

Date	8 Nov 2023
Team ID	590943
Project Name	ChatConnect - A Real-Time Chat And Communication App

Introduction

Project overview:

User Authentication:

- Users can create accounts using Firebase authentication.

Chatting Services:

- Real-time one-on-one chat. Group and family chat functionality.

Chat Themes:

- Customizable chat themes for a personalized experience.

Social Networking:

- Friend connections with profiles and status updates.

Media Sharing:

- Photo sharing in chats and social network.
- Video posting.

Video Features:

- Exercise videos.
- Other video content, potentially related to health.

Health and Wellness:

- Offline exercise functionality. Meditation features.

- Articles related to health.

Chatbot Integration (GPT):

- Integration of a GPT-powered chatbot for intelligent conversations.

Additional Features:

- Push notifications.
- User-friendly navigation and responsive design.
- Data security measures.

Future Considerations:

- Regular updates and improvements.
- Potential integration with additional services or APIs.

Purpose:

Communication and Connection:

- Facilitate real-time communication between users, including one-on-one, group, and family chats.
- Enhance social connections by providing a platform for friends and family to interact.

Personalization and Engagement:

- Allow users to personalize their chat experience with customizable themes.
- Foster engagement through various media-sharing options like photos and videos.

Health and Wellness:

- Promote user well-being by integrating features like exercise videos, offline exercises, meditation, and health-related articles.

Entertainment and Education:

- Provide a platform for sharing and viewing entertaining and educational videos.
- Enhance user experience with a diverse range of multimedia content.

Innovative Chatbot Integration:

- Integrate a chatbot powered by GPT to offer intelligent and natural language interactions.
- Provide users with information, answer queries, and enhance overall user experience through conversational AI.

User Authentication and Security:

- Ensure secure user authentication using Firebase, prioritizing the privacy and data security of users.
- Implement security measures to protect sensitive information and user data.

Future Expansion and Improvements:

- Lay the groundwork for future updates and improvements based on user feedback.

- Consider potential integrations with additional services or APIs to expand the application's functionality.

Versatility and Accessibility:

- Create a versatile app that caters to a wide range of user needs, from communication to health and entertainment.
- Enhance accessibility by including offline features for users with limited internet access.

Overall, the purpose is to offer a holistic and engaging user experience, combining communication, entertainment, and well-being within a single application. The project aims to meet the diverse needs of users, promoting connectivity and overall user satisfaction.

Literature Survey

Existing problem:

1. Lack of Comprehensive Chat Applications: The current landscape of chat applications often lacks a comprehensive solution that combines various features, such as real-time messaging, multimedia sharing, social networking, and health-related functionalities, within a single platform. Users are often required to use multiple applications for different purposes, leading to fragmentation and reduced user convenience.
2. Limited Integration of Health and Wellness Features: Many existing chat applications focus primarily on communication but lack integration with health and wellness features. Users seeking a holistic lifestyle app may find it challenging to seamlessly combine their social interactions with health-related activities such as exercise, meditation, or access to health-related articles.
3. Unexplored Potential of Conversational AI: While chatbots are becoming more prevalent, there is still an untapped potential for leveraging advanced conversational AI, such as GPT-powered chatbots. Current applications often provide basic automated responses, and there is room for improvement in creating more intelligent and engaging conversational experiences.
4. Privacy and Security Concerns: User privacy and data security are recurring concerns in the realm of chat applications. The

existing solutions may not provide robust security measures, potentially exposing users to privacy breaches and data vulnerabilities. Addressing these concerns is crucial for gaining and maintaining user trust.

5. Fragmented User Experience: The lack of a unified platform that seamlessly integrates various features may result in a fragmented user experience. Users may find it inconvenient to switch between different applications for communication, socializing, and health-related activities. A more cohesive and integrated solution is needed to enhance user satisfaction.

6. Limited Offline Functionality: In regions with unreliable or limited internet access, users often face challenges with online-only applications. Current chat applications may not adequately address the need for offline functionality, especially concerning exercises, meditation, and content consumption, limiting the app's accessibility.

References:

//write the reference for this project

Problem Statement Definition:

The project aims to address several critical issues observed in the current landscape of chat applications and related services. These

issues are identified through an extensive literature review/survey and are essential to understanding the motivation behind the development of the proposed solution.

1. **Fragmented User Experience:** The existing chat applications lack a unified platform that seamlessly integrates communication, social networking, and health-related features. Users often experience fragmentation, requiring them to navigate between multiple applications for different functionalities, leading to a disjointed and less user-friendly experience.

2. **Limited Integration of Health and Wellness:** Current chat applications predominantly focus on communication aspects, neglecting the integration of health and wellness features. Users seeking a holistic lifestyle app face challenges in seamlessly combining social interactions with health-related activities such as exercise, meditation, and access to health-related articles.

3. **Underutilization of Conversational AI:** While chatbots are becoming more commonplace, the potential of advanced conversational AI, such as GPT-powered chatbots, remains largely untapped. Existing applications often provide basic automated responses, missing the opportunity to create more intelligent, engaging, and personalized conversational experiences for users.

4. **Privacy and Security Concerns:** User privacy and data security are persistent concerns in the realm of chat applications. Current solutions may lack robust security measures, exposing users to potential privacy breaches and data vulnerabilities. Addressing these concerns is paramount for ensuring user trust and the long-term success of the application.

5. **Offline Functionality Challenges:** In regions with unreliable or limited internet access, users face difficulties with applications

that heavily depend on online connectivity. Existing chat applications may not adequately cater to the need for offline functionality, particularly in areas related to exercise, meditation, and content consumption, limiting the accessibility of the application.

By addressing these key problems, the project aims to provide a comprehensive and integrated solution that enhances user experience, promotes holistic well-being, and ensures a secure and privacy-aware environment for users.

Ideation & Proposed Solution

Empathy Map

Template



Empathy map canvas

Use this framework to empathize with a customer, user, or any person who is affected by a team's work. Document and discuss your observations and note your assumptions to gain more empathy for the people you serve.

Originally created by Dave Gray at

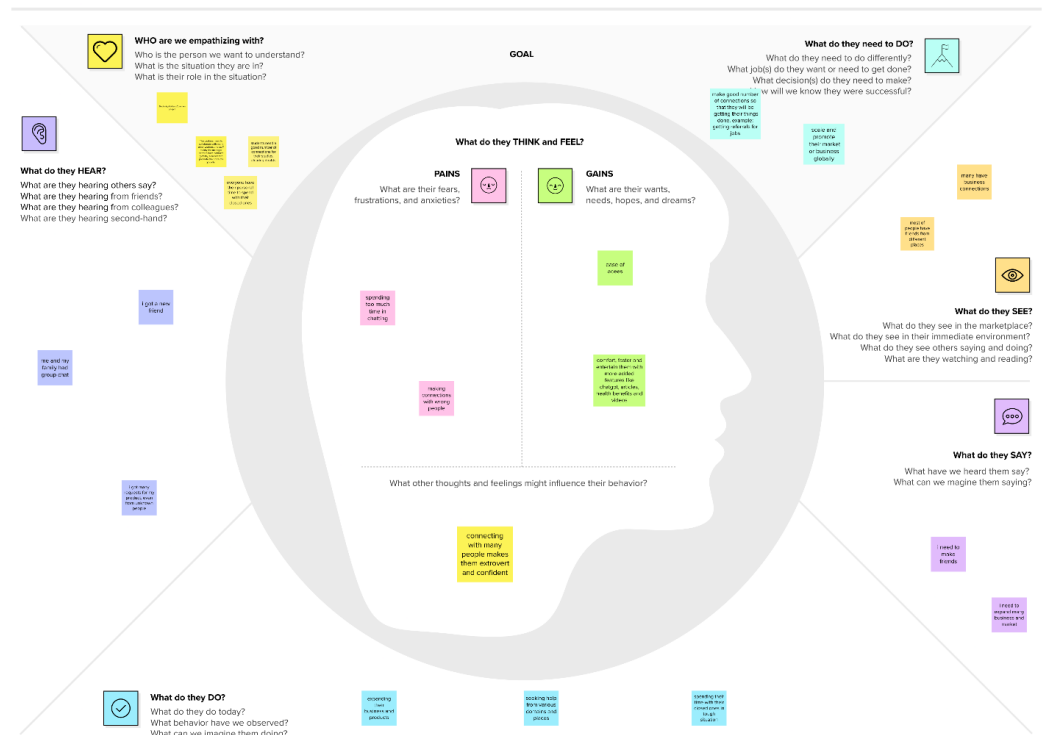


Share template feedback

35

Develop shared understanding and empathy

Summarize the data you have gathered related to the people that are impacted by your work. It will help you generate ideas, prioritize features, or discuss decisions.



