applied sciences

MDPI

Article

Cloud Database Analysis of Instant Messaging Apps on Mobile Devices

Roland Szabo

Faculty of Electronics, Telecommunications and Information Technologies, Politehnica University Timisoara, Vasile Parvan Av., No. 2, 300223 Timisoara, Romania; roland.szabo@upt.ro; Tel.: +40-256-40-3351

Abstract: The purpose of this paper is to present the creation, challenges and difficulties of four utility apps to increase production on Android handheld devices. The motivation of creating the apps and the problems encountered during creation are highlighted. The first app is a fast emoji app for WhatsApp. The motivation for creating such an app is that usually phone users do not like to type too much; with this app, they can communicate with each other just by sending emojis. The second app is a standalone emoji sender app, which does the same thing as the previous app but uses its own database, not the WhatsApp database.

Keywords: databases; electronic messaging; instant messaging; mobile applications; mobile computing

1. Introduction

1.1. Fast Emoji Sender for WhatsApp

The novelty of this app is that it is an instant messaging app without character input for WhatsApp. The user has to tap once on a contact and twice on the chosen emoji. With only double tapping, a user can send their feelings to a friend via WhatsApp. An app like this was not found on the Google Play Store or the Apple App Store.

The motivation behind creating a fast emoji sender for WhatsApp is that the user does not like to type too much, so it is an easier and more convenient way to communicate with just emojis [1]. In Unicode version 15.1., there are a total of 3782 emoji symbols available [2], which is enough to communicate using the most successful app [3]. And the apps most preferred by users are the simplest apps, the ones with only one button [4]. No matter what technology is behind the app, if the user interface is unfriendly [5], the app will just not become successful [6]. This app loads user contacts and can send an emoji to a selected contact through WhatsApp [7].

The fast emoji sender uses WhatsApp cloud storage. The novelty of the app is that it uses a faster way to send emojis in case the user has less time to chat or is not able to, being preoccupied with something else.

1.2. Standalone Fast Emoji Sender

The novelty of this app is that it is a non-character-typing standalone messaging app. The functioning mechanism is similar to the previous app; by double tapping, a user can send their feelings to their friend. However, having a standalone app, there is no need to have WhatsApp or other apps installed. An app like this was not found on the Google Play Store or the Apple App Store.

Academic Editors: Douglas O'Shaughnessy and Chih-Cheng Lu

Received: 15 November 2024
Revised: 15 January 2025
Accepted: 29 January 2025
Published: 2 February 2025

Citation: Szabo, R. Cloud Database Analysis of Instant Messaging Apps on Mobile Devices. *Appl. Sci.* **2025**, *15*, 1509. https://doi.org/10.3390/app15031509

The standalone fast emoji sender is similar to the previous application, but it will not use WhatsApp servers to send emojis; it will send emojis through its own server [8]. The idea is that maybe there are some users [9] who do not use WhatsApp, so they do not have WhatsApp installed and thus the previous app will not work, but the standalone fast emoji sender will work [10]. The motivation behind this app is similar to that of the previous app [11]. Users do not like to type too much, but would still like to communicate in a simple and fast way [12].

The standalone fast emoji sender uses its own cloud database to send emojis to friends. The database is a standard SQL database that is used in many common instant messaging apps.

This topic is important because these applications are unique, have been made by the author from scratch, and there are no apps like these on the Google Play Store.

In their 2019 study, Johansen et al. conducted a comparative analysis of secure instant messaging mobile apps, with a focus on end-to-end encryption protocols. The research evaluated different applications in terms of security and usability and found that although many apps utilize strong encryption, there are differences in their security and usability offerings. The study concluded with suggestions for improving these features in instant messaging applications [13].

In 2020, Bahramali et al. examined how vulnerable secure messaging apps such as Telegram, Signal, and WhatsApp are to traffic analysis attacks. Their research demonstrated that even with strong encryption, these instant messaging platforms might expose sensitive user data through encrypted traffic analyzed. The authors highlighted the importance of adopting robust traffic obfuscation strategies to protect user privacy [14].

Singh et al. (2021) introduced an end-to-end encryption framework utilizing blockchain to tackle the weaknesses in existing messaging apps. The strategy used blockchain to oversee the public key infrastructure, focusing on boosting user data confidentiality and security without relying on centralized servers. This pioneering approach aimed to address the current vulnerabilities in messaging applications [15].

In their 2024 study on Snapchat, Fang et al. used behavioral logs to create and assess user representations for broad applications. They developed a Transformer-based model that generates superior representations for instant messaging platforms, emphasizing the importance of grasping user interactions to improve both user experience and engagement [16].

2. Problem Formulation

2.1. Fast Emoji Sender for WhatsApp

The goal was to create the WhatsApp emoji sender app. Initially, a ListView is needed that will load all the phone contacts from the user's phone. The phone contacts will load after the user grants permission for this. When tapping on a contact, a pop-up window will appear with all the emojis. When the user taps on an emoji, the app will open WhatsApp and send the emoji to the selected contact through the WhatsApp servers. The cloud database used here is the WhatsApp database servers.

2.2. Standalone Fast Emoji Sender

The standalone fast emoji sender is similar to the previous application, but it needs its own server to send emojis and will not use WhatsApp servers to do this. This has an advantage for users who do not have WhatsApp installed. Here, the app is more complex because it requires the creation of an account in a database. The user has to register to use the app. The user has to register with their username, real name and password and then they can log in with their username and password. The user can add friends and block

friends. The user can also edit their profile by changing their real name and password. The user can also delete their profile and all their activities in the app. Otherwise, the app works similarly to the previous app; the user can tap on a friend, a pop-up will appear with the emojis, and an emoji is sent to a friend after the user taps on a selected emoji. The standalone fast emoji sender will send the emojis to its own SQL cloud database on its own personal server.

3. Problem Solution

All of the apps were written in the Kotlin programming language. Since 2019, Kotlin has been the preferred language for developing Android applications. Google encourages the usage of this programming language designed by the JetBrains team. In the latest versions of Android Studio, there is not even an option to create a project using the Java programming language, although Java classes can still be included alongside Kotlin classes in the project structure. The Kotlin programming language is strongly based on Java, but is more simplified and modern. The programming language is also quite new; it first appeared in 2011. Although more documentation on the Internet about the Java programming language can still be found, Kotlin has started to grow. Google introduced one of the best code converters in Android Studio, converting from Java to Kotlin programming language, making the creation of apps really fun.

3.1. Dynamic Modeling of Instant Messaging Behavior

The following variables are taken into account:

- $M(t)$: the total count of messages dispatched over time t;
- $U(t)$: the active user count at a given time t;
- $J(t)$: the rate at which new users are joining over time t;
- $C(t)$: the user churn rate over time t;
- $E(t)$: the user participation rate during the period t.

The number of messages exchanged and user interactions can be represented by the following system of differential equations:

$$\frac{dM(t)}{dt} = \alpha U(t)E(t) \tag{1}$$

Here, α denotes a fixed value that reflects the typical number of messages exchanged per user per engagement.

