierID/ SenderID", with state = "received", and "FriendRequests/SenderID/ ReceiverID" with state = "sent"
Remove from JSON object "FriendRequests" with Sender ID and Receiver ID matched
As child of JSON object "Friends" in form of "Friends/ RecevierID/ SenderID", and "Friend/SenderID/ ReceiverID" with value = date of accept

*PersonProfileActivity*

Friend requests are processed here. Buttons are clicked to switch between states: not friend, friend, to process friend request (sender or receiver).

Application Logic – Client Side – Friends and Messages (Continuous)

*Flowchart*

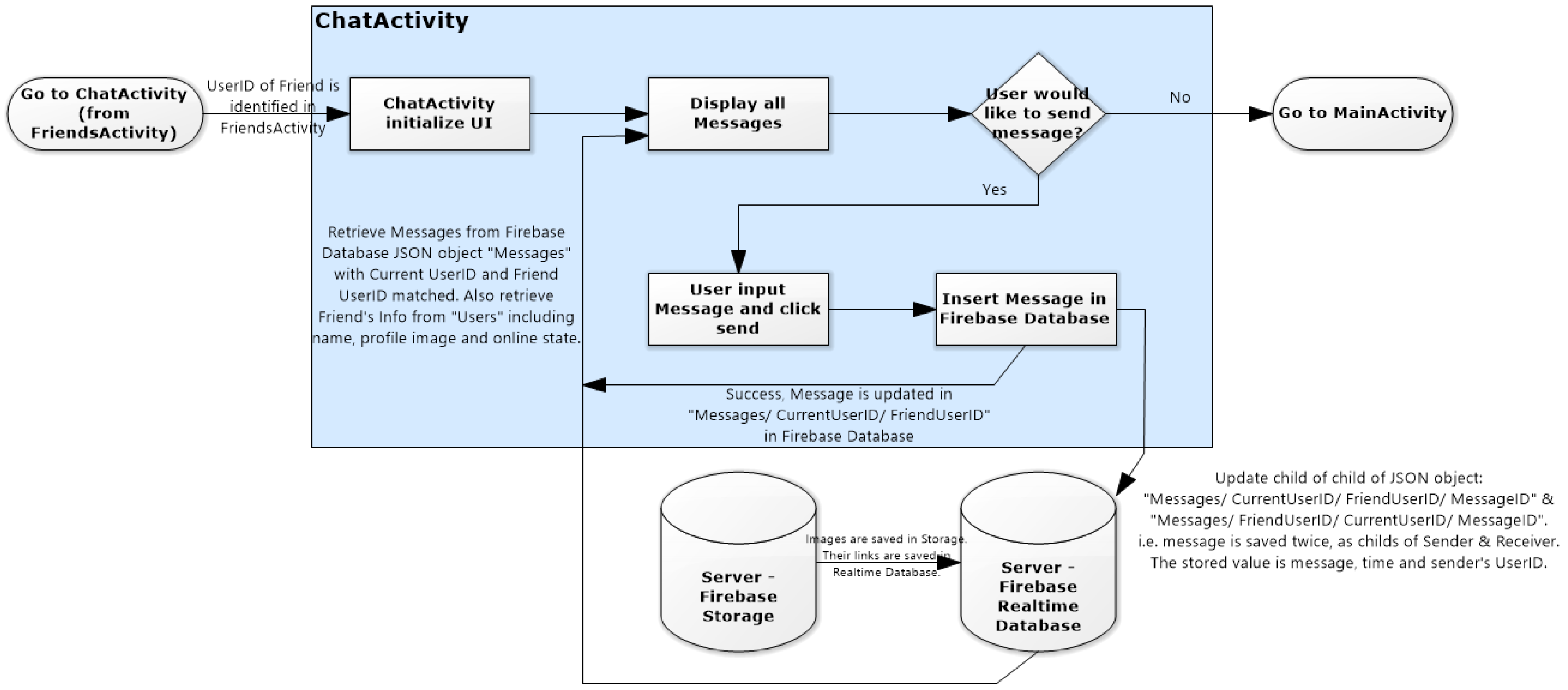

FriendsActivity
Go to FriendsActivity (from Menu)
FriendsActivity initialize UI
Wish to chat with Friends?
No, click Back Button
Send to MainActivity
Go to MainActivity
Yes
User click on Friend
Show dialog box
Choose which action?
Show Profile
Go to PersonProfileActivity
Send Message
Go to ChatActivity
Retrieve current User's Friends ID from Firebase Database JSON object "Friends" with current UserID matched. Then retrieve Friends' Info from JSON object "Users", including name, profile image and online state
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database

*FriendActivity*

Application retrieves basic profile information of a friend. User could click buttons to send message or view full profile information.