Need some inspiration?
See a finished version of this template to kickstart your work.
[Open example](#)



BrainStorming

Async brainstorming

A brainstorm method tailored for async collaboration

INTRODUCTION

Design an inclusive and effective brainstorm with this template tailored for async collaboration. These activities are great when calendars are packed, participants can't meet live because of time zone conflicts, or when you just want to give collaborators more time to think about their ideas.



People



Time



Difficulty

AGENDA

- 1 Define your problem statement
- 2 Brainstorm
- 3 Group ideas
- 4 Prioritize

PREPARATION FOR ASYNC WORK

Before sharing this mural with collaborators, review the facilitation recommendations for async projects. Then, define the problem statement and fill out section 1.

Provide collaborators with a timeline for each phase of the brainstorm – then explain the activity checkpoints below. Consider recording a quick explainer video, if collaborators are unfamiliar with async collaboration.

Give a brief overview or record a loom of the project



ACTIVITY CHECKPOINTS

Add your profile picture here to help track the team's progress. After you finish an activity, move your avatar below.

Tip

You can easily add your profile image by right-clicking your avatar in the lower part of the mural - selecting the option "copy image".

Left-click any part of the mural and paste the image with ctrl (cmd) + v.

Hello!

Starting point - I have read the problem statement

Add your avatar here

Brainstorm completed - I'm ready for grouping ideas

Move your avatar here

Group ideas completed

Move your avatar here

Goal reached - I have finished the prioritize step

Move your avatar here

Share your feedback

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a "How Might We" statement. This will be the focus of your brainstorm.

How Might We make the people stay connected with their friends and relatives and also make the their conversation easier,simpler and faster?

Give a brief overview or record a loom of the project



2 Brainstorm

Write down any ideas that come to mind that address your problem statement. Remember, the key rules of brainstorming are:

Advice

- Defer judgement
- Go for volume
- Build on the ideas of others
- Stay on topic
- Encourage wild ideas
- Be visual!

PRO TIP: Select a sticky note and click the pencil icon in the menu to sketch.



3 Group ideas

The facilitator should group all the ideas from the brainstorming process (step 2). After that, you should add your opinions by adding arrows to point ideas into other groups and sticky notes and icons to share your thoughts.

PRO TIP: This is a great place to use color coding. You can change the color of multiple sticky notes at once.

Devendra

AI Chatbot Integration: Enhance user experience by integrating chatbots for instant support and feedback.

Multi-Platform Support: Ensure the app is accessible and functional across various devices and operating systems.

Video and Voice Calls: Implement high-quality video and voice calling features for seamless communication.

Vikram

Privacy and Security: Prioritize user data security and privacy, ensuring compliance with relevant regulations.

Language Translation: Offer real-time language translation for users from different linguistic backgrounds.

Group Chat Enhancements: Introduce features like polls, quizzes, and shared documents to enrich group interactions.

Shajeth

Real-time Discussion: Enable live chat or voice discussions for immediate problem-solving and idea exchange.

Offline Messaging: Support asynchronous messaging for users who may not be online simultaneously.

Meeting Recording: Offer an option to record meetings for participants who cannot attend live sessions.

Thavanesh

AI Content Suggestions: Utilize AI algorithms to recommend relevant content, articles, or resources based on user interests.

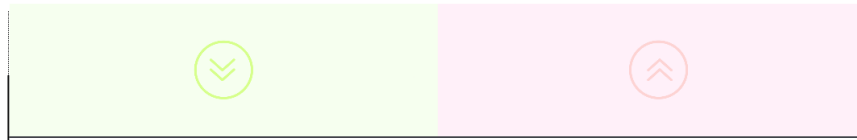
Interactive Quizzes: Create engaging quizzes or polls to assess user knowledge and gather feedback.

Virtual Events: Host virtual workshops, webinars, or Q&A sessions to foster community and expertise sharing.

4 Prioritize

The facilitator should copy and paste the groups from step 3 into this area and setup the vote details. Let's vote! Add a vote on sticky notes you think are a high priority. You can also add your vote to an entire group.

Vote area



Low-priority

High-priority

Requirement Analysis

1) Functional Requirements:

The functional requirements for the Chatting App, utilizing Firebase as its backend and integrating the ChatGPT API, are outlined below:

1.1 User Authentication:

The system shall provide secure user authentication using Firebase Authentication, allowing users to create accounts, log in, and recover forgotten passwords.

1.2 Real-time Chatting:

The application shall enable real-time chat functionality, utilizing Firebase Realtime Database to synchronize messages instantly between users. Users should be able to send and receive text messages seamlessly.

1.3 Multimedia Messaging:

The system shall support multimedia messaging, allowing users to send and receive images, videos, and other multimedia files through Firebase Cloud Storage. The multimedia content should be accessible within the chat interface.

1.4 Chat History:

The application shall maintain a chat history for each user, stored securely in Firebase Realtime Database, ensuring users can retrieve past conversations upon login.

1.5 User Presence:

The system shall display user presence status (online, offline) in real-time, utilizing Firebase Cloud Firestore to track and update user statuses.

1.6 Push Notifications:

The application shall implement push notifications through Firebase Cloud Messaging (FCM) to notify users of new messages or relevant updates, even when the app is in the background.

1.7 Group Chat:

The system shall support group chatting functionality, enabling users to create, join, and participate in group conversations. Group details and memberships will be managed through Firebase Realtime Database.

1.8 Emoticons and Reactions:

The application shall allow users to express themselves using emoticons and reactions within the chat, enhancing the overall user experience.

1.9 Integration with ChatGPT API:

The system shall integrate the ChatGPT API to provide advanced natural language processing capabilities, allowing users to engage in intelligent and context-aware conversations.

2) Non-Functional Requirements:

2.1 Performance:

The application shall demonstrate low-latency performance for sending and receiving messages, ensuring a smooth and responsive user experience.

2.2 Scalability:

The system shall be designed to scale horizontally, accommodating an increasing number of users and messages without compromising performance. Firebase services should be configured for scalability.

2.3 Security:

The application shall prioritize the security of user data, employing Firebase Authentication for secure user identity verification and Firebase Realtime Database and Cloud Firestore security rules to restrict unauthorized access.

2.4 Reliability:

The system shall be highly reliable, with Firebase Realtime Database providing data persistence and backup. Periodic data backups and Firebase Cloud Functions can be employed to ensure system reliability.

2.5 User Privacy:

The application shall adhere to privacy regulations and best practices, ensuring that user data is handled with confidentiality. Firebase services will be configured to prioritize user privacy.

2.6 Availability:

The system shall aim for high availability, minimizing downtime and ensuring that users can access the chat application consistently. Firebase Hosting can be utilized to enhance availability.

2.7 Cross-Platform Compatibility:

The application shall be compatible with multiple platforms (iOS, Android), ensuring a consistent user experience across different devices. Firebase services and ChatGPT API integration should support cross-platform development.