The user population $U(t)$ changes over time due to the rates of users joining and leaving:

$$\frac{dU(t)}{dt} = J(t) - C(t) \tag{2}$$

The engagement rate can be influenced by external elements, such as new features or promotions, denoted by the term $\beta(t)$:

$$E(t) = E_0 e^{-\gamma t} + \beta(t) \tag{3}$$

The overall count of messages dispatched during the interval from t_0 to t_f can be calculated as follows:

$$M_{\text{total}} = \int_{t_0}^{t_f} \alpha U(t)E(t)\, dt \tag{4}$$

The graphs illustrating the user population, the engagement rate and the total number of messages accumulated over time.

The graphs in Figures 1–3 represent an assumption of how the app can behave after being launched on Google Play. If the marketing strategy goes well, this is the expected user growth in the next period.

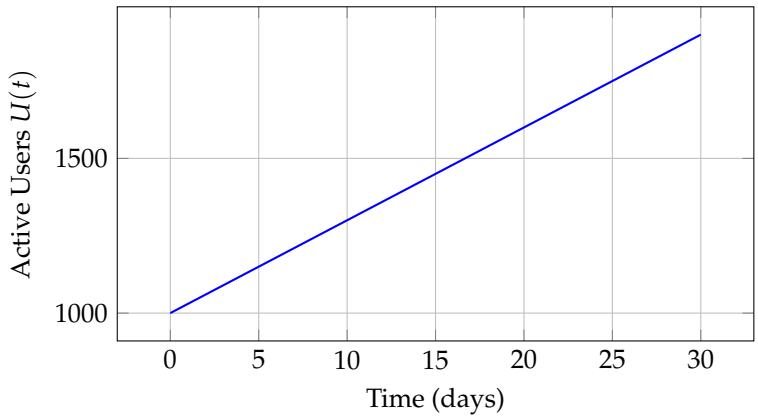

Figure 1. User population over time.

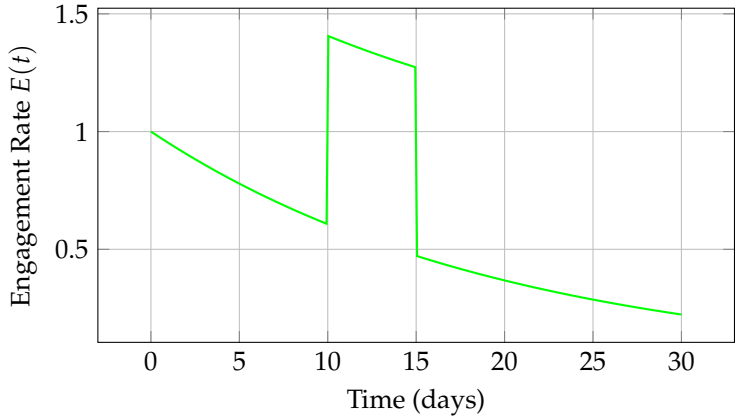

Figure 2. Engagement rate over time.

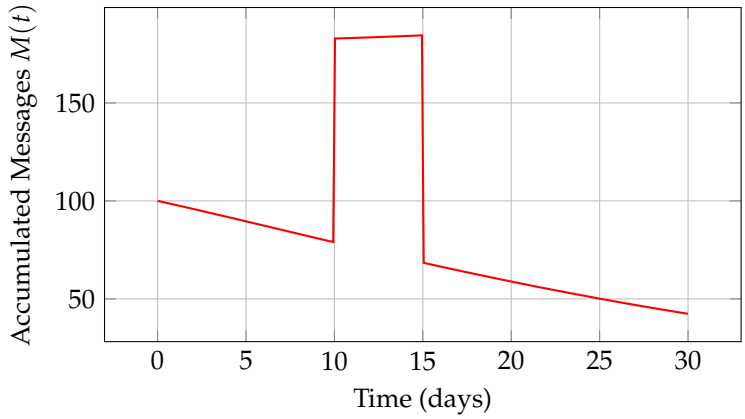

Figure 3. Accumulated messages over time.

We will assume certain constants and model–user interactions and message exchanges over a specific period. The following is what we will simulate:

Count of active users $U(t)$—influenced by the rate of joining $J(t)$ and the rate of churning $C(t)$.

Message frequency $M(t)$—corresponding to the user count and their engagement levels.

Engagement frequency $E(t)$—user fatigue leads to a decline over time, although external elements can cause occasional spikes.

Active users $U(t)$—the count of active users steadily increases over time because the rate at which users join exceeds the rate at which they leave.

Engagement frequency $E(t)$—user engagement initially declines over time but increases during the promotional period (days 10–15) due to external events that increase user activity.

Dispatched messages $M(t)$—the overall quantity of messages exchanged increases progressively, with a significant uptick observed during the promotional period when user interaction is elevated.

The forthcoming sections will utilize differential equations and integrals to characterize user growth, message frequency, and data traffic in mobile chat applications. These models aim to facilitate the analysis of both applications.

Let $U(t)$ represent the number of users at time t. A conventional model for user growth is often described by a differential equation reflecting logistic growth:

$$\frac{dU(t)}{dt} = rU(t)\left(1 - \frac{U(t)}{K}\right) \tag{5}$$

where:

- r represents the inherent expansion rate of the user base;
- K refers to the carrying capacity, which is the highest number of users that the application can accommodate.

To solve this differential equation, we obtain the following:

$$U(t) = \frac{K}{1 + \left(\frac{K-U_0}{U_0}\right)e^{-rt}} \tag{6}$$

Here, U_0 represents the number of users present initially at $t = 0$.

Let $M(t)$ represent the volume of messages transmitted over a given period. The transmission rate of messages can be represented as a function of the number of active users $U(t)$:

$$\frac{dM(t)}{dt} = aU(t) \tag{7}$$

Here, a represents the mean number of messages transmitted by each user within a given timeframe.

The cumulative count of messages sent over a period can be determined through integration:

$$M(t) = \int aU(t)\,dt \tag{8}$$

The logistic growth model is replaced with $U(t)$:

$$M(t) = \int a\frac{K}{1 + \left(\frac{K-U_0}{U_0}\right)e^{-rt}}\,dt \tag{9}$$

This integral can be calculated through numerical techniques.

Consider $D(t)$ as the data traffic produced by the app at the instant t. The data traffic rate can be characterized as being proportional to the number of messages transmitted:

$$D(t) = bM(t) \tag{10}$$

Here, b represents the mean data traffic per message.

A hypothetical table that illustrates user growth over time is provided on Table 1.

Table 1. User expansion over time.

Time t (Days)	Number of Users $U(t)$	Growth Rate $\frac{dU}{dt}$
0	1000	500
30	3000	1200
60	7000	1400
90	9500	500
120	9900	200

Figure 4 shows the growth of the user, modeled using a logistic function.

