A REAL TIME CHAT AND COMMUNICATION APP

ABSTRACT:

Chat application is an important tool for today's world as it help's everyone to contact from anywhere and talk with them at real time without any problem and this tool resolve the issue of calling every time for any query related anything about that work, Now with the help of chat application user can just send the message to the other user and resolve the issue or query without calling. The best part about the chat application is that we can send the same message to many users at a same time without any problem as we can make a group of so many people or even we can broadcast the message to selected users to whom the sender wishes to send the message. Chat application also provided us the best feature in this as you can share your current location with anyone at any time and even you can share the images or pdf or any other type of file using the application.

INTRODUCTION:

As the time going on, the development of information and communication technology is growing so faster and playing very important role in the society. As with the help of the information and communication technologies we can facilitate the people to receive the information and communicate with each other from anywhere, anytime and faster than before. If we see the today's scenario after the technology have been arrived most of the people don't use newspaper for getting the information and even not much uses the news channel now a days for getting information, Majority is using the smart technology for receiving any type of information with the help of smart phones or other devices and if we talk about the chat application, in the chat application everyone is sharing many news there itself only about what's going on around them or in any place in the world, as with the help of the chat application information is spreading so faster than before. Currently, the chat application has been developed by many developers and it has its own advantages and disadvantages as every other thing have, and it's giving us more options to decide which thing is better for us and what features are really required in the application and what not, as the review of the users using the application tells a lot about the application and their experience towards the application. The Chat application is fully responsive as it means totally user friendly for every web-based or mobile or any other device users. It is built using

Node.js, Socket.io, MongoDB, Passport.js, Auth2.0 and other JavaScript libraries. Node.js is a software platform that is used to build server side more flexible for applications in a network application. Socket.IO is a JavaScript library and it is an implementation of web sockets protocol and various other required improvisation for real-time chat application. JavaScript is a scripting language that is used to create a program so the HTML document displayed in the browser becomes more interactive and impressive for users. MongoDB is an open source NOSQL database based on Document-Oriented Database program, which was originally created in C ++. Therefore, It's expected that this chat application can run easily on real time.

MAIN PROBLEM:

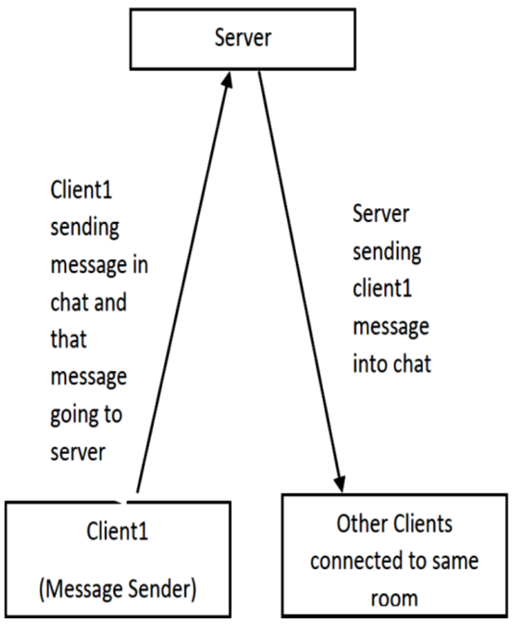
Based on the title above, the problem is how to design and implement responsive web-based or mobile or other devices chat applications in real time to make communication becomes easier and faster.

OBJECTIVE

The objective can be defined from the main problem given above is that to build a proper real-time multiplatform chat application which can be easily used by people to make their life easy to share the information and communicate with each other easily and faster.

A. Block Diagram

Here is a block diagram of server – client communication


Server
Client1 sending message in chat and that message going to server
Server sending client1 message into chat
Client1 (Message Sender)
Other Clients connected to same room

SYSTEM DESIGN

The application is built in two sides one is client and other is server. Client and Server connected bidirectional, when the client sends the message it goes to the server and when the server receives that message, server cannot do anything it could just print on terminal or do something else like dump it file into, but what we want to do is that make sure that everyone is connected to the chat room actually sees the message that this person has sent and at which time. We will bring other clients into mix and this time we send data from server to client. Server will send the client1 message to other clients that client 1 has sent the message, server will send that across to client and client can render that message through browser. In short when any client send the message it goes to server and then after receiving the message server sends it to all the other clients present in the room at that time.

PROJECT METHODOLOGY

In a chat system, clients can be mobile applications or web applications. Clients do not communicate directly. Instead, each client connects to a chat service, which supports all of the features mentioned above. Let's focus on the basic functionality.

The chat service should support the following activities:

1. Get messages from other clients.
2. Find the right recipients of each message and pass the message on to the recipients.
3. If the recipient is offline, hold the recipient's messages on the server until they are online.

