Hochschule Fulda
University of Applied Sciences

Distributed Applications - AI5109
Deleting and Editing Messages

(Module 4)

04.Feb.2024

Table of Contents

Abstract

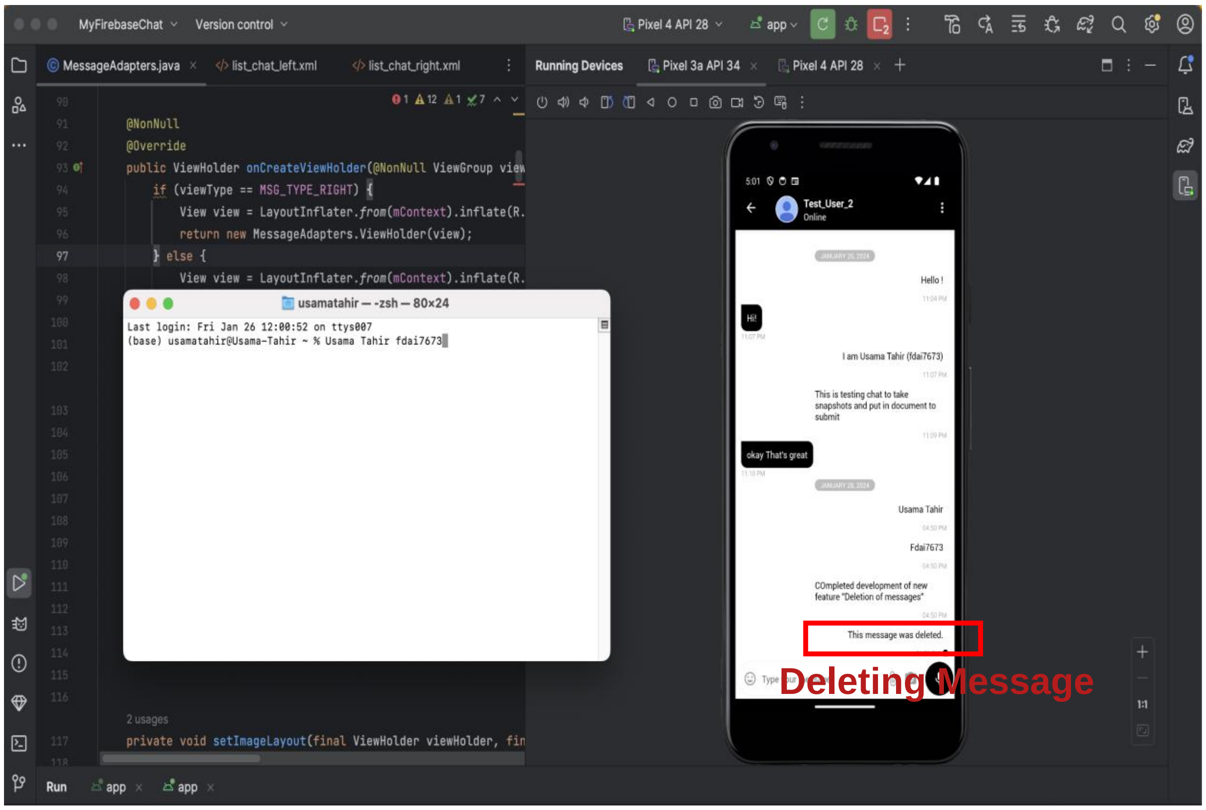
The "What's Chat" Android application is a modern and feature-rich chat platform developed using Android Studio. This cutting-edge chat application leverages the robust capabilities of Google Firebase as its backend infrastructure, tapping into key features such as authentication and Firestore for seamless real-time data management.

Key Features:

- **Authentication:** What's Chat ensures secure user registration and login through the implementation of Firebase Authentication. Users can confidently create accounts and access the app with ease, offering a foundation for a secure and personalized chat experience.
- **Firestore Integration:** The application harnesses the power of Firebase Firestore to facilitate real-time messaging. Firestore's NoSQL database allows for efficient and scalable storage, retrieval, and synchronization of chat data, ensuring a smooth and responsive user experience.
- **Push Notifications:** Integration with Firebase Cloud Messaging (FCM) ensures that users stay connected even when the app is not actively in use. Push notifications keep users informed about new messages, ensuring timely responses and enhancing overall engagement.
- **Media Sharing:** What's Chat goes beyond simple text messaging by allowing users to share images and other media files. This feature adds a multimedia dimension to conversations, enriching the user experience and facilitating more dynamic communication.