2.8 User Experience:

The application shall prioritize a user-friendly interface, with intuitive navigation and responsive design. The integration of ChatGPT should enhance the conversational experience and provide meaningful interactions.

These functional and non-functional requirements collectively define the features, performance expectations, and quality attributes that the Chatting App should exhibit, ensuring a robust, secure, and user-friendly communication platform.

Project Design

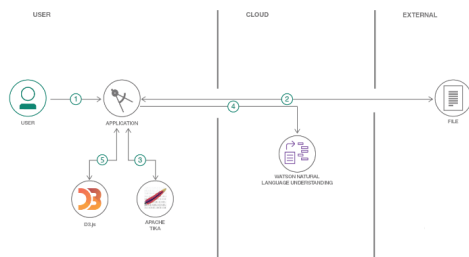
Data Flow Diagram & User Stories

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

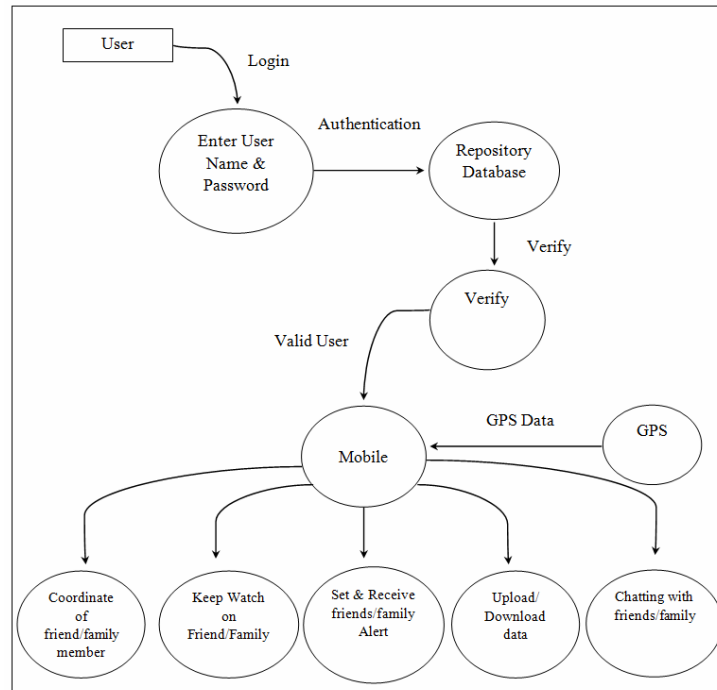
Example: [\(Simplified\)](#)

Flow



1. User configures credentials for the Watson Natural Language Understanding service and starts the app.
2. User selects data file to process and load.
3. Apache Tika extracts text from the data file.
4. Extracted text is passed to Watson NLU for enrichment.
5. Enriched data is visualized in the UI using the D3.js library.

Example: DFD Level 0 (Industry Standard)



User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register by connecting my Gmail account	Medium	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
				and access the dashboard.		
	Login	USN-5	As a user, I can log into the application by entering email & password	I can log in with correct credentials and access the app.	High	Sprint-1
	Dashboard			I can view my chat history, contacts, and user settings.		
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can receive a confirmation email, click confirm, and access the app.	High	Sprint-1
		USN-2	As a user, I will receive a confirmation email once I have registered for the application.		High	Sprint-2
	Dashboard	USN-3	I can view my chat history, contacts, and user settings.	I can register by connecting my Gmail account and access the dashboard.	Medium	Sprint-1
Customer Care Executive	Chat Management	USN-4	As a customer care executive, I can view and manage customer chats.	I can see a list of customer chats, reply to them, and mark them as resolved.	High	Sprint-3
		USN-5	I can search for specific customer chats based on user details or keywords.	I can search and filter customer chats effectively.	Medium	Sprint-3
Administrator	User Management	USN-6	As an administrator, I can manage user accounts, including creating, updating, and deleting them.	I can add, edit, and remove user accounts.	High	
		USN-7	I can view and export user activity and usage reports.	I can generate and download user activity reports.		

Solution Architecture

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Example - Solution Architecture Diagram:

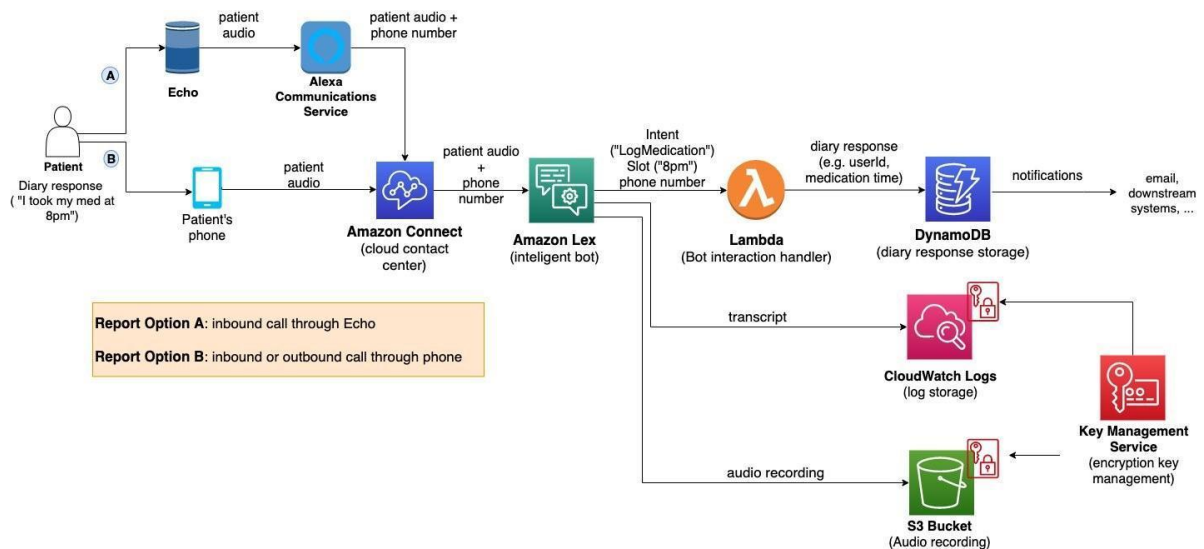
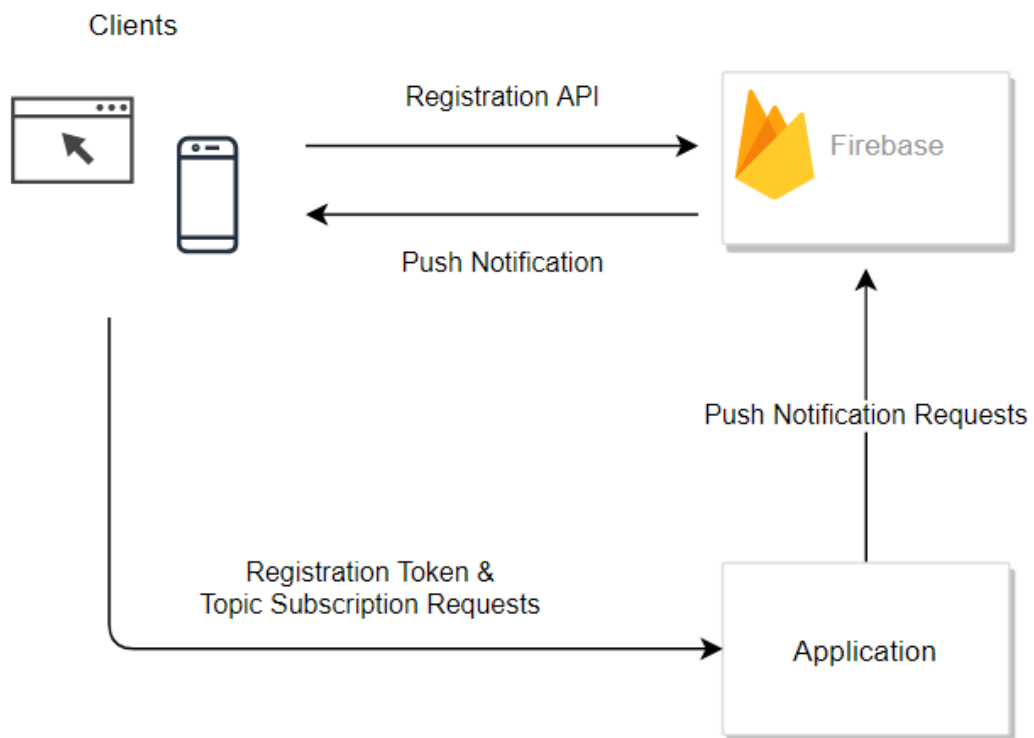
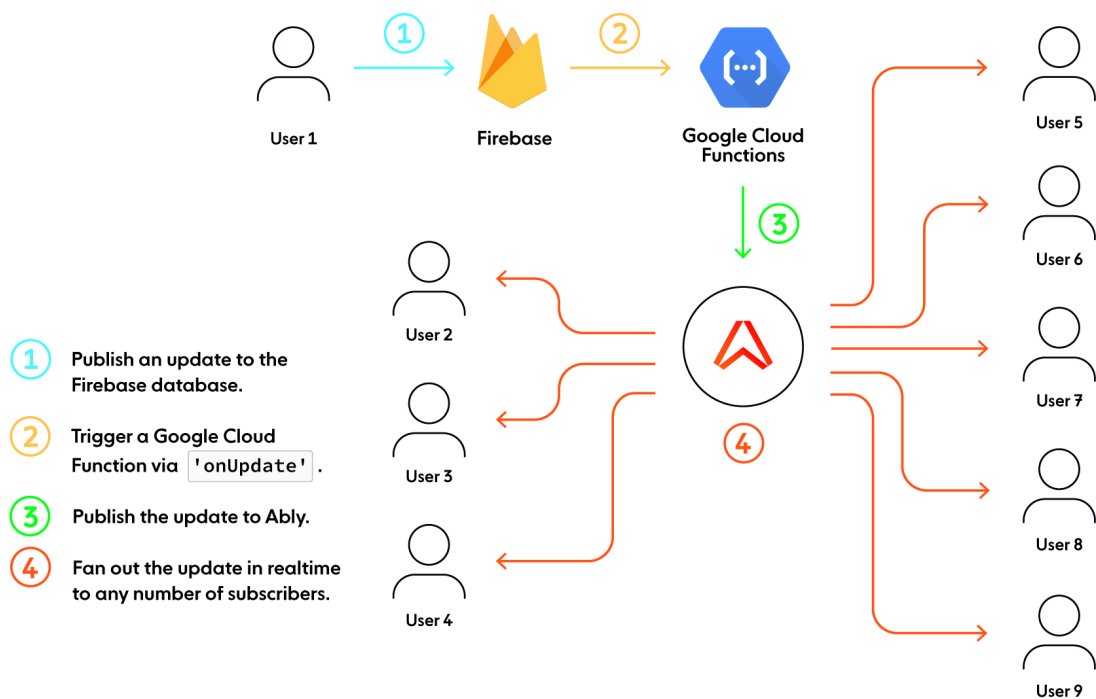