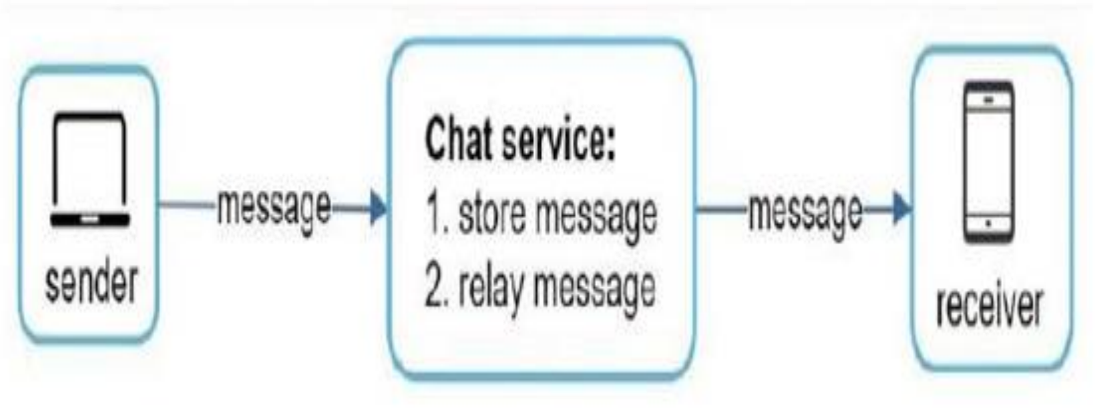
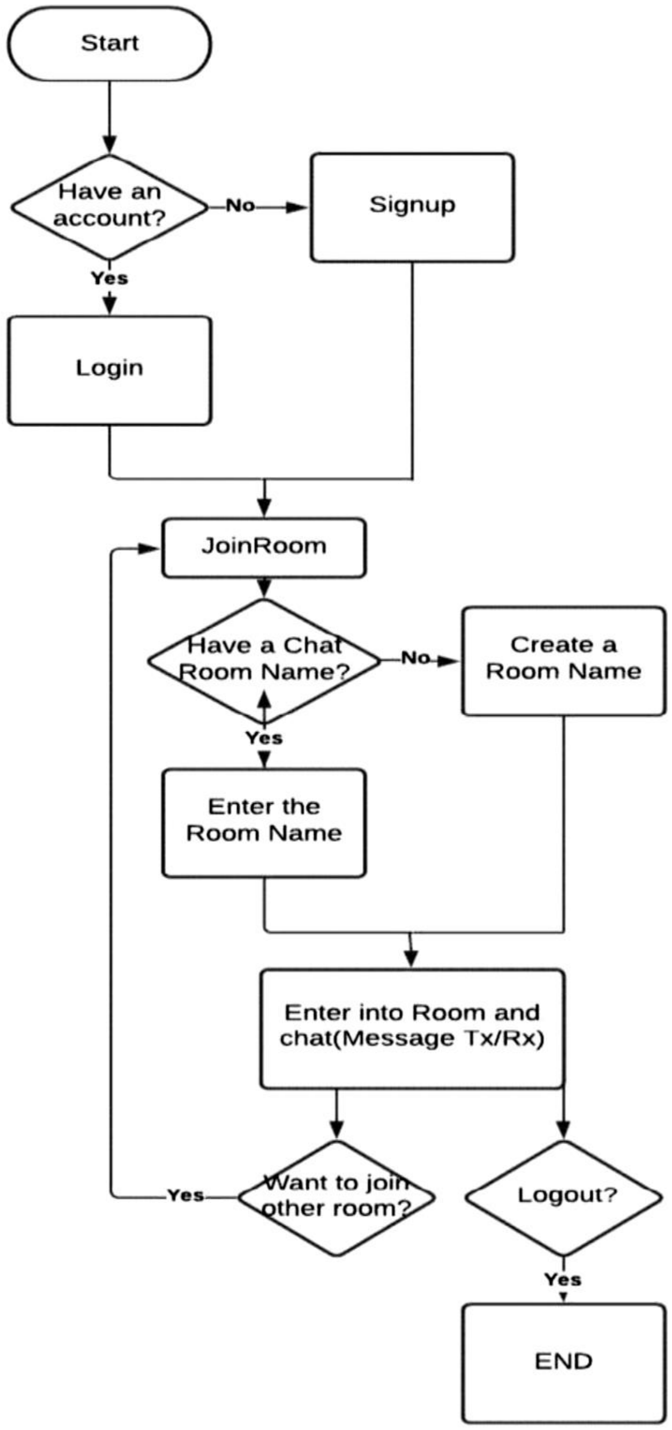


Figure 2. shows the relationships between clients (sender and receiver) and the chat service

Flow Chart

Chat application overall process is shown with the help of flow chart below for more clarity



The explanation for each flow from picture above is as follow:

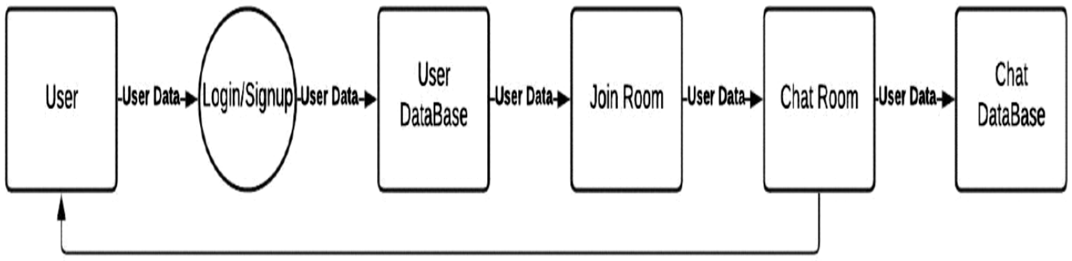
1. When the user enter into the chat application, user login credentials is required. If the user doesn't have an account, then the user should signup first, In both login and signup page user have option to sign up using Google or Facebook account.