The "What's Chat" Android application stands as a testament to the seamless integration of cutting-edge technologies, providing a user-friendly, secure, and feature-complete chat experience. By combining the power of Android Studio and Google Firebase, the application sets a new standard for modern chat applications, offering a platform where users can connect, communicate, and share in a dynamic and engaging environment.

Snapshots of Application



```
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
        if (viewType == MSG_TYPE_RIGHT) {
            View view = LayoutInflater.from(mContext).inflate(R.
            return new MessageAdapters.ViewHolder(view);
        } else {
            View view = LayoutInflater.from(mContext).inflate(R.

    2 usages
    private void setImageLayout(final ViewHolder viewHolder, fin
```

usamatahir — -zsh — 80×24

```
Last login: Fri Jan 26 12:00:52 on ttys007
(base) usamatahir@Usama-Tahir ~ % Usama Tahir fdai7673
```

5:01
Test_User_2
Online

JANUARY 25, 2024

Hello !
11:04 PM

Hi!
11:07 PM

I am Usama Tahir (fdai7673)
11:07 PM

This is testing chat to take snapshots and put in document to submit
11:09 PM

okay That's great
11:10 PM

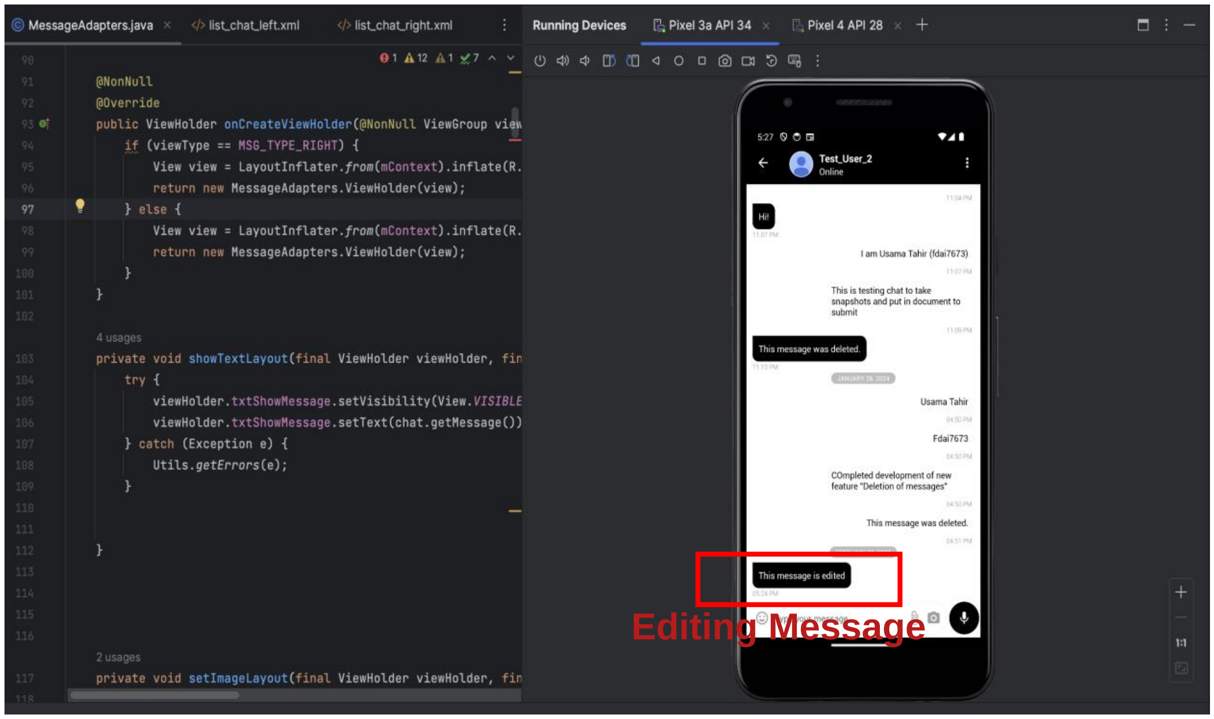
JANUARY 28, 2024

Usama Tahir
04:50 PM

Fdai7673
04:50 PM

COmpleted development of new feature "Deletion of messages"
04:50 PM

This message was deleted.



```
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
        if (viewType == MSG_TYPE_RIGHT) {
            View view = LayoutInflater.from(mContext).inflate(R.
            return new MessageAdapters.ViewHolder(view);
        } else {
            View view = LayoutInflater.from(mContext).inflate(R.
            return new MessageAdapters.ViewHolder(view);
        }
    }

    4 usages
    private void showTextLayout(final ViewHolder viewHolder, fin
        try {
            viewHolder.txtShowMessage.setVisibility(View.VISIBLE
            viewHolder.txtShowMessage.setText(chat.getMessage())
        } catch (Exception e) {
            Utils.getErrors(e);
        }


    }




    2 usages
    private void setImageLayout(final ViewHolder viewHolder, fin
```