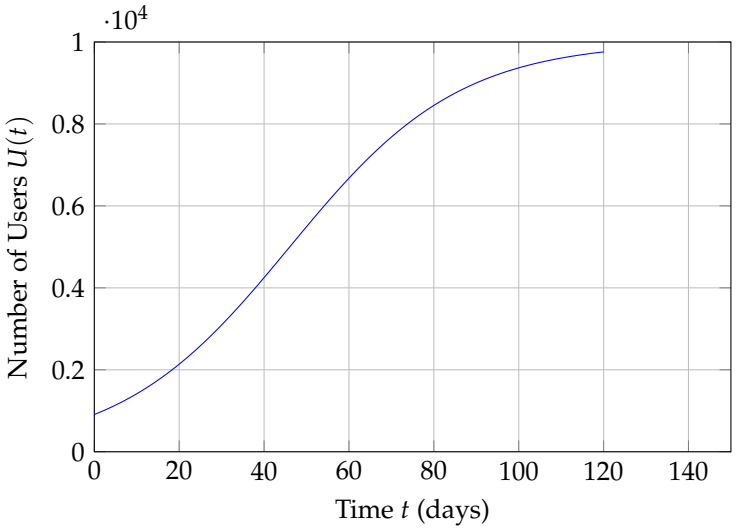

Figure 4. User expansion across time.

In Figure 5, the projected data traffic over time can be seen based on the volume of messages.

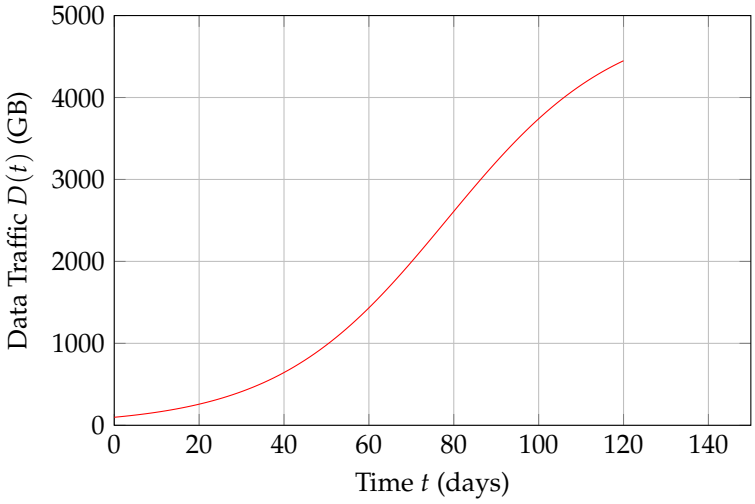

Figure 5. Evolution of data traffic over time.

The differential and integral models shed light on the temporal dynamics of chat applications. By incorporating real-world data, these models can be improved to precisely forecast user growth, message frequency, and data traffic.

3.2. Fast Emoji Sender for WhatsApp

The idea of this app is to create a no character typing instant messaging app, which can be used only to send emojis to contacts. The user can receive responses in the WhatsApp application. The app idea is based on the fact that we have less and less time, people often want to send messages while driving (which is not advised), and there are children who want to send messages and did not learn to write, as well as visually impaired people who can use a no-character-typing instant messaging app.

Actually, with this app, a user can send any emoji. Sometimes it is not about the specific emoji you sent, but that you thought about that person. Also, food trucks or ice cream machines can send emoji's to people when they are around, so it can also be used in marketing.

This app is not very complex; basically, it is made from a single Activity *.kt file with a corresponding Layout *.xml file, which loads all user contacts in ListView and sort it alphabetically (Figure 6). At the top, there is a search box with an autocomplete function; this has another Layout *.xml file. When the user starts typing, if it finds a matching contact from the user's contacts, it will show it in a drop-down box (Figure 7).

The app has also a floating green share button, which creates an Intent to share the app to friends, to send the app's Google Play link via a chosen instant messaging platform.

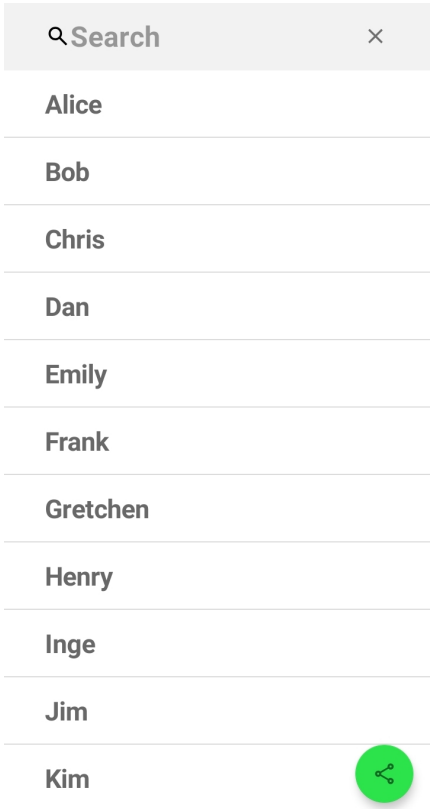

Figure 6. Loading contact list to send emoji.

When the user taps on a contact name, another Layout *.xml file with emojis will pop up (Figure 8). When the user taps on an emoji, the app will search for WhatsApp. If it is installed, it will open WhatsApp, using an Intent, to send the emoji to the selected contact via WhatsApp (Figure 9). If WhatsApp is not installed, it will ask the user to install it, because the app can only send emojis through this platform.

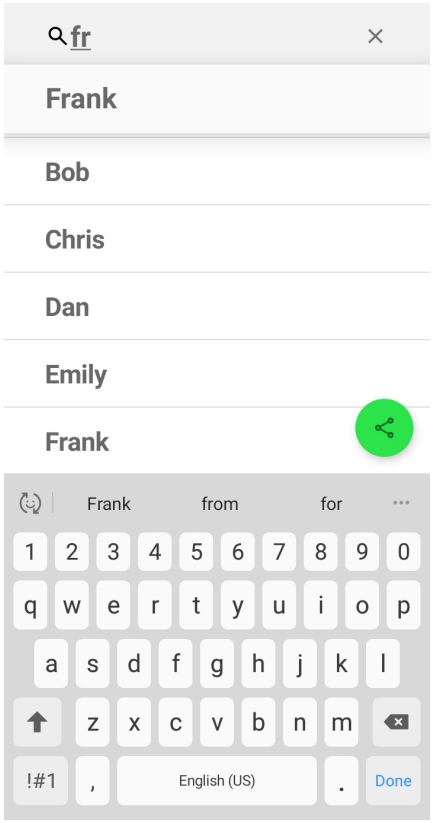

Figure 7. Searching for a contact to send emoji.

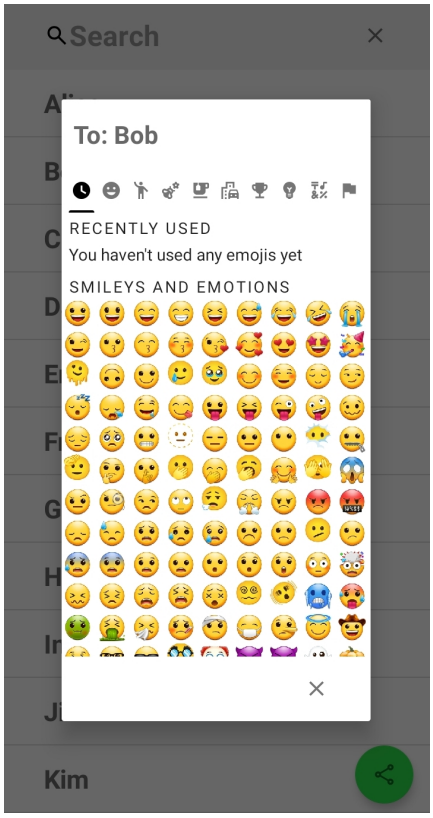

Figure 8. Select and send an emoji.