Application Logic – Client Side – Friends and Messages (Continuous)

*Flowchart*


ChatActivity
Go to ChatActivity (from FriendsActivity)
UserID of Friend is identified in FriendsActivity
ChatActivity initialize UI
Display all Messages
User would like to send message?
No
Go to MainActivity
Yes
Retrieve Messages from Firebase Database JSON object "Messages" with Current UserID and Friend UserID matched. Also retrieve Friend's Info from "Users" including name, profile image and online state.
User input Message and click send
Insert Message in Firebase Database
Success, Message is updated in "Messages/ CurrentUserID/ FriendUserID" in Firebase Database
Server - Firebase Storage
Images are saved in Storage. Their links are saved in Realtime Database.
Server - Firebase Realtime Database
Update child of child of JSON object: "Messages/ CurrentUserID/ FriendUserID/ MessageID" & "Messages/ FriendUserID/ CurrentUserID/ MessageID". i.e. message is saved twice, as childs of Sender & Receiver. The stored value is message, time and sender's UserID.

*ChatActivity*

Application create a RecyclerView to show all messages. User could enter message and insert it in database. Application will download the message from database in simultaneously.

## Application Logic – Client Side – Disconnection Management

### *Fetching in RecyclerView*

The application is an Instagram-like application, which require users to have continuous connection with server to access all function. Powerful local data storage is unnecessary for this application. Hence, no local database is built for copying information from server. Instead some fetching techniques are used, which allows users to view the content even there is a short connection loss with the server.

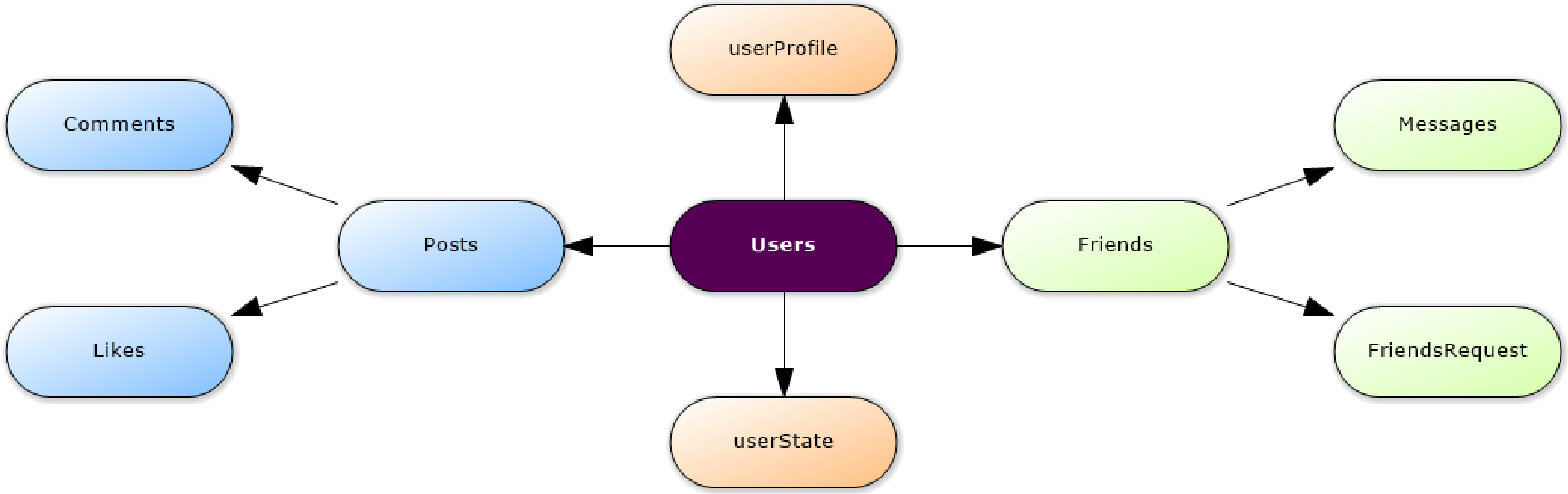
RecyclerView is a build-in library in android for display a scrolling list of elements based on large data sets (or data that frequently changes). It automatically cached data from server. To read the data, a RecyclerView.Adapter should be built. All data are temporary cached, and user could continuous browsing even connection is lost. However, creating new posts or changing profile information is not allowed in this offline mode.