5:27
Test_User_2
Online

11:04 PM

Hi!
11:07 PM

I am Usama Tahir (fdai7673)
11:07 PM

This is testing chat to take snapshots and put in document to submit
11:09 PM

This message was deleted.
11:10 PM

JANUARY 28, 2024

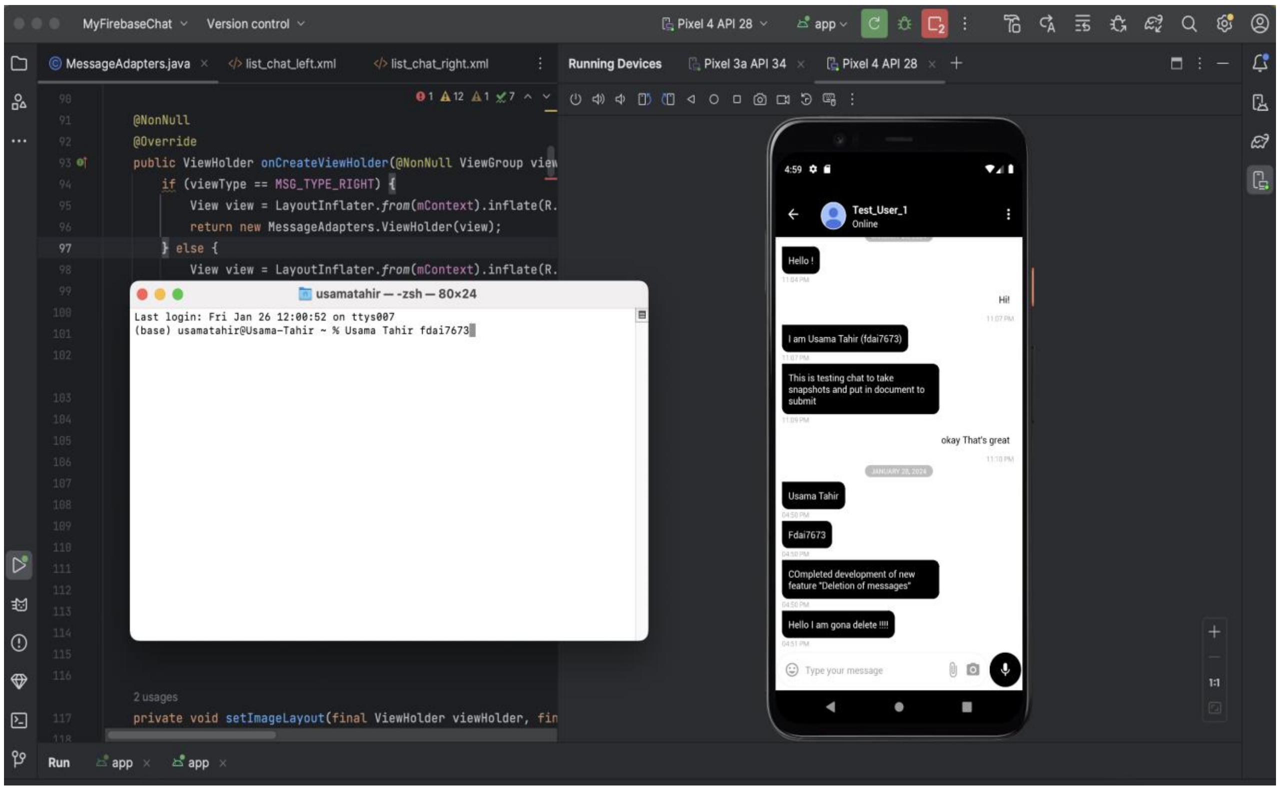
Usama Tahir
04:50 PM

Fdai7673
04:50 PM

COmpleted development of new feature "Deletion of messages"
04:50 PM

This message was deleted.
04:51 PM

This message is edited
05:24 PM



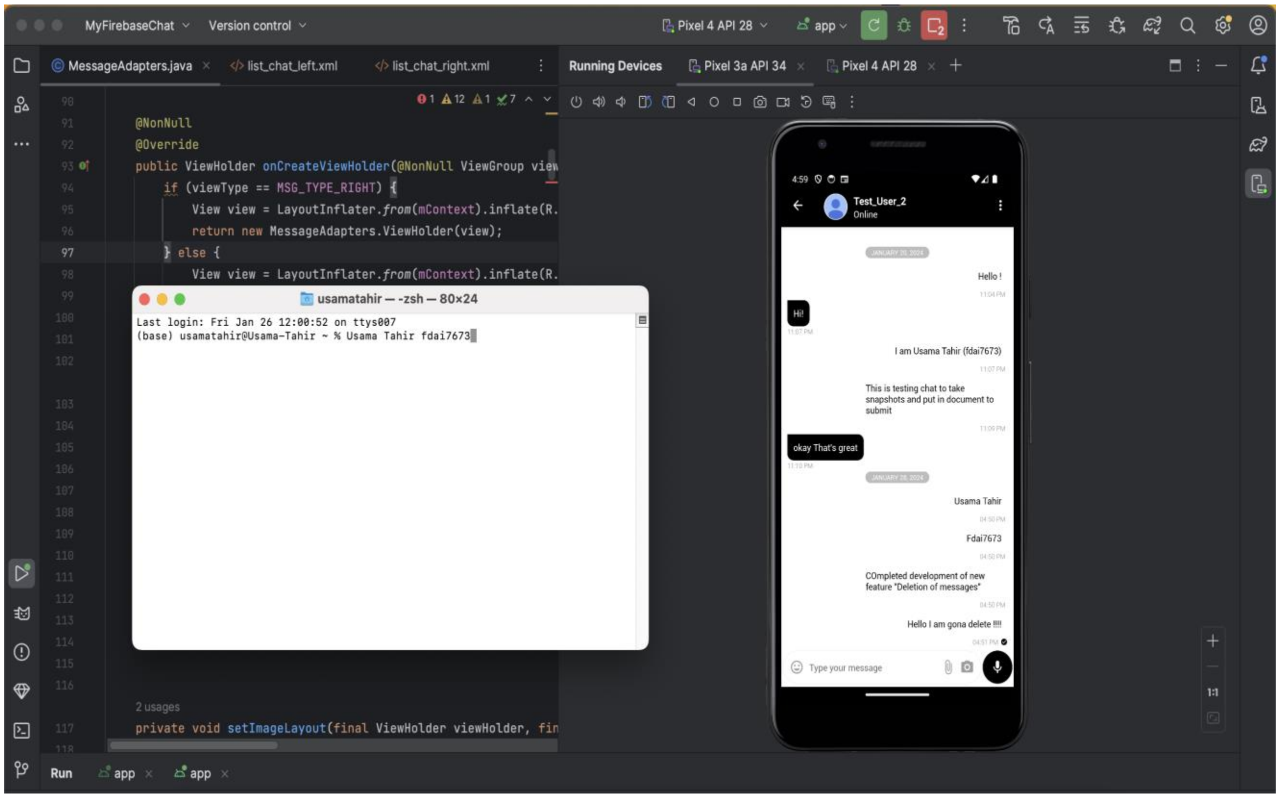
```
@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
    if (viewType == MSG_TYPE_RIGHT) {
        View view = LayoutInflater.from(mContext).inflate(R.
        return new MessageAdapters.ViewHolder(view);
    } else {
        View view = LayoutInflater.from(mContext).inflate(R.

2 usages
private void setImageLayout(final ViewHolder viewHolder, fin
```

```
Last login: Fri Jan 26 12:00:52 on ttys007
(base) usamatahir@Usama-Tahir ~ % Usama Tahir fdai7673
```