2. After login, user have to enter the room name in which user can start a chat. Therefore, if user doesn't have the room name, user can create its own room name and invite their friends to chat with them in that room. After entering the room name user will be redirected to chat page.

3. There are two sides in the chat application in left side sidebar is given where user can see the room name in which they have joined and they can also see the connected online user at that time in chat room. Also, there is an option to join other room and option to logout.

Data Flow Diagram

Here is a data flow diagram to describe overall process of chat application data.



Our SDKs are available from MavenCentral, with some of our dependencies being hosted on Jitpack. Update your repositories in the settings.gradle

- Select the Empty Activity template
- Name the project ChatTutorial
- Set the package name to com.example.chattutorial
- Select your language - **Kotlin** (recommended) or Java
- Set the Minimum SDK to 21 (or higher).

Let's have a quick look at the source code shown above:

- **Step 1**: We create a StreamOfflinePluginFactory to provide offline support. The OfflinePlugin class employs a new caching mechanism powered by side-effects we applied to ChatClient functions.
- **Step 2**: We create a connection to Stream by initializing the ChatClient using an API key. This key points to a tutorial environment, but you can **sign up for a free Chat trial** to get your own later. Next, we add the offlinePluginFactory to the ChatClient with withPlugin method for providing offline storage capabilities. For a production app, we recommend initializing this ChatClient in your Application class.
- **Step 3**: We create a User instance and pass it to the ChatClient's connectUser method, along with a pre-generated user token, in order to authenticate the user. In a real-world application, your authentication backend would generate such a token at login / signup and hand it over to the mobile app. For more information, see the **Tokens & Authentication** page.
- **Step 4**: We configure the ChannelListViewModelFactory with a filter and a sort option. We're using the default sort option which orders the channels by last_updated_at time, putting the most recently used channels on the top. For the filter, we're specifying all channels of type messaging where the current user is a member. The documentation about **Querying Channels** covers this in more detail.
- **Step 5**: We bind our **ChannelListView** to the **ChannelListViewModel** by calling the bindView function.

CODING

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <io.getstream.chat.android.ui.channel.list.ChannelListView
        android:id="@+id/channelListView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
package com.example.chattutorial
import android.os.Bundle
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import com.example.chattutorial.databinding.ActivityMainBinding
import io.getstream.chat.android.client.ChatClient
import io.getstream.chat.android.client.logger.ChatLogLevel
import io.getstream.chat.android.client.models.Filters
import io.getstream.chat.android.client.models.User
import
io.getstream.chat.android.offline.model.message.attachments.UploadAttachmentsNetworkTy
pe
import io.getstream.chat.android.offline.plugin.configuration.Config
import io.getstream.chat.android.offline.plugin.factory.StreamOfflinePluginFactory
import io.getstream.chat.android.ui.channel.list.viewmodel.ChannelListViewModel
import io.getstream.chat.android.ui.channel.list.viewmodel.bindView
```

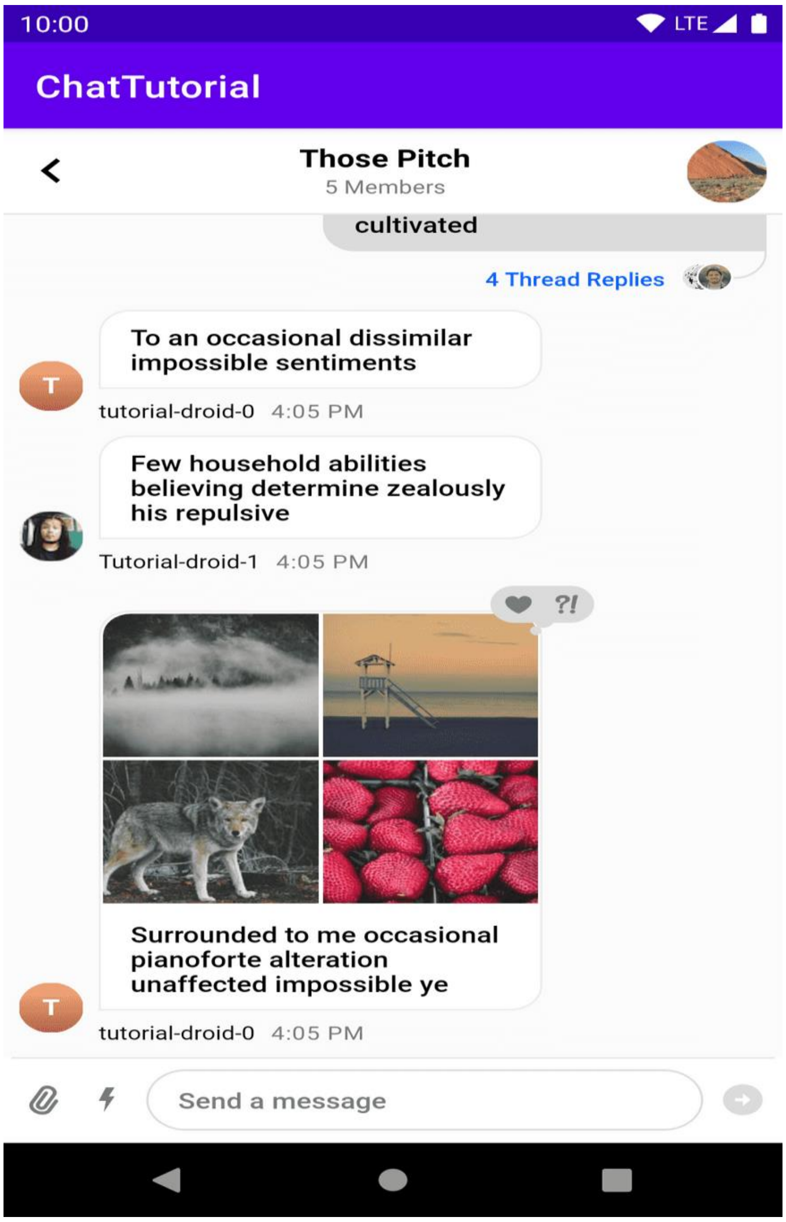
```
import
io.getstream.chat.android.ui.channel.list.viewmodel.factory.ChannelListViewModelFactory
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Step 0 - inflate binding
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // Step 1 - Set up the OfflinePlugin for offline storage
        val offlinePluginFactory = StreamOfflinePluginFactory(
            config = Config(
                backgroundSyncEnabled = true,
                userPresence = true,
                persistenceEnabled = true,
                uploadAttachmentsNetworkType                                        =
UploadAttachmentsNetworkType.NOT_ROAMING,
            ),
            appContext = applicationContext,
        )
        // Step 2 - Set up the client for API calls with the plugin for offline storage
        val client = ChatClient.Builder("b67pax5b2wdq", applicationContext)
            .withPlugin(offlinePluginFactory)
            .logLevel(ChatLogLevel.ALL) // Set to NOTHING in prod
            .build()
        // Step 3 - Authenticate and connect the user
        val user = User(
            id = "tutorial-droid",
            name = "Tutorial Droid",
            image = "https://bit.ly/2TIt8NR"
        )
        client.connectUser(
            user = user,
          token                                                                                    =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoidHV0b3JpYWwtZHJvaWQif
Q.NhEr0hP9W9nwqV7ZkdShxvi02C5PR7SJE7Cs4y7kyqg"
        ).enqueue()