**Application Logic and Disconnection Management – Server Side**

Our application server is supported by Firebase. Most of the logic are done by Firebase, including
- User registration and Login
- Inserting records in database (JSON) and random key generation
- Monitoring connection with clients

For server side, our focus in on creating suitable database structure and accessing the functions supported by Firebase.

*Overview of server-side database structure*

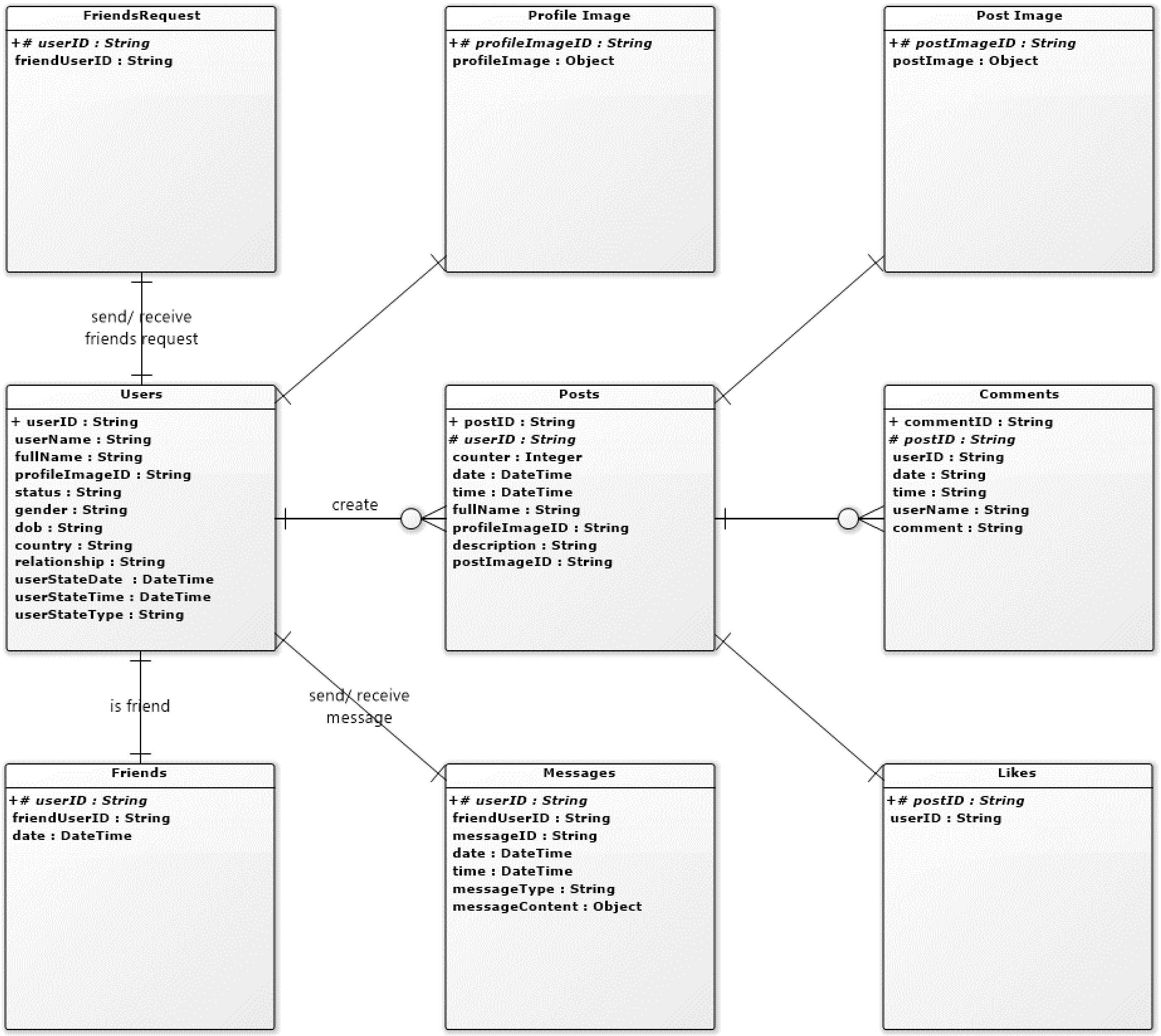


Matters related user identification are solved by Firebase build-in functions. For handling the other three types of client-side requests, three types of tables are created in server-side.
- Orange: Customize user profile
- Green: View, comment, like and create posts
- Blue: Sending messages to friends

Data are stored in the noSQL database in Firebase using JSON in tree-structure. However, for easier application development, the client-side application will access database by assuming that it no differences with relational database. Firebase supports query processing which is similar to SQL.