Test_User_1
Online

Hello !
11:04 PM
Hi!
11:07 PM
I am Usama Tahir (fdai7673)
11:07 PM
This is testing chat to take snapshots and put in document to submit
11:09 PM
okay That's great
11:10 PM
JANUARY 28, 2024
Usama Tahir
04:50 PM
Fdai7673
04:50 PM
COmpleted development of new feature "Deletion of messages"
04:50 PM
Hello I am gona delete !!!!
04:51 PM



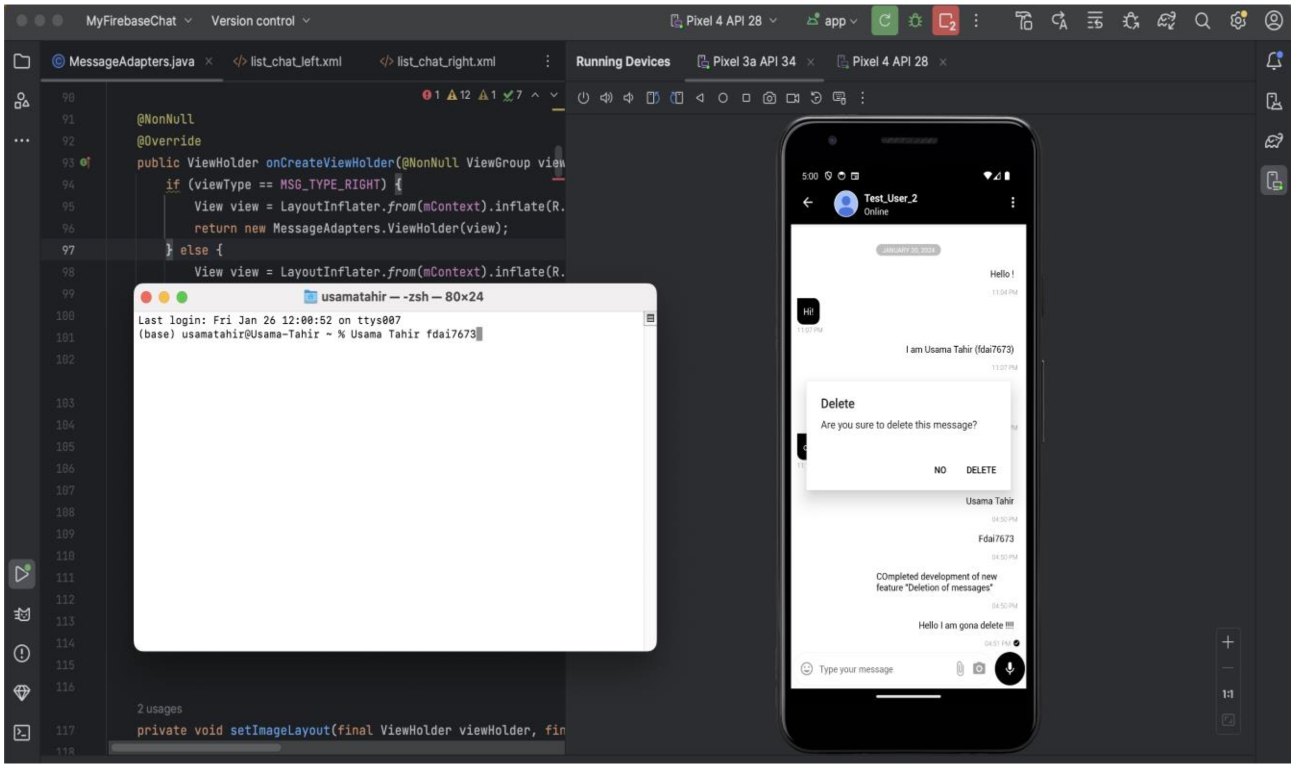
```
@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
    if (viewType == MSG_TYPE_RIGHT) {
        View view = LayoutInflater.from(mContext).inflate(R.
        return new MessageAdapters.ViewHolder(view);
    } else {
        View view = LayoutInflater.from(mContext).inflate(R.

2 usages
private void setImageLayout(final ViewHolder viewHolder, fin
```

```
Last login: Fri Jan 26 12:00:52 on ttys007
(base) usamatahir@Usama-Tahir ~ % Usama Tahir fdai7673
```