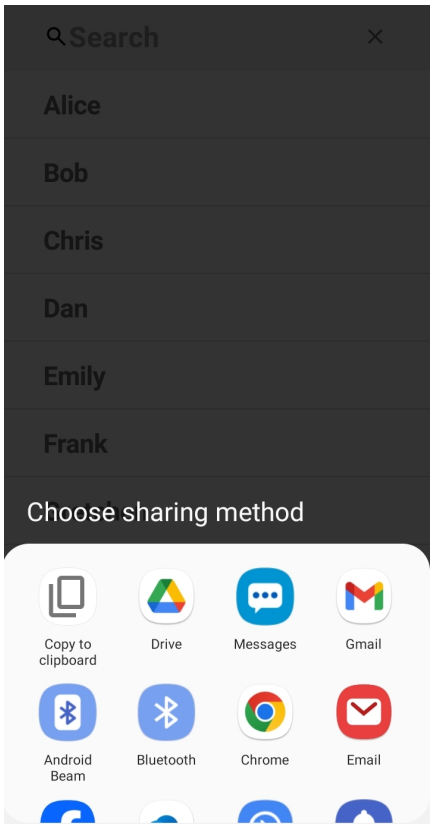

Figure 9. Share app with friends.

The idea of doing it this way (using WhatsApp to send emojis) came from multiple considerations. One is that it is quite hard to compete with applications like WhatsApp, so instead of trying to beat it, it is better to join it. Another reason is that it is quite complex to create the database part and the encryption part of sending messages from one phone to another. Another reason is that in the case of creating an app with its own databases, the app has to be installed by both parties, the sender and the receiver; this implies a big marketing cost.

If the app is made to send emojis through WhatsApp, then only the sender needs to have the app installed, but the receiver does not need to have the app installed. The receiver needs to only have WhatsApp installed, and most people will have it installed because WhatsApp is one of the most used instant messaging apps, installed on billions of handheld devices. Maybe a recipient will decide to install the app if the sender invites them to install it using the green share floating action button. Another reason for not using a separate database for this fast emoji sender app is that maintaining a database can have a high initial cost. There is no database that can be heavily used and be free; all free plans, if used a little bit more intensively, will then block the user's account.

SQL databases are also a little too complex and need an initial budget, which is not the best when an app creator is willing to give the app for free when they already paid 25 USD to publish the app on the Google Play Store. While programming an SQL database, the security is laid on the programmers' shoulders to avoid SQL injection attacks. To create an SQL bookend for an instant messaging app, the developers also need strong PHP, SQL, and JSON knowledge. For the NoSQL database, the task is a little bit simple, but the cost is not less. The Firebase free Spark plan allows for only 100 concurrent users, which is too low in case we would like to make some revenue from ads. To make a little income from ads, the lowest number of users would be at least 1000 users, but this would only break even and return the 25USD publishing fee for Google to publish the app on the Google Play Store.

The idea behind the app is for it to be really simple and fast and to send emojis to anyone with two taps: one tap on the contact and a second tap on the selected emoji. No typing is needed.

Some insights about the app would be that for the ListView, we actually have two ArrayLists of Strings, one with the contact's name and the other with the contact's phone number, while WhatsApp uses phone numbers to send messages to users via the internet. The ListView is filled from the first ArrayList with the names, and to know where to send the emoji, the corresponding phone number of the selected contact will be found by using the same index from the name ArrayList and the same index from the phone number ArrayList. For the search box, other ArrayLists are used; the only difference is that for some reason, because of the autocomplete function, an index association between the two ArrayLists (name and phone number) was not possible. To address this, a trick was performed. The name found in the autocomplete popup is written normally and the phone number is written with a transparent colored tint on the autocomplete popup. The user cannot see it, but the software can copy the phone number and in this way will know which user to send the emoji via WhatsApp.

3.3. Standalone Fast Emoji Sender

The basic idea of this app is similar to the previous one, the emoji sender through WhatsApp, the difference being that it sends all of the emojis and performs user profile creation with its own database, rather than using a third-party app for the database. This app is more complex than it seems and has more Activities than we could imagine at first glance. Of course, most Activities access *.php files to execute SQL queries on the server. Data communication with the *.php files is performed in a JSON-encoded format. The Activities are written in the Kotlin programming language, since this has been the official programming language for Android apps since 2019. The Activities (*.kt files) are the following: AddFriend, BlockFriends, ChangeName, ChangePass, Dashboard, EditProfile, Main, Friends, Login, Singleton, NotificationEventReceiver, NotficationIntentService, NotficationServiceStaterReceiver, Register, SessionHandler and User. The Layouts (*.xml files) are the following: addfriend, blockfriends, changename, changepass, dashboard, editprofile, main, activity_friends, activity_login, activity_register and emojiview. The *.php files to execute SQL queries are the following: add_friend, block_user, blocked_friends, change_full_name, change_pass,db_connect,delete _friend, delete_profile, delete_user, friends, functions, login, read, register, send and unblock_user.

The AddFriend Activity has the matching addfriend Layout and add _friend *.php file for server communication. It is used to add a friend via the app. The Activity sends data in a JSON-encoded format and the *.php file executes the SQL query to add to the initiating username, the username of the friend and the status "1", meaning they are friends. The BlockFriends Activity has the matching blockfriends Layout, block _user, blocked_friends, delete_user and unblock_user *.php files for server communication. The Activity files, as seen, communicate with four *.php files, and blocked_friends is used to read the list of blocked friends, friends with status "2". This is shown in a ListView in the BlockFriends Activity. When tapping on a friend, the user can chose to Block/Unblock or Delete their friend. The Block will be performed by the block_user *.php file, by writing the status "2" in the list of friends. The Unblock will be performed by the unblock_user *.php file, by writing the status "1" in the list of friends. The Delete will be performed by the delete_user *.php file, by deleting all of the row with the initiating user, friend and status for the friends table. The ChangeName Activity has the matching changename Layout and change_full_name *.php file for server communication. The Activity is used to update the full_name field in the database entry of the initiating user. The ChangePass Activity has the matching

changepass Layout and change _pass *.php file for server communication. The Activity is used to update the password field in the database entry of the initiating user. The operation is a little more complex, because the password needs to be encrypted as a hash using a salt key.

The Dashboard Activity has the matching dashboard Layout but no *.php files, while this Activity does not execute any command on the server side. The Dashboard can be seen after user login and it looks like the screen capture in Figure 10. On the Dashboard, the name of the user can be seen. The user also has the possibility to invite another user to install the app via an Intent by sending the link of the app via a messenger app (SMS and WhatsApp). The app can be used if both users have the app installed, not like the previous app (emoji sender via WhatsApp), where only the sender would need to have the app installed and the recipient needs to have only WhatsApp installed. The user can add a friend by accessing the AddFriend Activity; the user can also see the friends list by accessing the Friends Activity. The friends list can be seen in Figure 11. The user can edit the profile by accessing the EditProfile Activity. The user can see the blocked friends list by accessing the BlockFriends Activity and, of course, the user can logout by ending the session and accessing the Main Activity. The EditProfile Activity has the matching editprofile Layout and delete _profile *.php file for server communication. The Activity has more buttons to access different Activities; it can access the ChangeName and ChangePass Activities and can delete the profile by accessing the delete_profile *.php file. This last action deletes everything from the server: the username, password, friend list and conversations (sent emojis). The Main Activity has the matching main Layout. It has two buttons on the initial screen when installing the app; the buttons access two Activities, Register and Login. The Friends Activity has the matching activity_friends Layout, delete _friend, friends and send *.php files for server communication.