Figure 1: Architecture and data flow of the voice patient diary sample application

Solution Architecture Diagram



User Registration



Chatting with user's using Firebase's Real time database

Reference:

<https://aws.amazon.com/blogs/industries/voice-applications-in-clinical-research-powered-by-ai-on-aws-part-1-architecture-and-design-considerations/>

Project Planning Phase

Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

Product Backlog, Sprint Schedule, and Estimation

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Devendra
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Devendra
Sprint-2		USN-3	As a user, I can register for the application through Google	2	Low	Devendra
Sprint-1		USN-4	As a user, I can register for the application through Gmail	2	Medium	Devendra
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	1	High	Devendra
	Dashboard	USN-6	As a user, I can chat with bot	1	Medium	Devendra
Sprint-3		USN-7	As a user, I can edit my profile information.	2	Medium	Devendra
Sprint-3		USN-8	As a user, I receive real-time notifications for new messages.	2	Medium	Devendra
Sprint-4		USN-9	As a user, I can start one-on-one chats with other users.	3	High	Devendra

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	3 Days	22 Oct 2023	25 Oct 2023	20	25 Oct 2023
Sprint-2	20	2 Days	25 Oct 2023	27 Oct 2023	20	27 Oct 2023
Sprint-3	20	2 Days	27 Oct 2023	29 Oct 2023	20	29 Oct 2023
Sprint-4	20	3 Days	29 Oct 2023	1 Nov 2023	20	1 Nov 2023

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{\text{sprint duration}}{\text{velocity}} = \frac{20}{10} = 2$$

Burndown Chart:

A burndown chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

<https://www.visual-paradigm.com/scrum/scrum-burndown-chart/>
<https://www.atlassian.com/agile/tutorials/burndown-charts>

Reference:

<https://www.atlassian.com/agile/project-management>
<https://www.atlassian.com/agile/tutorials/how-to-do-scrum-with-jira-software>
<https://www.atlassian.com/agile/tutorials/epics>
<https://www.atlassian.com/agile/tutorials/sprints>
<https://www.atlassian.com/agile/project-management/estimation>
<https://www.atlassian.com/agile/tutorials/burndown-charts>

Coding And Solutioning

1) Family chat

Single ValueEventListener for User Timestamps and Data:

The code structure has been modified to use a single ValueEventListener for both fetching user timestamps and user data. This simplifies the structure, making it easier to understand. It also reduces the number of nested listeners, which can improve code maintainability.

Loading State Handling:

The logic for handling the loading state remains intact. However, I mentioned considering the addition of a loading spinner or progress bar to indicate that data is being fetched. This visual indicator can enhance the user experience, especially when there is a delay in retrieving data.

Family chat code

```
class FamilyListsFragment : Fragment() {  
    private lateinit var binding: FragmentFamiliyListsBinding  
    private lateinit var mDbRef: DatabaseReference  
    private lateinit var mAuth: FirebaseAuth  
    private lateinit var userList: ArrayList<User>  
    private lateinit var adapter: RecyclearUserAdapter  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,
```



```

        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentFamiliyListsBinding.inflate(layoutInflater)

        val viewPager = activity?.findViewById<ViewPager2>(R.id.viewPager)

        mDbRef = FirebaseDatabase.getInstance().getReference()
        mAuth = FirebaseAuth.getInstance()

        val i = Intent()
        val senderRoom = i.getStringExtra("receiverUid")
        userList = ArrayList()
        Log.d("idididiididid",senderRoom.toString())
        binding.recycler.layoutManager = LinearLayoutManager(context)
        adapter = RecycleUserAdapter(requireContext(), userList,
senderRoom.toString())
        binding.recycler.adapter = adapter

        mDbRef.child("list").child(mAuth.uid.toString())
            .addValueEventListener(object :
                ValueEventListener {
                    override fun onDataChange(snapshot: DataSnapshot) {

                        val userTimestampMap = mutableMapOf<String, Long>()

                        for (snap in snapshot.children) {
                            val userID = snap.key

```

```

        val timestamp = snap.getValue(Long::class.java)
        if (userID != null && timestamp != null) {
            userTimestampMap[userID] = timestamp
        }
    }

    // Sort the user IDs based on timestamps in descending order
    val sortedUserIDs =
        userTimestampMap.keys.sortedByDescending {
            userTimestampMap[it] }

    userList.clear() // Clear the list before adding new users
    for (userID in sortedUserIDs) {
        mDbRef.child("user").child(userID)
            .addListenerForSingleValueEvent(object : ValueEventListener {
                override fun onDataChange(userDataSnapshot:
                    DataSnapshot) {
                    val currentUser =
                        userDataSnapshot.getValue(User::class.java)
                    if (currentUser != null) {
                        mDbRef.child("Family").child("FamilyMembers")
                            .child(mAuth.currentUser?.uid!!)
                            .addListenerForSingleValueEvent(object :
                                ValueEventListener {
                                    override fun onDataChange(snapshot:
                                        DataSnapshot) {
                                        if (snapshot.hasChild(currentUser.uid!!)) {