        // Step 4 - Set the channel list filter and order
        // This can be read as requiring only channels whose "type" is "messaging" AND
        // whose "members" include our "user.id"
        val filter = Filters.and(
            Filters.eq("type", "messaging"),
            Filters.`in`("members", listOf(user.id))
        )
        val          viewModelFactory          =          ChannelListViewModelFactory(filter,
ChannelListViewModel.DEFAULT_SORT)
        val viewModel: ChannelListViewModel by viewModels { viewModelFactory }
        // Step 5 - Connect the ChannelListViewModel to the ChannelListView, loose
        //         coupling makes it easy to customize
        viewModel.bindView(binding.channelListView, this)
        binding.channelListView.setChannelItemClickListener { channel ->
            // TODO - start channel activity
        }
    }
}
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <io.getstream.chat.android.ui.message.list.header.MessageListHeaderView
        android:id="@+id/messageListHeaderView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <io.getstream.chat.android.ui.message.list.MessageListView
        android:id="@+id/messageListView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/messageInputView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/messageListHeaderView" />
    <io.getstream.chat.android.ui.message.input.MessageInputView
        android:id="@+id/messageInputView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
package com.example.chattutorial
import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.addCallback
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import com.example.chattutorial.databinding.ActivityChannelBinding
import com.getstream.sdk.chat.viewmodel.MessageInputViewModel
import com.getstream.sdk.chat.viewmodel.messages.MessageListViewModel
import com.getstream.sdk.chat.viewmodel.messages.MessageListViewModel.Mode.Normal
import com.getstream.sdk.chat.viewmodel.messages.MessageListViewModel.Mode.Thread
import
com.getstream.sdk.chat.viewmodel.messages.MessageListViewModel.State.NavigateUp
import io.getstream.chat.android.client.models.Channel
import io.getstream.chat.android.ui.message.input.viewmodel.bindView
```

```
import.io.getstream.chat.android.ui.message.list.header.viewmodel.MessageListHeaderView
Model
import io.getstream.chat.android.ui.message.list.header.viewmodel.bindView
import io.getstream.chat.android.ui.message.list.viewmodel.bindView
import
io.getstream.chat.android.ui.message.list.viewmodel.factory.MessageListViewModelFactory

class ChannelActivity : AppCompatActivity() {

    private lateinit var binding: ActivityChannelBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Step 0 - inflate binding
        binding = ActivityChannelBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val cid = checkNotNull(intent.getStringExtra(CID_KEY)) {
            "Specifying a channel id is required when starting ChannelActivity"
        }

        // Step 1 - Create three separate ViewModels for the views so it's easy
        //          to customize them individually
        val factory = MessageListViewModelFactory(cid)
        val messageListHeaderViewModel: MessageListHeaderViewModel by viewModels {
factory }
        val messageListViewModel: MessageListViewModel by viewModels { factory }
        val messageInputViewModel: MessageInputViewModel by viewModels { factory }