Test_User_2
Online

JANUARY 20, 2024
Hello !
11:04 PM
Hi!
11:07 PM
I am Usama Tahir (fdai7673)
11:07 PM
This is testing chat to take snapshots and put in document to submit
11:09 PM
okay That's great
11:10 PM
JANUARY 28, 2024
Usama Tahir
04:50 PM
Fdai7673
04:50 PM
COmpleted development of new feature "Deletion of messages"
04:50 PM
Hello I am gona delete !!!!
04:51 PM



```java
@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
    if (viewType == MSG_TYPE_RIGHT) {
        View view = LayoutInflater.from(mContext).inflate(R.
        return new MessageAdapters.ViewHolder(view);
    } else {
        View view = LayoutInflater.from(mContext).inflate(R.
```

```
2 usages
private void setImageLayout(final ViewHolder viewHolder, fin
```

usamatahir — -zsh — 80×24

```
Last login: Fri Jan 26 12:00:52 on ttys007
(base) usamatahir@Usama-Tahir ~ % Usama Tahir fdai7673
```

5:00

Test_User_2
Online

Hello !
11:04 PM

Hi!
11:07 PM

I am Usama Tahir (fdai7673)
11:07 PM

Delete
Are you sure to delete this message?
NO DELETE

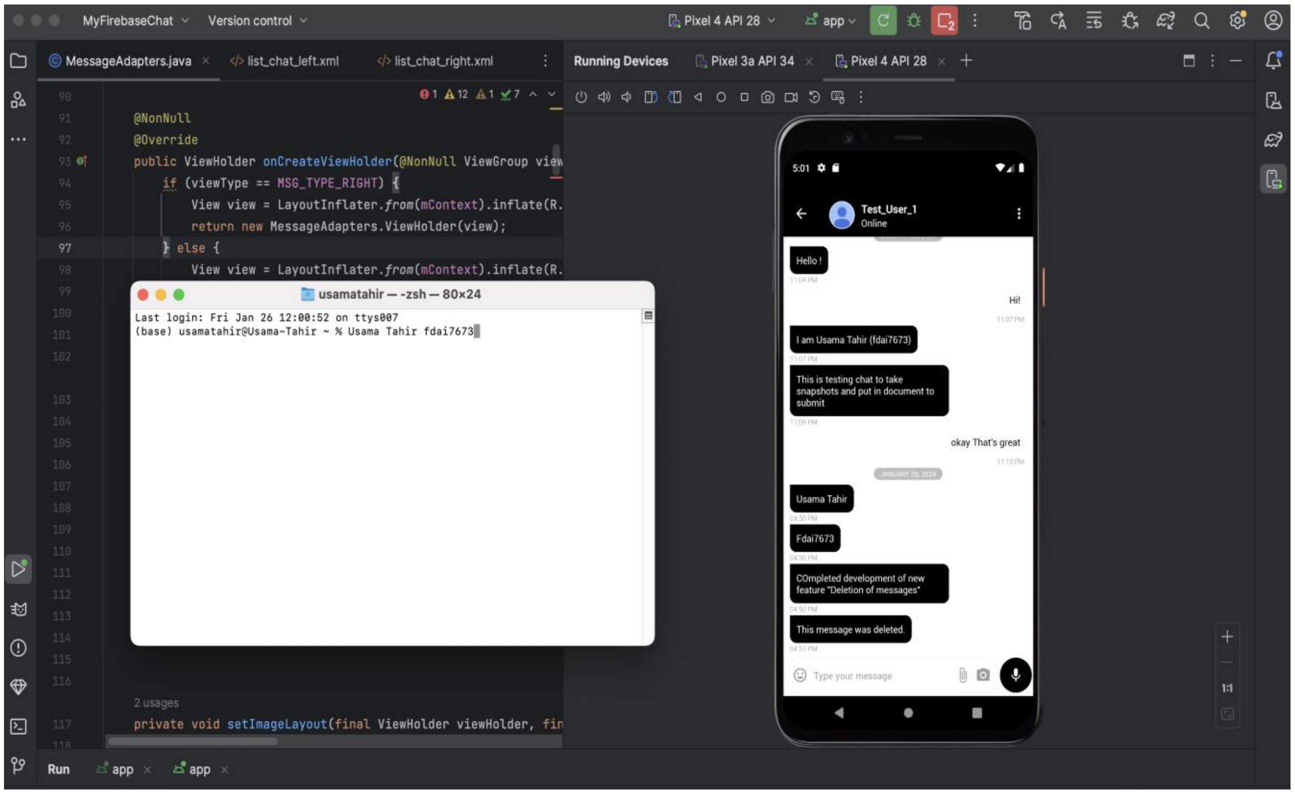
Usama Tahir
04:50 PM

Fdai7673
04:50 PM

COmpleted development of new feature "Deletion of messages"
04:50 PM

Hello I am gona delete !!!!
04:51 PM

Type your message



```java
@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup view
    if (viewType == MSG_TYPE_RIGHT) {
        View view = LayoutInflater.from(mContext).inflate(R.
        return new MessageAdapters.ViewHolder(view);
    } else {
        View view = LayoutInflater.from(mContext).inflate(R.
```

```
2 usages
private void setImageLayout(final ViewHolder viewHolder, fin
```

usamatahir — -zsh — 80×24

```
Last login: Fri Jan 26 12:00:52 on ttys007
(base) usamatahir@Usama-Tahir ~ % Usama Tahir fdai7673
```

5:01

Test_User_1
Online

Hello !
11:04 PM

Hi!
11:07 PM

I am Usama Tahir (fdai7673)
11:07 PM

This is testing chat to take snapshots and put in document to submit
11:09 PM

okay That's great
11:10 PM

JANUARY 28, 2024

Usama Tahir
04:50 PM

Fdai7673
04:50 PM

COmpleted development of new feature "Deletion of messages"
04:50 PM

This message was deleted.
04:51 PM

Type your message

Answers to the questions:

Question 1 : List and explain the various communication paradigms

Communication paradigms refer to the ways in which communication is structured and carried out between entities in a system. There are several communication paradigms, each with its own characteristics and use cases. Here are some of the common communication paradigms:

Point-to-Point (P2P) Communication:

Point-to-Point (P2P) communication is a straightforward communication paradigm characterized by its direct interaction between precisely two entities, which can be nodes or processes. This model is known for its simplicity and efficiency in establishing a direct link between sender and receiver. In practical terms, P2P communication finds relevance in scenarios such as file transfers between two devices, where a sender directly transmits data to a designated recipient without intermediaries. Similarly, chat applications leverage the simplicity of P2P communication, allowing users to exchange messages directly between one another. The inherent nature of P2P communication makes it an apt choice for applications requiring a direct, uncomplicated interaction between two endpoints.

Publish-Subscribe (Pub-Sub) Communication:

Publish-Subscribe (Pub-Sub) communication is a versatile paradigm defined by its involvement of three essential components: publishers, subscribers, and a central message broker. In this model, publishers transmit messages to the central broker, which then distributes these messages to all subscribed entities. The key feature of Pub-Sub is the decoupling of senders (publishers) and receivers (subscribers), enabling a more scalable and flexible communication architecture. This separation allows publishers to broadcast information without needing knowledge of specific recipients, while subscribers can receive relevant messages without direct interaction with the publishers. Pub-Sub finds practical application in real-time data streaming scenarios, where a continuous flow of information needs to be disseminated efficiently, as well as in event-driven systems, where dynamic responses to events are crucial for system behavior.

Client-Server Communication:

Client-Server communication is characterized by the interaction between two distinct entities: clients and servers. In this paradigm, clients initiate communication by requesting services or resources from servers, which, in turn, respond to these requests. This model is fundamental to various computing applications, including web applications that heavily rely on HTTP communication. In the context of web development, clients, typically web browsers, send requests to servers for web pages, data, or other resources. Additionally, client-server communication plays a crucial role in database access scenarios, where clients, such as applications or users, make requests to servers for data retrieval or manipulation through operations like SQL queries. This fundamental communication model forms the backbone of many distributed systems and is essential for enabling seamless interactions in various online services and applications.