```

```
        userList.add(currentUser)
        adapter.notifyDataSetChanged()
        binding.cuteDogLootie.isVisible = false
        binding.noGroupText.isVisible = false
    }
}
```

```
    override fun onCancelled(error: DatabaseError) {
        // Handle error
    }
})
}
```

```
    override fun onCancelled(error: DatabaseError) {
        // Handle error
    }
})
}
```

```
    override fun onCancelled(error: DatabaseError) {
    }
})
```

```

        if(userList.isEmpty()){
            binding.cuteDogLootie.isVisible = true
            binding.noGroupText.isVisible = true
            binding.noGroupText.setText("No Family Added")
        }else{
            binding.cuteDogLootie.isVisible = false
            binding.noGroupText.isVisible = false
        }

        return binding.root
    }

}

```

Group Chat Code

```

class GroupListFragment : Fragment() {
    private lateinit var mDbRef: DatabaseReference
    private lateinit var mAuth: FirebaseAuth
    private lateinit var binding: FragmentGroupListBinding
    private lateinit var groupList: ArrayList<Group>
    private lateinit var adapterGroup: GroupViewAdapter
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

```

```
binding = FragmentGroupListBinding.inflate(layoutInflater,container,false)
```

```
mDbRef = FirebaseDatabase.getInstance().getReference()
```

```
mAuth = FirebaseAuth.getInstance()
```

```
groupList = ArrayList()
```

```
binding.recycler.layoutManager = LinearLayoutManager(context)
```

```
adapterGroup = GroupViewAdapter(requireContext(), groupList)
```

```
binding.recycler.adapter = adapterGroup
```

```
mDbRef.child("Groups").child("GroupMembers")
```

```
    .addValueEventListener(object : ValueEventListener {
```

```
        override fun onDataChange(snapshot: DataSnapshot) {
```

```
            val userGroups = mutableListOf<String>()
```

```
            val userUid = FirebaseAuth.getInstance().currentUser?.uid
```

```
            for (groupSnapshot in snapshot.children) {
```

```
                val groupID = groupSnapshot.key
```

```
                val groupMemberList = groupSnapshot.children.map { it.key
```

```
            }.toSet()
```

```
            Log.d("listishereagain", groupMemberList.toString())
```

```
            if (groupID != null && userUid != null) {
```

```
                for (user in groupMemberList) {
```

```
                    if (userUid == user) {
```

```
                        userGroups.add(groupID)
```

```

        Log.d("listishereagain", userUid)
    }
}
}
}

```

```

Log.d("listishereagain", userGroups.toString())
groupList.clear() // Clear the list before adding new users
for (userID in userGroups) {
    // Fetch user data using the user ID from the "user" node
    mDbRef.child("Groups").child("Data").child(userID)
        .addValueEventListener(object :
            ValueEventListener {
                override fun onDataChange(userDataSnapshot:
DataSnapshot) {

                    val currentUser =
                        userDataSnapshot.getValue(Group::class.java)
                    if (currentUser != null) {
                        groupList.add(currentUser)

                    }

                    binding.cuteDogLootie.isVisible = false
                    binding.noGroupText.isVisible = false
                    adapterGroup.notifyDataSetChanged()
                }
            }
        )
}

```

```

        override fun onCancelled(error: DatabaseError) {
            }
        })
    }
}

    override fun onCancelled(error: DatabaseError) {
        // Handle the error if needed
    }
})

if(groupList.isEmpty()){
    binding.cuteDogLootie.isVisible = true
    binding.noGroupText.isVisible = true
    binding.noGroupText.setText("No Group Added")
}else{
    binding.cuteDogLootie.isVisible = false
    binding.noGroupText.isVisible = false
}
return binding.root
}
}

```

Chat Gpt integration code

Initialization and UI Setup:

The activity extends AppCompatActivity.

UI elements are initialized using findViewById for a RecyclerView, EditText, and ImageButton.

A custom layout binding (ActivityChatGptBinding) is used to inflate the layout.

Status Bar and Navigation Bar Configuration:

The code adjusts the status bar and navigation bar color for devices running Android Lollipop and above.

RecyclerView Setup:

A RecyclerView is set up with a custom adapter (MessageAdapterChatGpt) to display chat messages.

Clicking the send button triggers the addition of the user's message to the chat.

Adding Messages to Chat:

The addToChat function adds a message to the chat and updates the UI on the main thread.

API Call to OpenAI GPT-3:

The callAPI function constructs a JSON request and sends it to the OpenAI GPT-3 API using OkHttp.

It handles API response, parsing the generated text, and adding it to the chat.

Updating User Status:

The updateUserStatus function updates the user's online status in the Firebase Realtime Database when the activity is resumed or paused.

```
class ChatGPT : AppCompatActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var messageEditText: EditText
    private lateinit var sendButton: ImageButton
    private lateinit var messageListGpt: MutableList<MessageGpt>
    private lateinit var messageAdapter: MessageAdapterChatGpt
    private val JSON = "application/json; charset=utf-8".toMediaType()
    private val client = OkHttpClient()

    private lateinit var binding: ActivityChatGptBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityChatGptBinding.inflate(layoutInflater)
        setContentView(binding.root)

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            val window: Window = this.window
```

```
        window.statusBarColor = ContextCompat.getColor(this, R.color.fav)

        window.decorView.systemUiVisibility =
View.SYSTEM_UI_FLAG_LIGHT_STATUS_BAR
    }

    this.window.navigationBarColor = resources.getColor(R.color.fav)


    messageListGpt = mutableListOf()


    recyclerView = findViewById(R.id.recycler_view)


    messageEditText = findViewById(R.id.message_edit_text)
    sendButton = findViewById(R.id.send_btn)


    // setup recycler view
    messageAdapter = MessageAdapterChatGpt(messageListGpt)
    recyclerView.adapter = messageAdapter
    recyclerView.layoutManager = LinearLayoutManager(applicationContext)


    sendButton.setOnClickListener {
        val question = messageEditText.text.toString().trim()
        addToChat(question, MessageGpt.SENT_BY_ME)
        messageEditText.text.clear()
        callAPI(question)
        binding.robotHi.visibility = View.GONE
    }
}
```

```

private fun addToChat(message: String, sentBy: String) {
    runOnUiThread {
        messageListGpt.add(MessageGpt(message, sentBy))
        messageAdapter.notifyDataSetChanged()
        recyclerView.smoothScrollToPosition(messageAdapter.itemCount)
    }
}

private fun addResponse(response: String) {
    messageListGpt.removeAt(messageListGpt.size - 1)
    addToChat(response, MessageGpt.SENT_BY_BOT)
}

private fun callAPI(question: String) {
    messageListGpt.add(MessageGpt("Typing... ", MessageGpt.SENT_BY_BOT))

    val jsonBody = JSONObject().apply {
        put("model", "text-davinci-003")
        put("prompt", question)
        put("max_tokens", 4000)
        put("temperature", 0)
    }

    val body = jsonBody.toString().toRequestBody(JSON)
    val request = Request.Builder()
        .url("https://api.openai.com/v1/completions")