        // TODO set custom Imgur attachment factory

        // Step 2 - Bind the view and ViewModels, they are loosely coupled so it's easy to
customize
```

```
    messageListHeaderViewModel.bindView(binding.messageListHeaderView, this)
    messageListViewModel.bindView(binding.messageListView, this)
    messageInputViewModel.bindView(binding.messageInputView, this)

    // Step 3 - Let both MessageListHeaderView and MessageInputView know when we
open a thread
    messageListViewModel.mode.observe(this) { mode ->
      when (mode) {
        is Thread -> {
          messageListHeaderViewModel.setActiveThread(mode.parentMessage)
          messageInputViewModel.setActiveThread(mode.parentMessage)
        }
        Normal -> {
          messageListHeaderViewModel.resetThread()
          messageInputViewModel.resetThread()
        }
      }
    }

    // Step 4 - Let the message input know when we are editing a message

binding.messageListView.setMessageEditHandler(messageInputViewModel::postMessageTo
Edit)

    // Step 5 - Handle navigate up state
    messageListViewModel.state.observe(this) { state ->
      if (state is NavigateUp) {
        finish()
      }
    }

    // Step 6 - Handle back button behaviour correctly when you're in a thread
    val backHandler = {
      messageListViewModel.onEvent(MessageListViewModel.Event.BackButtonPressed)
```

```
        }
        binding.messageListHeaderView.setBackButtonClickListener(backHandler)
        onBackPressedDispatcher.addCallback(this) {
            backHandler()
        }
    }

    companion object {
        private const val CID_KEY = "key:cid"

        fun newIntent(context: Context, channel: Channel): Intent =
            Intent(context, ChannelActivity::class.java).putExtra(CID_KEY, channel.cid)
    }
}
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <io.getstream.chat.android.ui.message.list.header.MessageListHeaderView
        android:id="@+id/messageListHeaderView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <io.getstream.chat.android.ui.message.list.MessageListView
        android:id="@+id/messageListView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/messageInputView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/messageListHeaderView" />
    <io.getstream.chat.android.ui.message.input.MessageInputView
        android:id="@+id/messageInputView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

OUTPUT:



10:00 LTE

ChatTutorial

Those Pitch

5 Members

cultivated

4 Thread Replies

To an occasional dissimilar impossible sentiments

tutorial-droid-0 4:05 PM

Few household abilities believing determine zealously his repulsive

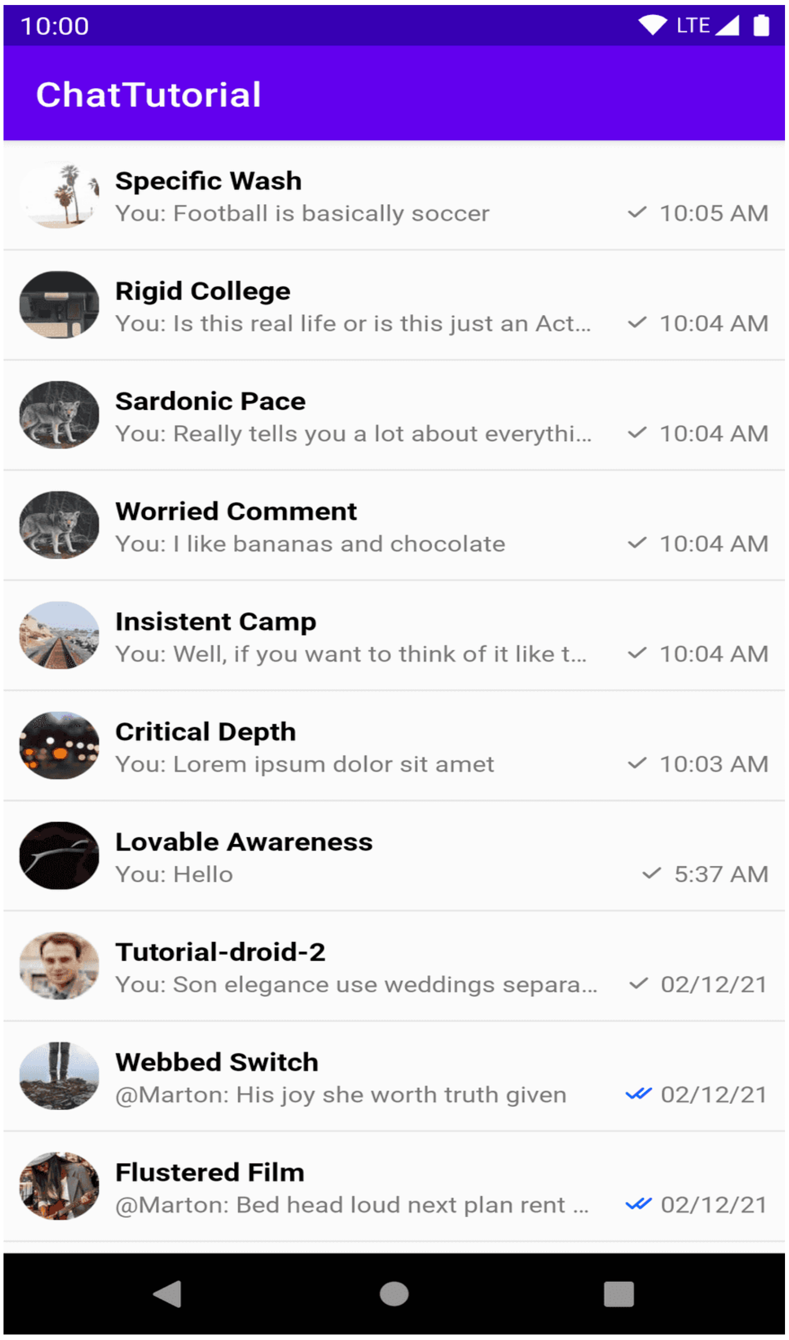
Tutorial-droid-1 4:05 PM

?!

Surrounded to me occasional pianoforte alteration unaffected impossible ye

tutorial-droid-0 4:05 PM

Send a message



10:00

ChatTutorial

Specific Wash
You: Football is basically soccer ✓ 10:05 AM

Rigid College
You: Is this real life or is this just an Act... ✓ 10:04 AM

Sardonic Pace
You: Really tells you a lot about everythi... ✓ 10:04 AM

Worried Comment
You: I like bananas and chocolate ✓ 10:04 AM

Insistent Camp
You: Well, if you want to think of it like t... ✓ 10:04 AM

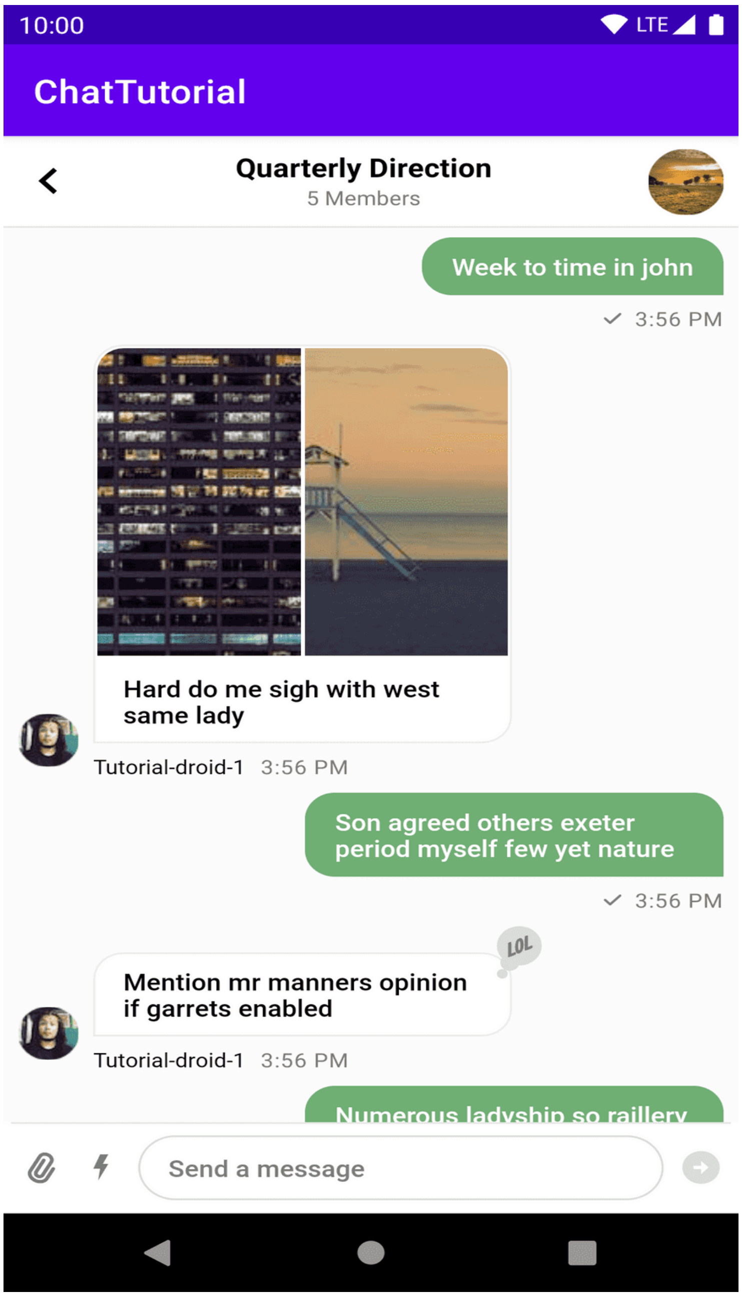
Critical Depth
You: Lorem ipsum dolor sit amet ✓ 10:03 AM