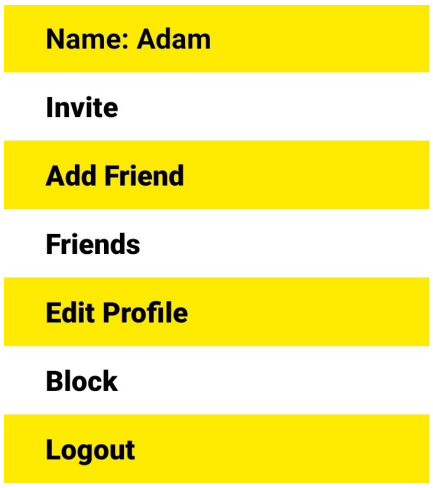

Figure 10. Dashboard after login.

Actually, the Activity loads the user's friend list as shown in Figure 11 by using the friends *.php file, which reads the user's friends from the database.

The user can tap on a name and an emoji pop-up will appear as shown in Figure 12, which is made with the emojiview Layout. When the user taps again on an emoji, the emoji will be sent to the selected friend using the send *.php file, which stores the specific emoji in the database. The database will have a messages table, which has three columns: from, to, and message. The emoji is written in the message column. The delete_friend *.php file is executed by long tapping on a friend and tapping on the Delete button in the AlertDialog popup. The Friends Activity has some circular floating action buttons for sharing the app

(Invite Intent), adding friends (AddFriend Activity) or going back to the Dashboard. These buttons are added to this Activity due to the fact that the app stays logged in on the user's phone, and this is the Activity that is accessed when the user resumes the app.

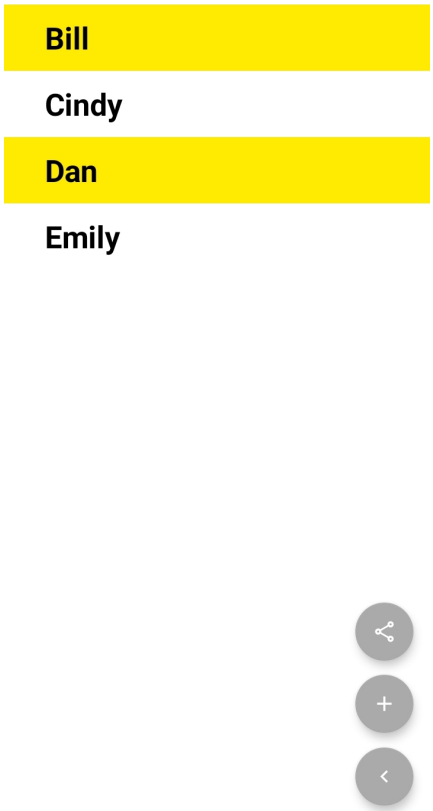

Figure 11. Loading the list of friends.

The Login Activity has the matching activity_login Layout and login *.php file for server communication. The Activity has two input EditText boxes, which can be used to enter the username and password, and a Login button to execute the login. The password is encrypted with hash using a salt key. Actually, the login *.php file checks the user's existence in the database and the correctness of the password; if everything is OK, the user will be logged in by accessing the Dashboard Activity as shown in Figure 10. The Register Activity has the matching activity_register Layout and register *.php file for server communication. The *.php file on the register checks if the user is already registered or not. In case they are not registered, the user's full name and password are written in the database. Of course, the password is encrypted with a hash using a salt key. The activity_register Layout has three EditText boxes (full name, username, and password) and a Register button.

There are some Activities that do not require a Layout file, being executed in the background or just being used for session handling or for global variables (user). The Singleton and SessionHandler are used to maintain the session to keep the user logged in by using private SharedPreferences. NotificationEventReceiver, NotficationIntentService and NotficationServiceStaterReceiver Activities are used to receive messages from a friend as notifications in the notification area, as shown in Figure 13.

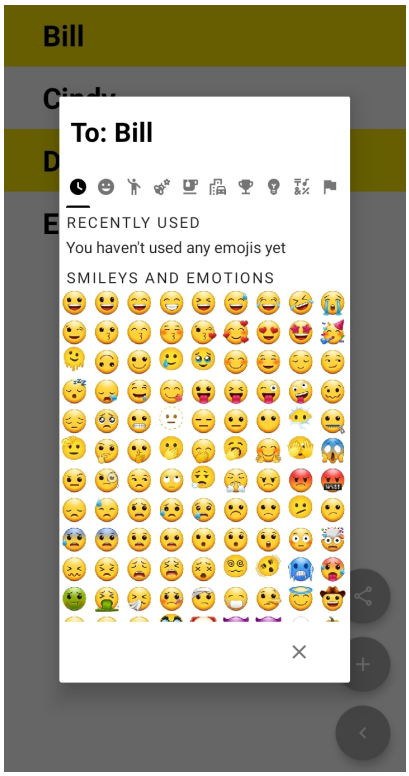

Figure 12. Selecting an emoji to send from the popup window.

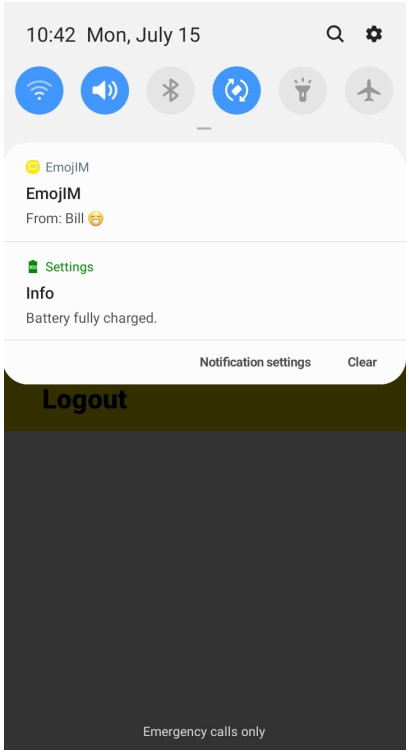

Figure 13. Receiving an emoji in the notification bar.

The NotficationIntentService Activity uses the read *.php file to periodically read the message shown on the message table of the database. The NotficationServiceStaterReceiver Activity starts the notification even after the phone restarts, and NotificationEventReceiver sets the notification period.

There are two more *.php files. One is db_connect, which has the credentials to connect to the database; it is safe to have this on a separate *.php file. The other is the *.php file,

which contains different functions used in different database operations. It has the functions to check if the user exists, friend exists, and it was a successful login. The *.php file also has the function to delete the contents of the message table once a day; because the app does not have a message history, the messages (the emojis) disappear when read, like Snapchat. The *.php file also has the salt function for creating password hashes.