Remote Procedure Call (RPC):

Remote Procedure Call (RPC) is a communication paradigm that enables a program to invoke a procedure or subroutine to execute in a different address space, often on a remote machine. One of its key characteristics is the encapsulation of communication details, providing a mechanism that closely resembles calling a local procedure. RPC abstracts the complexities of remote communication, making it more convenient for developers to interact with processes on distant machines as if they were local. This communication model is widely employed in distributed systems, where multiple interconnected components collaborate across networked environments. Additionally, RPC is a foundational concept in client-server applications, facilitating seamless communication between clients and servers while abstracting the intricacies of network interactions. Its use in distributed computing environments contributes to the efficiency and modularity of software systems by allowing procedures to be executed remotely without the need for developers to manage low-level communication protocols.

Message Passing:

Message Passing is a communication paradigm characterized by the exchange of messages between processes, where communication occurs through the sending and receiving of messages. In this model, processes interact by transmitting messages to one another, providing a means for data and instructions to be communicated across a system. This approach is particularly prevalent in parallel computing, where multiple processes run simultaneously and need to coordinate and share information. Message Passing is instrumental in the development of parallel algorithms and applications that require synchronization and data exchange among concurrently executing tasks. Moreover, it plays a critical role in inter-process communication within operating systems, enabling different processes to communicate, share data, and coordinate their activities. The simplicity and modularity of the message passing model contribute to its effectiveness in scenarios where seamless communication between processes is essential for efficient system operation.

Question 2 : Briefly explain the request/reply protocol

The request/reply protocol serves as a fundamental communication paradigm in computing, defining the orchestration of interactions between clients and servers. In this structured model, a client initiates communication by forwarding a request to a server, which, upon receipt, meticulously processes the enclosed information or executes the operation specified by the client. Following this, the server crafts a detailed reply, encapsulating the outcomes or pertinent data, and transmits it back to the awaiting client. The synchronous and sequential nature of this protocol ensures that the client patiently awaits the server's response before progressing further.

This communication pattern finds extensive utility in various computing scenarios. In the realm of web applications, for instance, user-triggered actions, such as form submissions or button clicks, prompt clients to dispatch requests to servers. The servers, in turn, respond with updated content or acknowledgments, allowing for dynamic and responsive user experiences. Additionally, the request/reply protocol plays a pivotal role in Remote Procedure Call (RPC) mechanisms. Here, clients invoke procedures on remote servers, and the protocol governs the structured exchange of requests and corresponding results.

In essence, the request/reply protocol establishes a framework that not only facilitates seamless communication between clients and servers but also ensures an organized and predictable flow of interactions. Its widespread adoption underscores its significance in shaping the architecture of interactive computing across diverse applications and systems.

Question 3 : Explain Multicast, concurrency and parallelism in detail

Multicast, concurrency, and parallelism are core concepts that underpin the efficiency and performance of computer systems. Starting with multicast, this communication paradigm enables a single sender to broadcast data to a predefined group of recipients simultaneously. This approach optimizes the dissemination of information to multiple parties without the need for replicating data for each recipient. Multicast is particularly useful in scenarios where a collective audience is interested in receiving the same information, enhancing communication efficiency.

Concurrency, on the other hand, addresses the ability of a system to handle and progress multiple tasks or processes concurrently. Unlike true simultaneous execution, concurrency allows for overlapping time intervals, allowing independent tasks to make progress concurrently. This concept is vital for managing various activities simultaneously, enhancing overall system responsiveness. Concurrency is often applied in scenarios where tasks are independent, and progress in one task does not significantly impact others.

Parallelism takes the notion of simultaneous execution further, involving the actual execution of multiple tasks at the same time. This concept is crucial for achieving speedup in computational tasks by leveraging multiple processors or cores. Parallelism requires breaking down complex problems into smaller, independent units that can be executed simultaneously, leading to enhanced performance. It is a fundamental aspect of designing systems that handle computationally intensive workloads efficiently.