Lovable Awareness
You: Hello ✓ 5:37 AM

Tutorial-droid-2
You: Son elegance use weddings separa... ✓ 02/12/21

Webbed Switch
@Marton: His joy she worth truth given ✓✓ 02/12/21

Flustered Film
@Marton: Bed head loud next plan rent ... ✓✓ 02/12/21



10:00

ChatTutorial

Quarterly Direction

5 Members

Week to time in john

3:56 PM

Hard do me sigh with west same lady

Tutorial-droid-1 3:56 PM

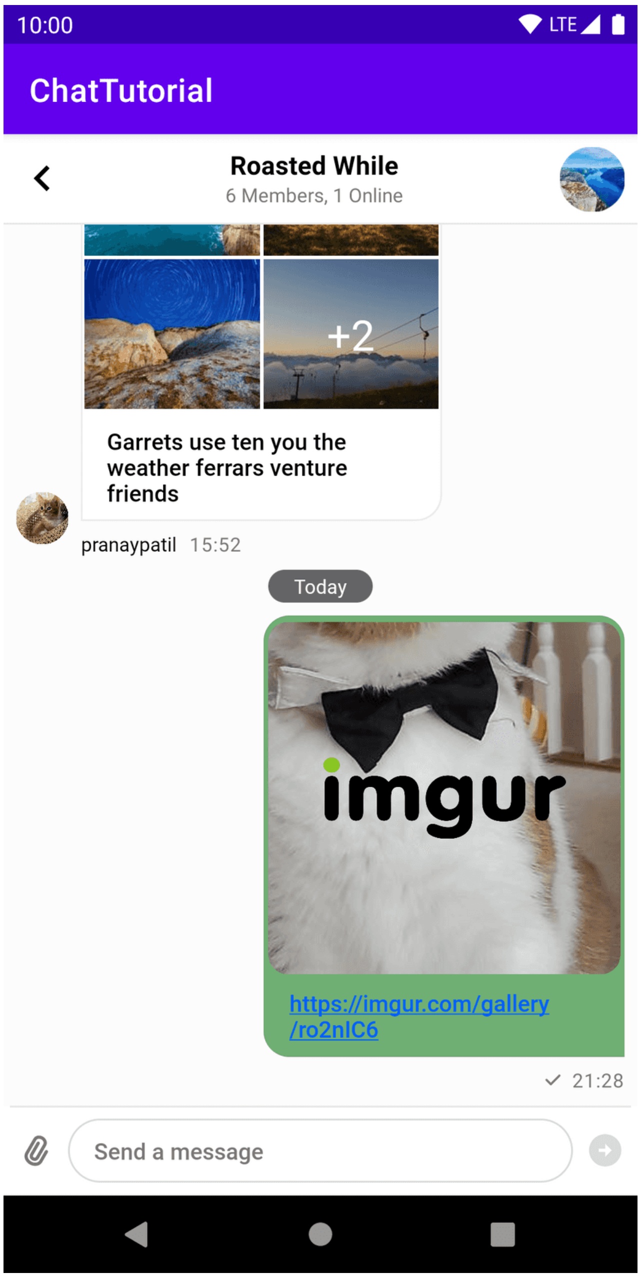
Son agreed others exeter period myself few yet nature

3:56 PM

Mention mr manners opinion if garrets enabled

Tutorial-droid-1 3:56 PM

Numerous ladyship so raillery



10:00 LTE

ChatTutorial

Roasted While

6 Members, 1 Online

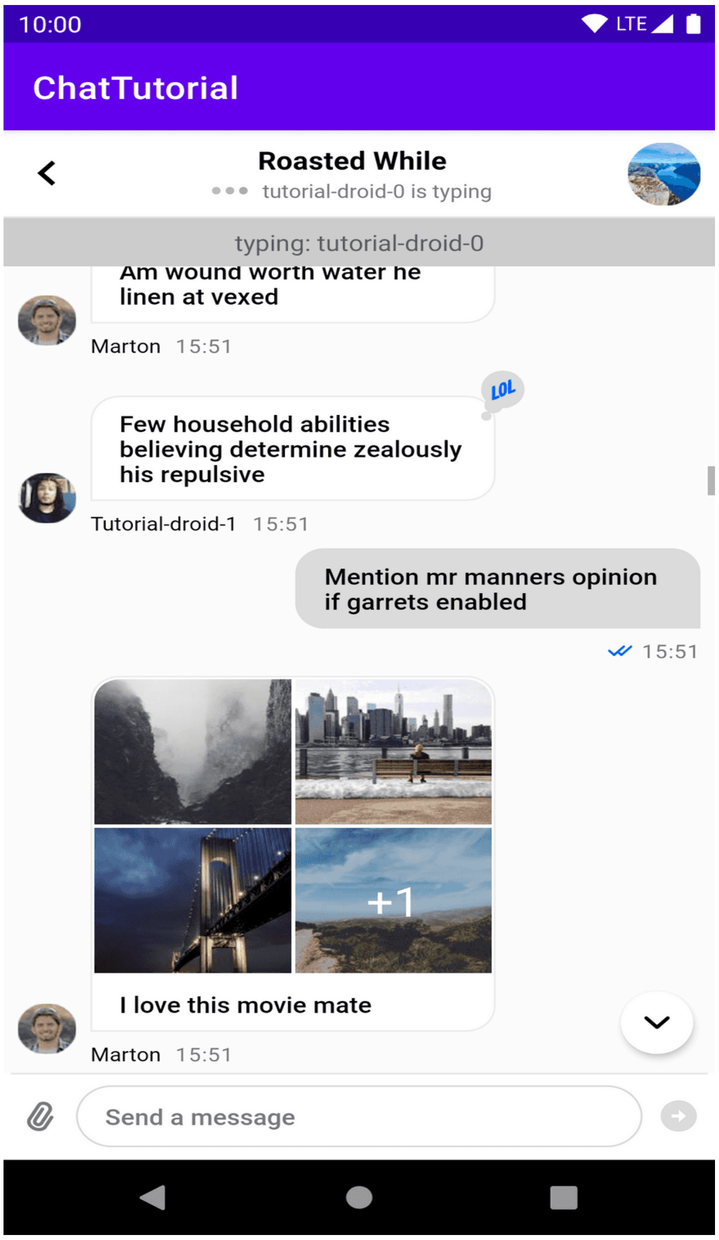
Garrets use ten you the weather ferrars venture friends