All the *.php files are written carefully (with connection prepare, bind_param, execute, bind_result and result fetch) to prevent SQL injection attacks.

As it was presented, even though the user interface for a chat application appears to be simple, there is a lot of coding in the background. For this app, 43 files were used to create a chat application without character typing, which only sends emojis and does not store the received messages (emojis). This is why sometimes the emoji sender through WhatsApp can make more sense, as it is much less complicated to code because the database and the sending part are performed by a third-party app (WhatsApp). The personally owned database is an extra cost and a lot of coding effort, not to mention taking care of security breaches and database hacks.

3.4. Differential Equations for Databases

The differential equation that governs the rate of growth of the database (Figure 14) will be presented.

$$\frac{dN(t)}{dt} = rN(t) - c \tag{11}$$

where:

- $N(t)$ denotes the quantity of database records (such as messages or users) at a given moment t;
- r is the pace of increase (e.g., the number of new users joining over a specified time period);
- c represents a constant that denotes churn (e.g., users discontinuing the service).

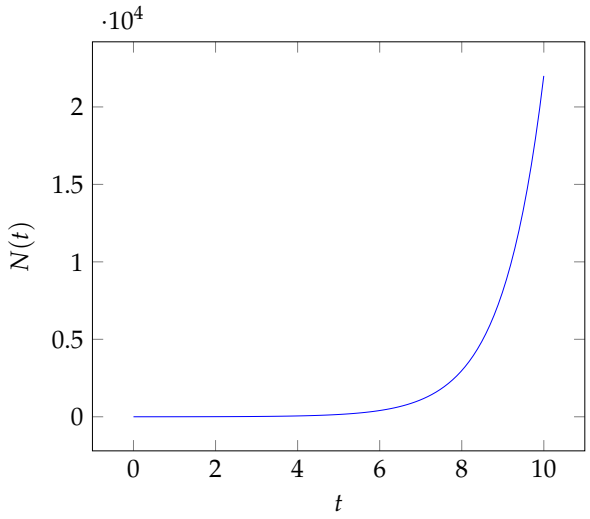

Figure 14. Growth of the database over time.

The increase in the number of messages within a chat database can be described by the following differential equation:

$$\frac{dM(t)}{dt} = \alpha M(t) - \beta \tag{12}$$

where:

- $M(t)$ indicates the total number of messages over time t;
- α represents the rate at which new messages are generated;

- β is the frequency of message removal or turnover.

Figure 15 shows the increase in messages over time.

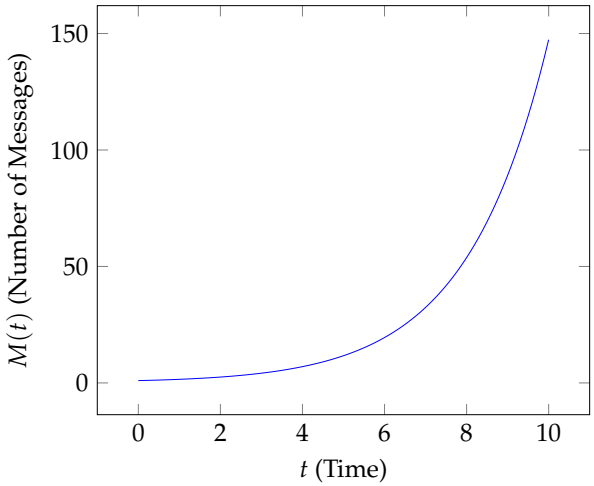

Figure 15. Message growth over time.

User growth in a chat application can be represented by the following logistic growth equation:

$$\frac{dU(t)}{dt} = rU(t)\left(1 - \frac{U(t)}{K}\right) \tag{13}$$

where:

- $U(t)$ represents the count of users at a given time t;
- r represents the inherent growth rate;
- K refers to the system's maximum capacity.

Figure 16 illustrates the logistic growth of users over a period of time.

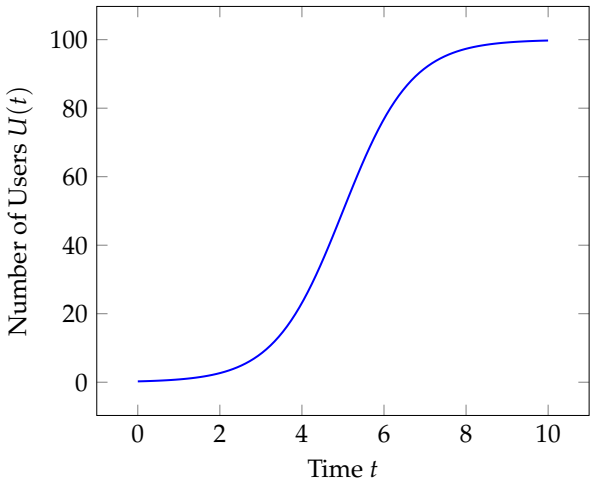

Figure 16. User growth over time (logistic growth).

The frequency of message exchanges (new messages within a given timeframe) in the chat application can be represented as follows:

$$\frac{dM(t)}{dt} = \alpha U(t) - \beta M(t) \tag{14}$$

where:

- $M(t)$ represents the count of messages at a given time t;

- α is the rate of message generation proportional to the number of users;
- β is the degradation of the message (e.g., removal of outdated messages or dormancy of inactive users).

Figure 17 illustrates the traffic of messages over time, considering the number of active users.

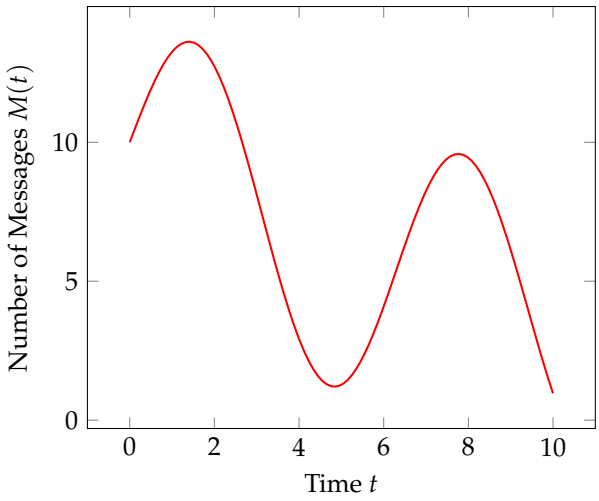

Figure 17. Message traffic over time.

The server load can be represented by the following:

$$\frac{dL(t)}{dt} = \gamma M(t) - \delta L(t) \tag{15}$$

The server load, determined by message traffic and data retention, is illustrated in Figure 18.

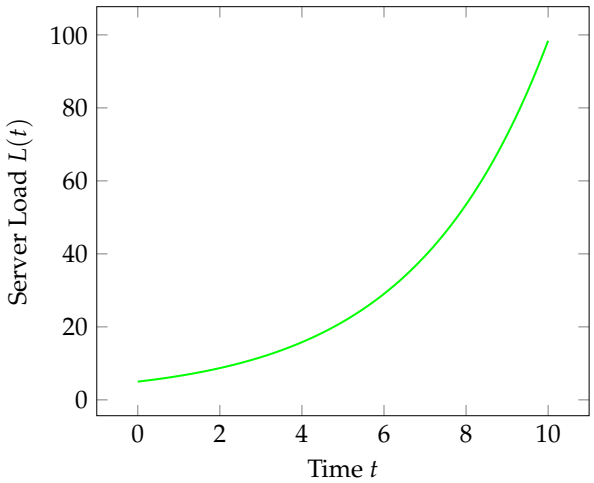

Figure 18. Server load over time.