```

```
        .header("Authorization", "Bearer  
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")
```

```
        .post(body)
```

```
        .build()
```

```
client.newCall(request).enqueue(object : Callback {  
    override fun onFailure(call: Call, e: IOException) {  
        addResponse("Failed to load response due to " + e.message)  
    }  
})
```

```
    override fun onResponse(call: Call, response: Response) {  
        if (response.isSuccessful) {  
            var jsonObject: JSONObject? = null  
            try {  
                jsonObject = JSONObject(response.body!!.string())  
                val jsonArray = jsonObject.getJSONArray("choices")  
                val result = jsonArray.getJSONObject(0).getString("text")  
                addResponse(result.trim())  
            } catch (e: JSONException) {  
                e.printStackTrace()  
            }  
        } else {  
            addResponse("Failed to load response due to " +  
response.body?.string())  
        }  
        response.close()  
    }  
}
```

```
    })  
}
```

```
override fun onResume() {  
    super.onResume()  
    updateUserStatus(true)  
}
```

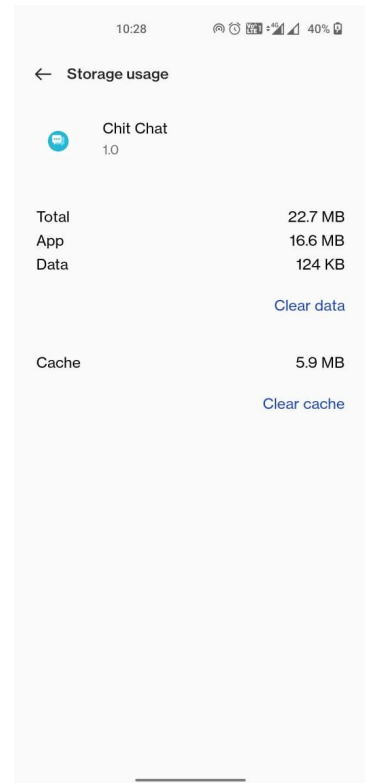
```
override fun onPause() {  
    super.onPause()  
    updateUserStatus(false)  
}
```

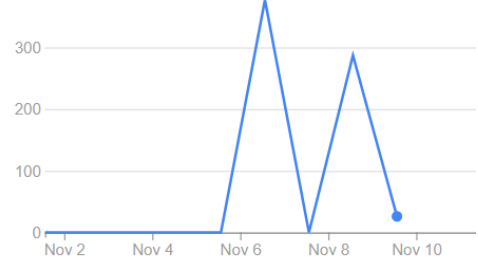

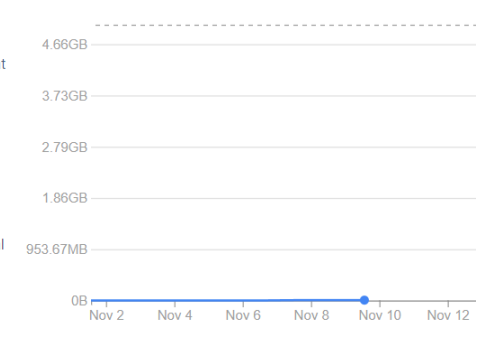
```
private fun updateUserStatus(status: Boolean) {  
    val mAuth = FirebaseAuth.getInstance()  
    val uid = mAuth.currentUser?.uid  
    val firebaseDef =  
FirebaseDatabase.getInstance().getReference("user/$uid")  
    firebaseDef.child("status").setValue(status)  
}  
}
```

Project Development Phase
Model Performance Test

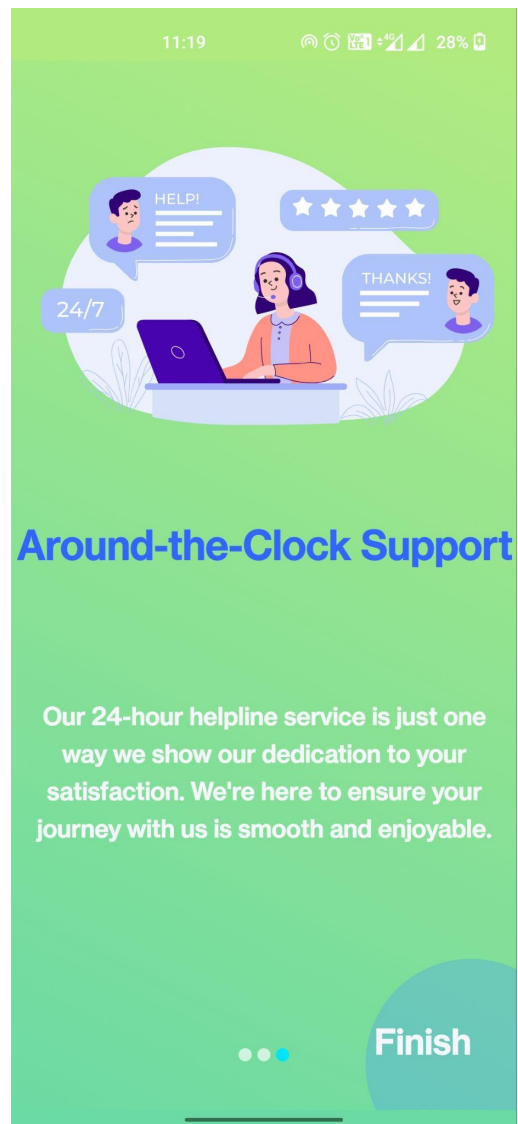
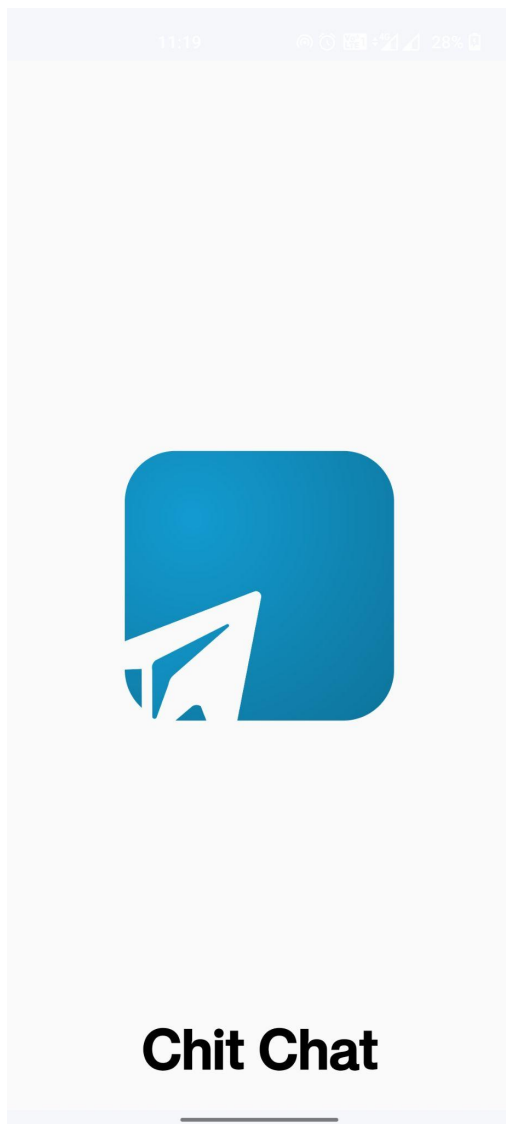
Model Performance Testing:

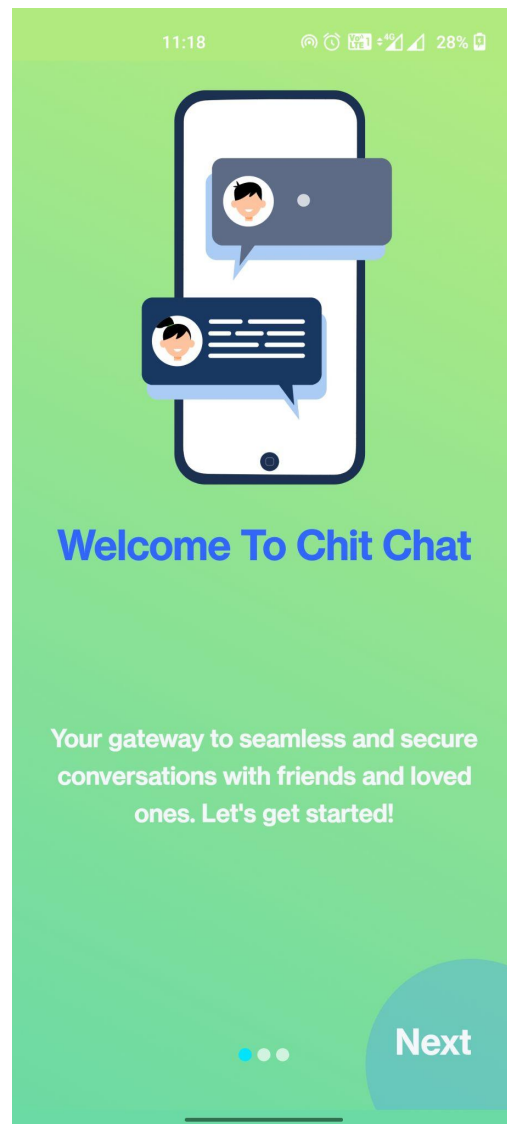
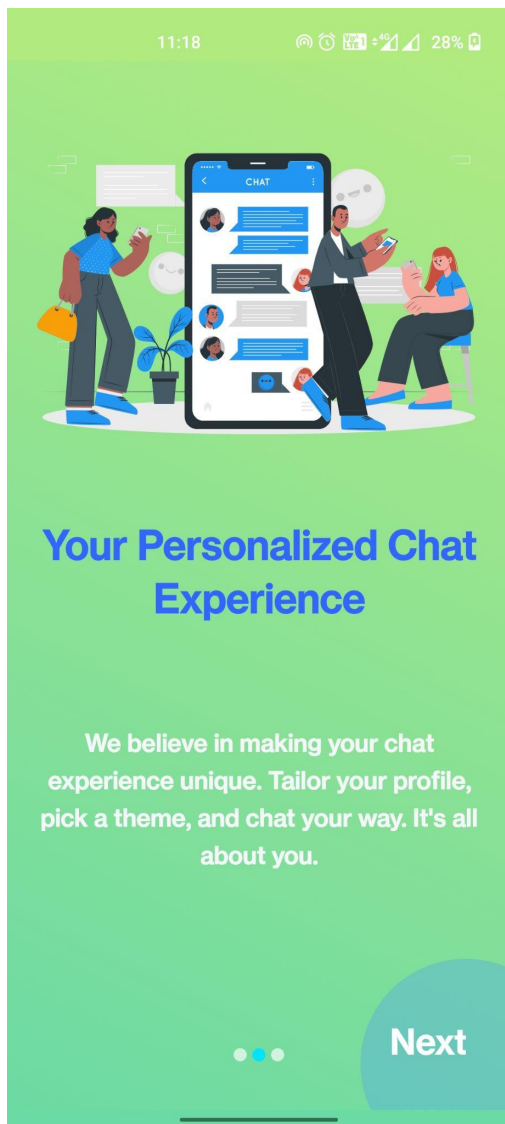
Project team shall fill the following information in model performance testing template.

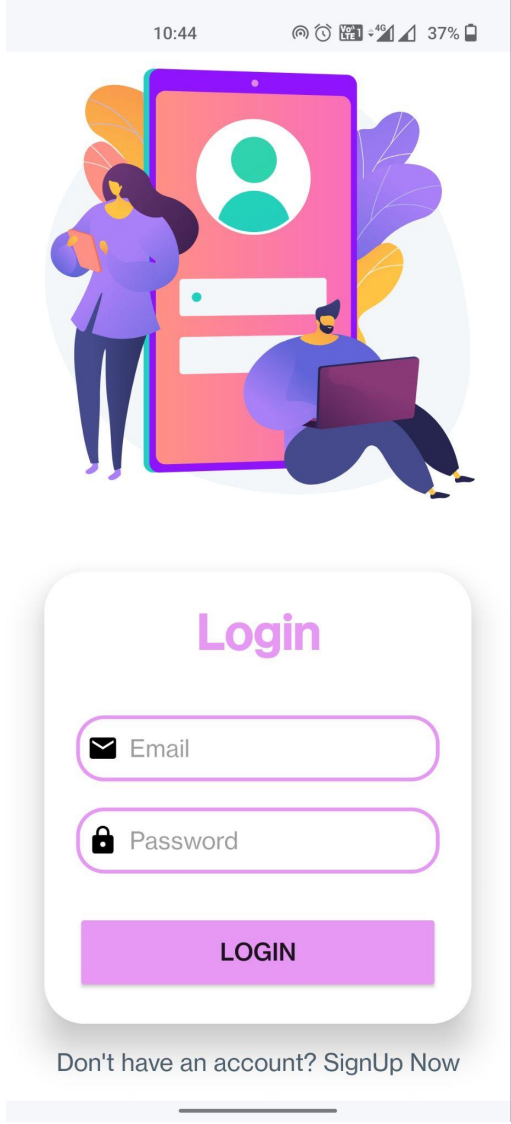
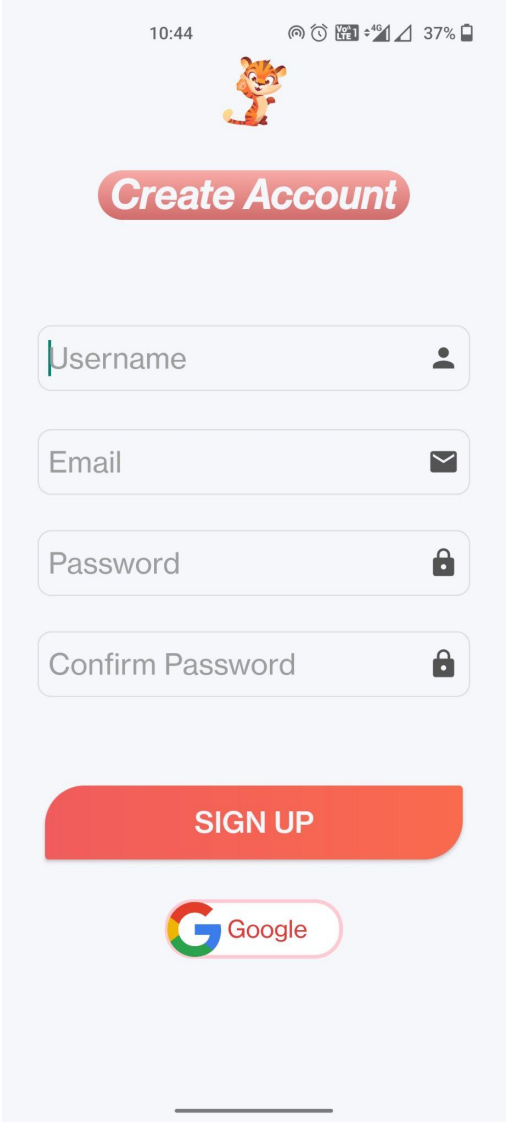
S.No.	Parameter	Values	Screenshot
1.	Metrics	App Launch Time- Screen Render Time- Code Quality-	2 Seconds 2.5 Seconds Good
2.	Usage	App Size- Customer Experience-	