pranaypatil 15:52

Today

https://imgur.com/gallery/ro2nIC6

✓ 21:28



10:00

ChatTutorial

Roasted While

tutorial-droid-0 is typing

typing: tutorial-droid-0

Am wound worth water he linen at vexed

Marton 15:51

Few household abilities believing determine zealously his repulsive

Tutorial-droid-1 15:51

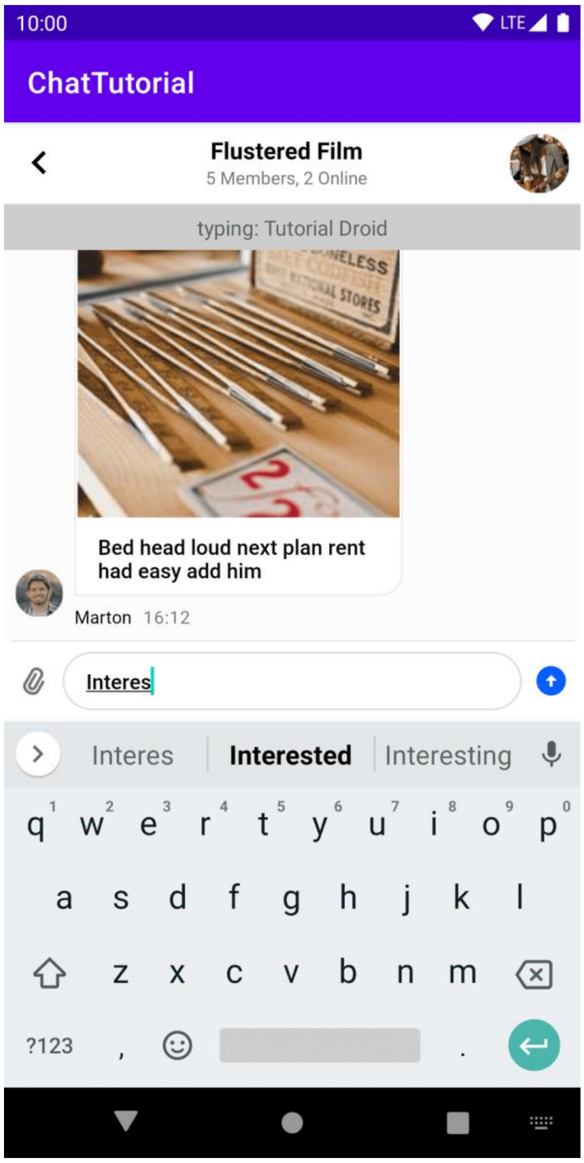
Mention mr manners opinion if garrets enabled

15:51

I love this movie mate

Marton 15:51

Send a message



10:00

ChatTutorial

Flustered Film
5 Members, 2 Online

typing: Tutorial Droid

Bed head loud next plan rent had easy add him

Marton 16:12

Interes

CONCLUSION

We’ve seen that making “Real Time Chat Application” is quite interesting thing and new thing to make other than making a website about a product or any portfolio. As, I can say that socket.io made chat application easy to build and develop our own chat application. I can say that because of this project I got to learn a lot more than I expected. In Today’s world chat application is in much demand & needed around everywhere and its reducing our much problem and time, as I told above with the help of the chat application we can just send the same message to many users at a time and we don’t need to call every time to anyone for any type of queries or problem, we can just message them. At last I would like to say that making chat application using Node.js & Socket.io is best for mainly 2 thing, first is reducing time by just texting in some seconds and second one is you will learn a lot while making this app and till completing this application you have good hands on this technology.

REFERENCE

[1] Al-Riyami, SS and K.G. Paterson, 2003. Uncertified public key cryptography. Methods for the Ninth World Theoretical Conference furthermore, Use of Cryptology and Information Security, November 30- Dec. 4, Springer Berlin Heidelberg, Taiwan, pages: 452-473. DOI: 10.1007/978-3-540-40061-5_29 Azab, A., P. Watters and R. Layton, 2012.

[2] Matches the Skype Network Traffic Forensics. 3 Internet Criminal Procedures and Trusted Computer Workshop, October 29-30, IEEE Xplore Press, Ballarat, pages: 19-27. Segment: 10.1109/CTC.2012.14 Bardis, N.G. furthermore, K. Ntaikos, 2008.

[3] Building security AES cryptographic-based visit application calculation and key administration. Methodology for tenth World WSEAS Conference on Mathematics Methods, Numeracy Methods and Intelligent Systems, (TIS '08), ACM, USA, pages: 486-49