Where:

- $L(t)$ is the load on the server at that moment t;
- γ is the speed at which messages add to the load;
- δ is the speed at which the system processes or purges.

All three graphs overlaid are shown in Figure 19.

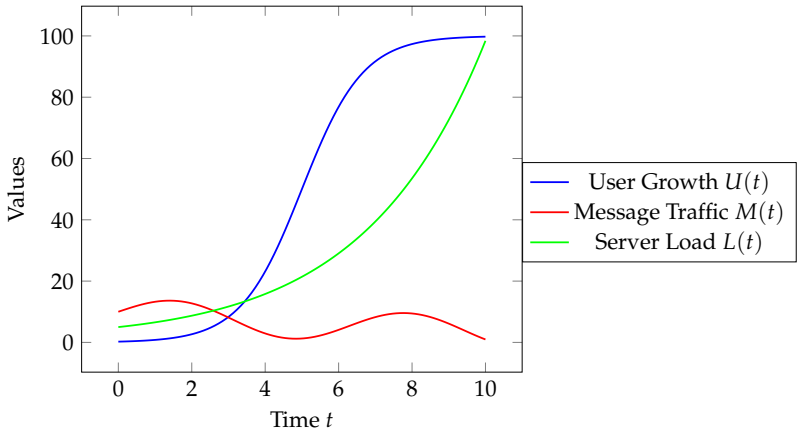

Figure 19. Dynamics of user growth, message traffic and server load.

The previous graph illustrates the relationship between user growth, message traffic and server load.

3.5. Statistical Analysis of Databases Used in Mobile Chat Applications

The pattern of messages sent in a 24-hour period can be influenced by user activity. The histogram in Figure 20 illustrates the frequency of messages per hour.

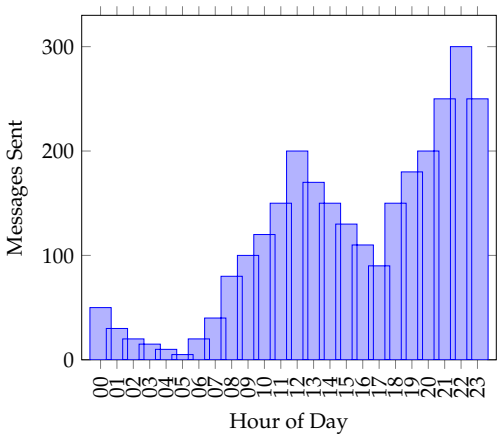

Figure 20. Message frequency distribution.

Figure 21 illustrates the distribution of active users in various time zones. The accompanying pie chart provides a visual representation of this statistical information.

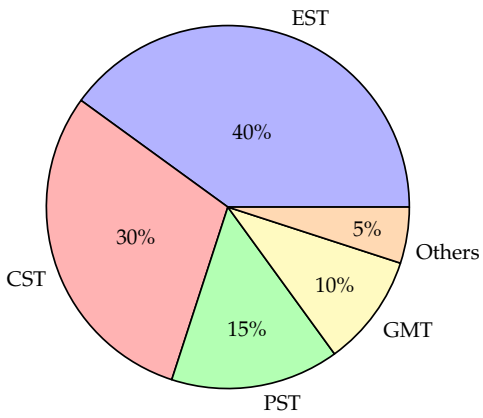

Figure 21. Distribution of active users in different time zones.

The line graph in Figure 22 illustrates the progression of storage usage within the chat application database over time.

The box plot in Figure 23 illustrates the response times for various user queries measured in milliseconds. It is useful to highlight the variability and median response times.

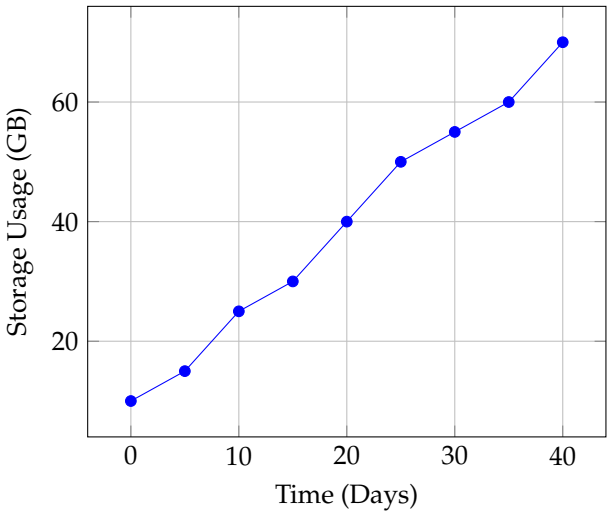

Figure 22. Storage usage over time.

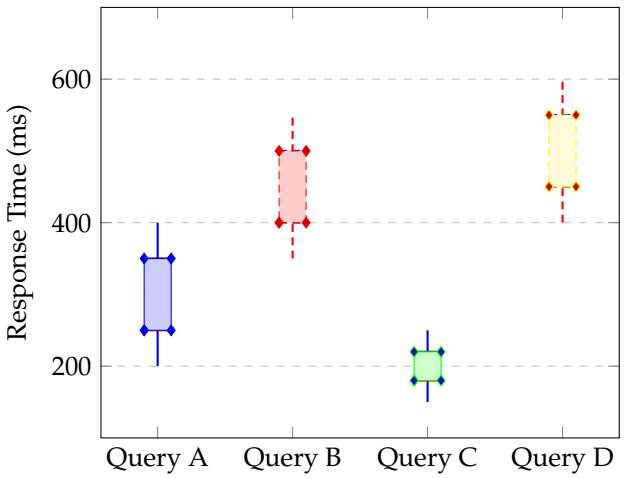

Figure 23. Response time analysis.

4. Discussions

4.1. Fast Emoji Sender for WhatsApp

The fast emoji sender was created to create a mobile application without typing that can send an emoji to a friend via WhatsApp. The idea was to tell a friend that you are thinking about them with an emoji. The app works with two taps, one tap on the contact (friend) and the second tap on the emoji. After this, the selected emoji is sent to the selected contact (friend) through WhatsApp.

The app loads all contacts from the phone, so no contact (friend) needs to be added. The app has another advantage that only the sender needs to have the app installed, and the recipient needs to have only WhatsApp installed.

The disadvantage of the app is that it is not a standalone application and depends on WhatsApp being installed on the mobile device. It relies on WhatsApp services to send emojis, not its own server with a database. This disadvantage is minimal due to the fact that most users have WhatsApp installed because it is one of the most widely used messaging platforms, and if it is not installed, the application can ask the user to install WhatsApp.

The other fact is that WhatsApp servers can sometimes be more reliable than a separate server with a database.