## Application Logic – Server Side – Database

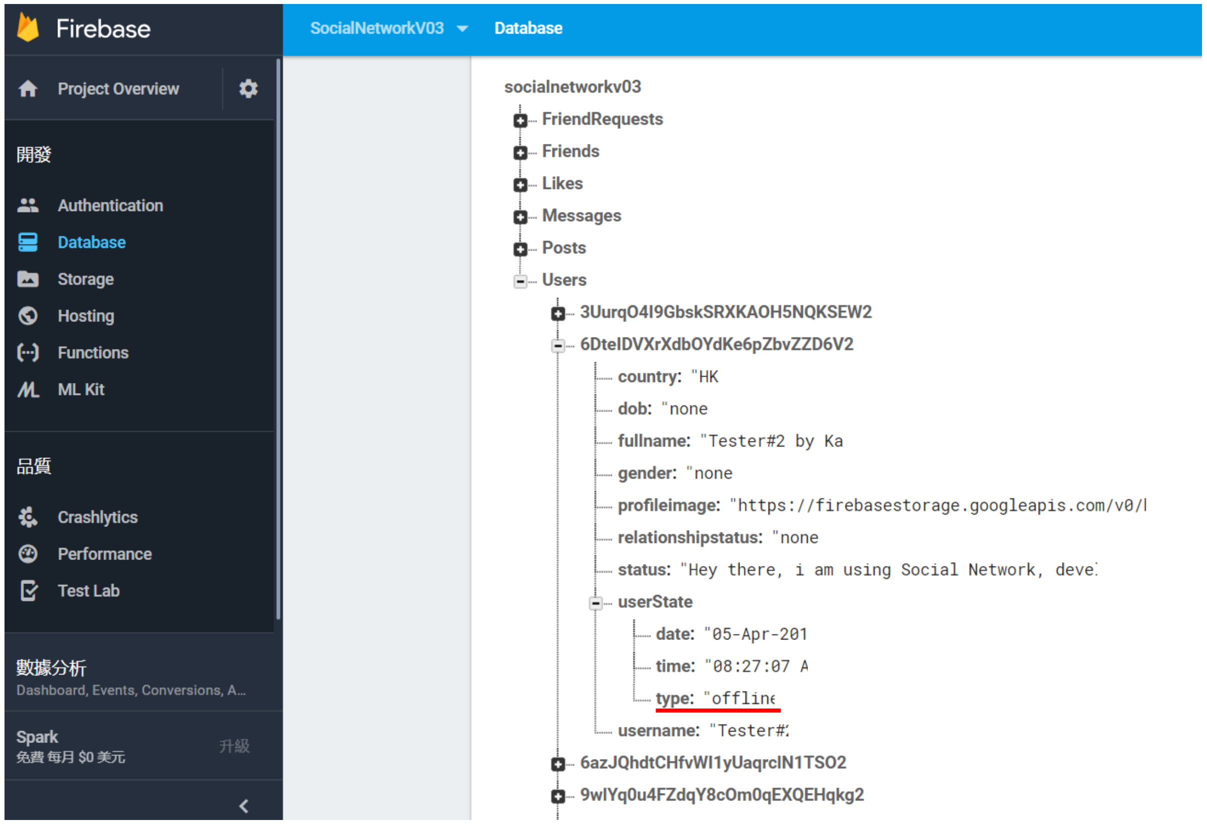
*ER Diagram for Details of Database*



## Application Logic – Server Side – Disconnection Management

*User State Tracking in Database, and Update Database by OnDestory Method*

In Firebase database, there is an attribute in "Users" for tracking users' connection.

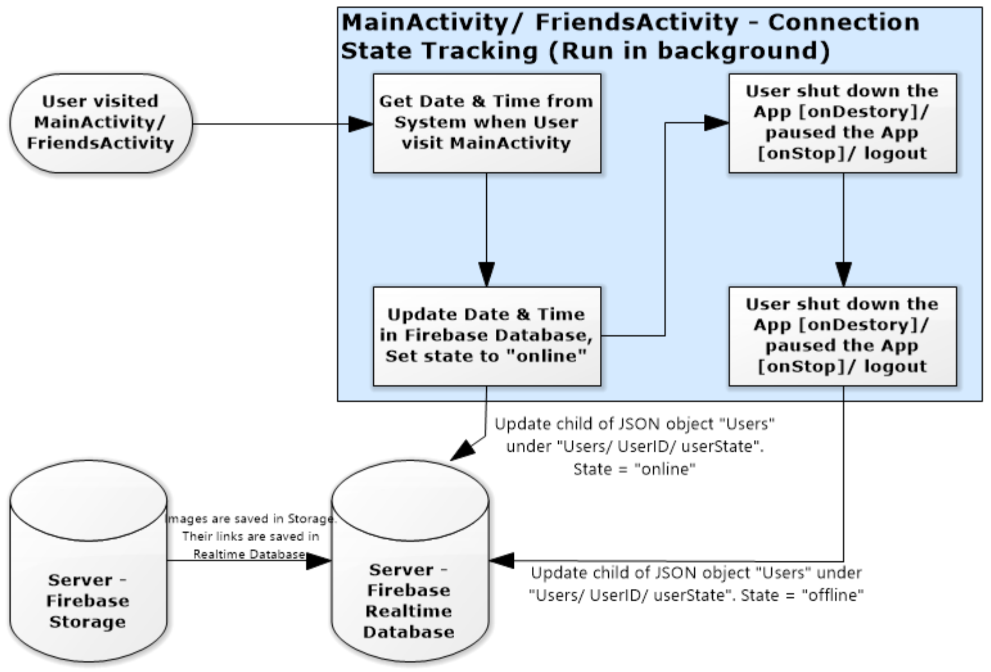


```
socialnetworkv03
  FriendRequests
  Friends
  Likes
  Messages
  Posts
  Users
    3Uurq04I9GbskSRXKAOH5NQKSEW2
    6DteIDVXrXdbOYdKe6pZbvZZD6V2
      country: "HK
      dob: "none
      fullname: "Tester#2 by Ka
      gender: "none
      profileimage: "https://firebasestorage.googleapis.com/v0/l
      relationshipstatus: "none
      status: "Hey there, i am using Social Network, deve
      userState
        date: "05-Apr-201
        time: "08:27:07 A
        type: "offline
      username: "Tester#
    6azJQhdtCHfvWI1yUaqrcIN1TSO2
    9wIYq0u4FZdqY8cOm0qEXQEHqkg2
```

Once the client-side application is connected to server and *MainActivity/ FriendActivity* is loaded, client would update this attribute to "online". Firebase could monitor the connection state of users. On client-side, functions are written to tell Firebase to switch the attribute to "offline" when the application is closed or crashed.

Application Logic – Server Side – Disconnection Management (Continuous)

*User State Tracking in Database, and Update Database by OnDestory Method (Continuous)*


MainActivity/ FriendsActivity - Connection State Tracking (Run in background)
User visited MainActivity/ FriendsActivity
Get Date & Time from System when User visit MainActivity
Update Date & Time in Firebase Database, Set state to "online"
User shut down the App [onDestory]/ paused the App [onStop]/ logout
User shut down the App [onDestory]/ paused the App [onStop]/ logout
Update child of JSON object "Users" under "Users/ UserID/ userState". State = "online"
Update child of JSON object "Users" under "Users/ UserID/ userState". State = "offline"
Images are saved in Storage. Their links are saved in Realtime Database
Server - Firebase Storage
Server - Firebase Realtime Database

The method *OnDestory* is used for updating the Firebase database. If the application is closed or crashed, a message will be sent to Firebase database to switch the state to offline.

Our application is an Instagram-like application. Most of the client-side functions could be perform only if there is a connection with server. Our goal is to ensure that server has the most updated data only. For case that client loss connection with server, our solution is to stop the services provided to client to prevent any inconsistent of data. Hence, this connection management initialized by client is sufficient for our application.

Since the all data are stored in server, clients could always download the missed data from server when connection is established again. It is unnecessary for the server to perform tasks to guarantee that all clients are connected and have the most updated data.

[After]