Keeping the app as simple as possible by using WhatsApp servers can make the app load faster, more responsive, and of course have a lower chance of crashing. This is because the shorter the code, the less chance there is of having bugs that could cause the app to crash.

The challenge of this app was integrating it with WhatsApp and also making it easier to send emojis, so it is easier to use this app than WhatsApp itself. The app can send emojis just by tapping on a contact and then tapping again on a selected emoji. It is a communication app that sends emojis with two taps.

4.2. Standalone Fast Emoji Sender

The standalone fast emoji sender is similar to the app presented before, but uses its own server with a database.

Having its own server with a database can offer the advantage that the app will not rely on other databases to send the message (emoji); this can be a big advantage if the app owner has a really reliable server subscription. Another advantage is that the user needs to have only this app installed, not other third-party apps such as WhatsApp, which could be used to send the emojis.

This app can also have some disadvantages. One is that if the server with the database is not reliable, then it is better to use a third-party app like WhatsApp and use their server to send the emojis. Another disadvantage could be that both the sender and recipient need to have the app installed and have an account on the app to send emojis to each other. In the previous app, only the sender needed to have the app installed. Maybe the biggest disadvantage could be that the app became really complex with 43 coded files in Kotlin, XML and PHP. This can increase the probability of a bug appearing and the app crashing. From my coding experience, the shorter and the simpler the code, the less chance there is that an app throws an exception and crashes.

The challenge of this app is that it uses a personal cloud database. Although the app is very simple, the complexity is quite high. In the SQL database, there were three tables: one with the user list, one with the friends list, and one with the sent messages. All three databases were databases that were connected to each other in order for the messaging app to work.

5. Conclusions

5.1. Fast Emoji Sender for WhatsApp

The fast emoji sender was created with the idea of creating a no-character-typing messaging application.

Further enhancements can be made, such as sending predefined text or the location of the user. Functions like storing incoming emojis and sending multiple emojis or stickers at a time can be added. The app can also be extended to be able to send emojis not only through WhatsApp but also through other messaging apps or even via SMS. Via SMS, it will be a little bit complicated due to the fact that lately using SMS permissions in the Manifest file is only allowed for apps on the Google Play Store with a really good explanation. The app can also be ported to the iPhone. In addition, another enhancement can be performed by porting the app to smartwatches and also by adding voice commands to further ease the communication between two devices.

5.2. Standalone Fast Emoji Sender

The standalone emoji sender was made to send emojis fast, without using third-party apps but using its own server with a database.

The app can be improved by adding additional features, such as location sharing, sending predefined messages and sending multiple emojis or even stickers. The app can be recoded by not using an SQL database but using a NoSQL database, like Firebase. The app can also be ported to the iPhone to cover a larger user base. The app can be enhanced to control the voice or port the messaging app to smartwatches.

Funding: This research was funded by Politehnica University Timisoara, Romania.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All the images are available on request from the author.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Yan, M.; Chan, C.A.; Li, W.; I, C.-L.; Bian, S.; Gygax, A.F.; Leckie, C.; Hinton, K.; Wong, E.; Nirmalathas, A. Network Energy Consumption Assessment of Conventional Mobile Services, Over-the-Top Instant Messaging Applications. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3168–3180. [CrossRef]
2. Rincón-Nigro, M.; Deng, Z. A Text-Driven Conversational Avatar Interface for Instant Messaging on Mobile Devices. *IEEE Trans. Hum.-Mach. Syst.* **2013**, *43*, 328–332. [CrossRef]
3. Sahafizadeh, E.; Tork Ladani, B. A Model for Social Communication Network in Mobile Instant Messaging Systems. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 68–83. [CrossRef]
4. Meng, L.-S.; Shiu, D.-S.; Yeh, P.-C.; Chen, K.-C.; Lo, H.-Y. Low Power Consumption Solutions for Mobile Instant Messaging. *IEEE Trans. Mob. Comput.* **2012**, *11*, 896–904. [CrossRef]
5. Fu, Y.; Xiong, H.; Lu, X.; Yang, J.; Chen, C. Service Usage Classification with Encrypted Internet Traffic in Mobile Messaging Apps. *IEEE Trans. Mob. Comput.* **2016**, *15*, 2851–2864. [CrossRef]
6. Wu, C.-J.; Ho, J.-M.; Chen, M.-S. A Scalable Server Architecture for Mobile Presence Services in Social Network Applications. *IEEE Trans. Mob. Comput.* **2013**, *12*, 386–398.
7. Wu, J.-Y.; Wu, K.; Wang, M. Power-Constrained Quality Optimization for Mobile Video Chatting With Coding-Transmission Adaptation. *IEEE Trans. Mob. Comput.* **2021**, *20*, 2862–2876. [CrossRef]
8. Radio, N.; Zhang, Y.; Tatipamula, M.; Madisetti, V.K. Next-Generation Applications on Cellular Networks: Trends, Challenges, and Solutions. *Proc. IEEE* **2012**, *100*, 841–854. [CrossRef]
9. Wu, Y.; Shum, K.W.; Wong, W.S.; Shen, L. Safety-Message Broadcast in Vehicular Ad Hoc Networks Based on Protocol Sequences. *IEEE Trans. Veh. Technol.* **2014**, *63*, 1467–1479. [CrossRef]
10. Fogue, M.; Martinez, F.J.; Garrido, P.; Fiore, M.; Chiasserini, C.-F.; Casetti, C.; Cano, J.-C.; Calafate, C.T.; Manzoni, P. Securing Warning Message Dissemination in VANETs Using Cooperative Neighbor Position Verification. *IEEE Trans. Veh. Technol.* **2015**, *64*, 2538–2550. [CrossRef]
11. Wang, E.K.; Li, Y.; Ye, Y.; Yiu, S.M.; Hui, L.C.K. A Dynamic Trust Framework for Opportunistic Mobile Social Networks. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 319–329. [CrossRef]
12. de Vargas, M.F.; Marino, D.; Weill–Duflos, A.; Cooperstock, J.R. Speaking Haptically: From Phonemes to Phrases With a Mobile Haptic Communication System. *IEEE Trans. Haptics* **2021**, *14*, 479–490. [CrossRef]
13. Johansen, C.; Mujaj, A.; Arshad, H.; Noll, J. The Snowden Phone: A Comparative Survey of Secure Instant Messaging Mobile Applications. *arXiv* **2019**, arXiv:1807.07952. [CrossRef]
14. Bahramali, A.; Soltani, R.; Houmansadr, A.; Goeckel, D.; Towsley, D. Practical Traffic Analysis Attacks on Secure Messaging Applications. In Proceedings of the Network and Distributed Systems Security (NDSS), San Diego, CA, USA, 23–26 February 2020; pp. 1–18.

15. Singh, R.; Singh Chauhan, A.N.; Tewari, H. Blockchain-Enabled End-to-End Encryption for Instant Messaging Applications. *arXiv* **2021**, arXiv:2104.08494.
16. Fang, Q.; Zhou, Z.; Barbieri, F.; Liu, Y.; Neves, L.; Nguyen, D.; Oberski, D.L.; Bos, M.W.; Dotsch, R. General-Purpose User Modeling with Behavioral Logs: A Snapchat Case Study. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, Washington, DC, USA, 14–18 July 2024; pp. 2431–2436.