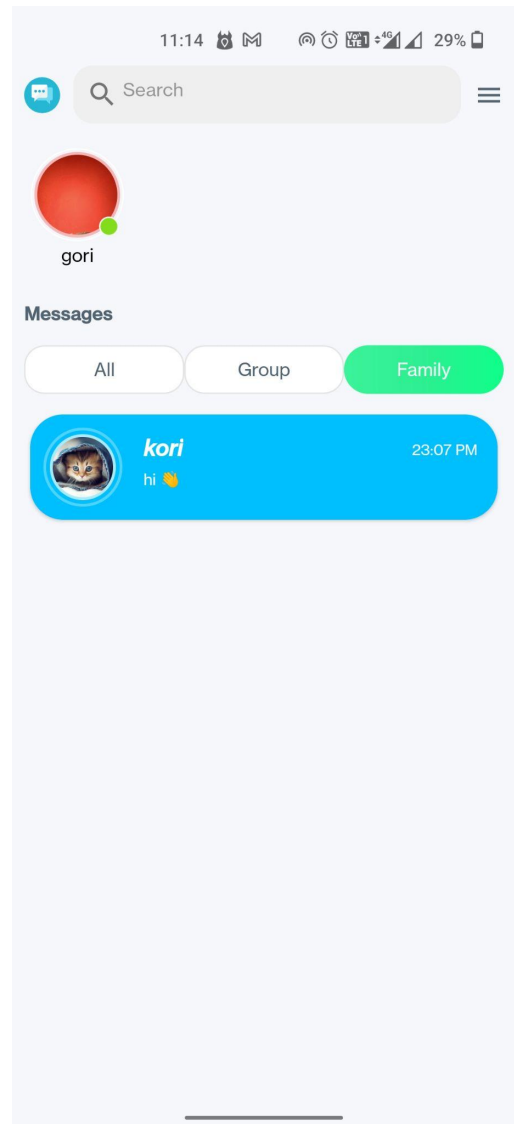
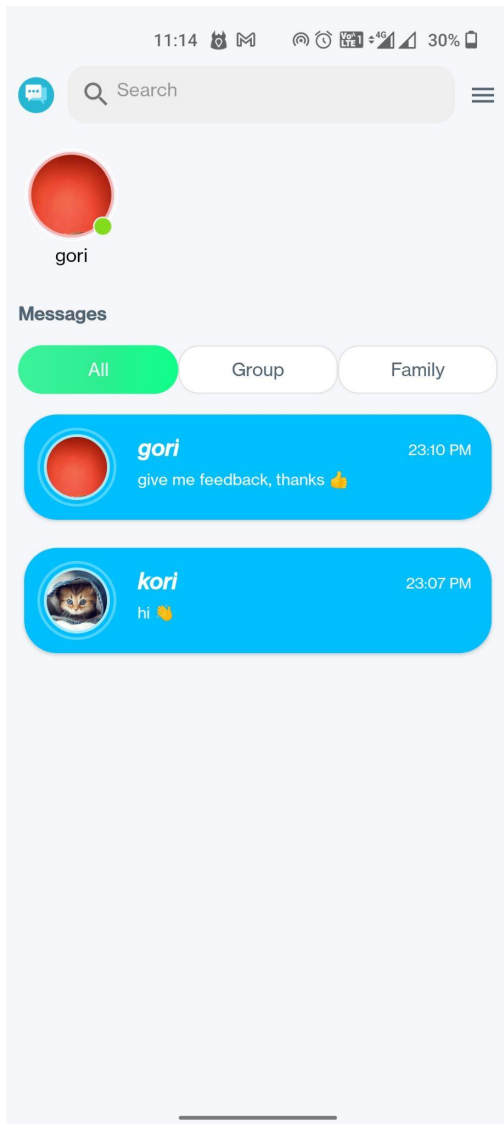
3.	Performance	Error and Crash Rates- Database Query Performance-	<div><div>Rules metrics</div><div><div>Allows [?]</div><div><input checked="" type="checkbox"/> 690 total</div></div><div><div>Denies [?]</div><div><input type="checkbox"/> total</div></div><div><div>Errors [?]</div><div><input type="checkbox"/> total</div></div><div></div></div> <div><div>Load metrics</div><div><div>Load [?]</div><div><1% peak</div></div><div></div></div> <div><div>Bytes stored [?]</div><div><input checked="" type="radio"/> 5.7MB current</div><div><div>Object count [?]</div><div><input type="radio"/> 3 current</div></div><div><div>Bandwidth sent [?]</div><div><input type="radio"/> 5.72MB total</div></div><div><div>Requests [?]</div><div><input type="radio"/> 6 total</div></div><div></div></div>
----	-------------	---	---

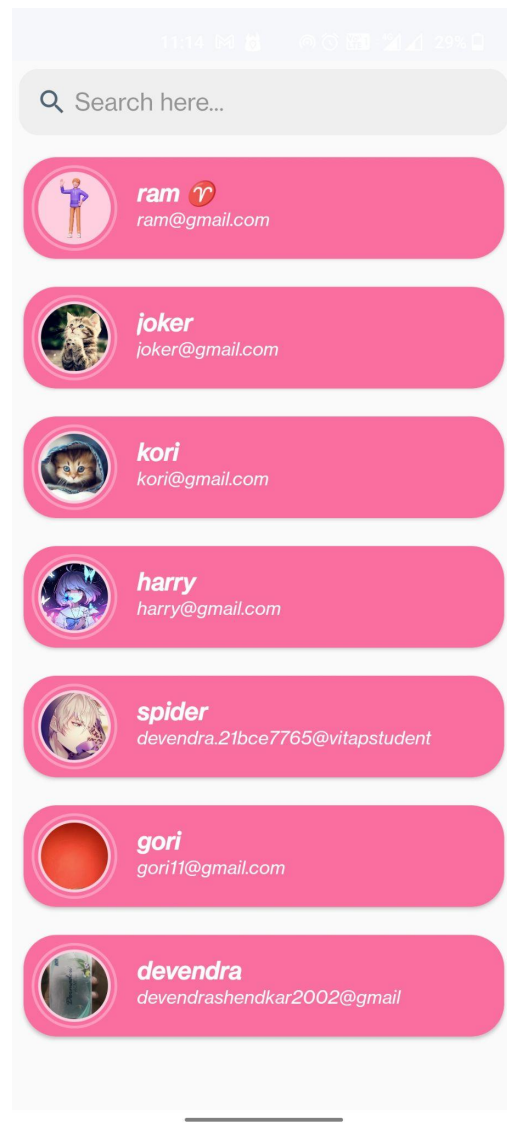
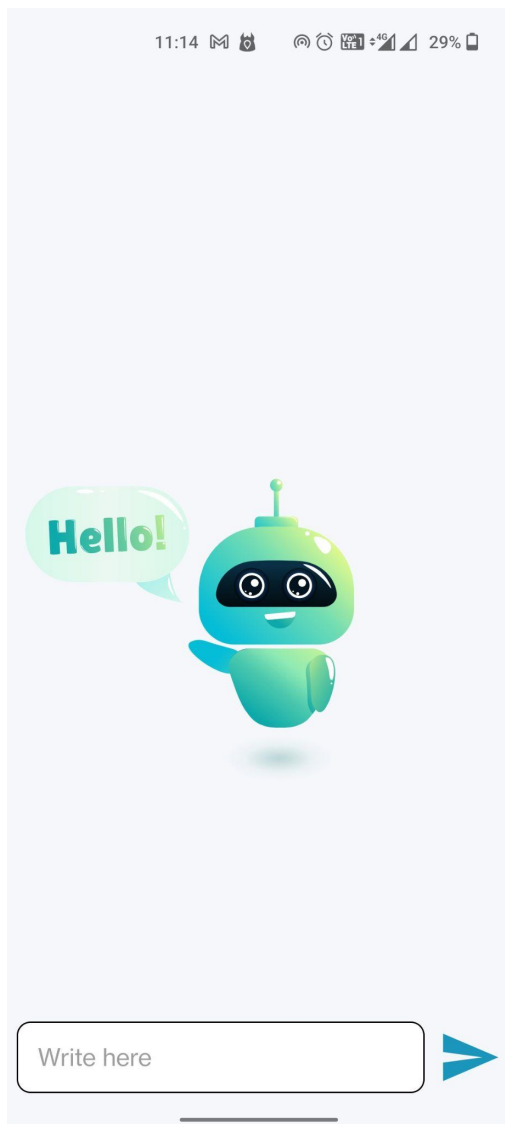
Results

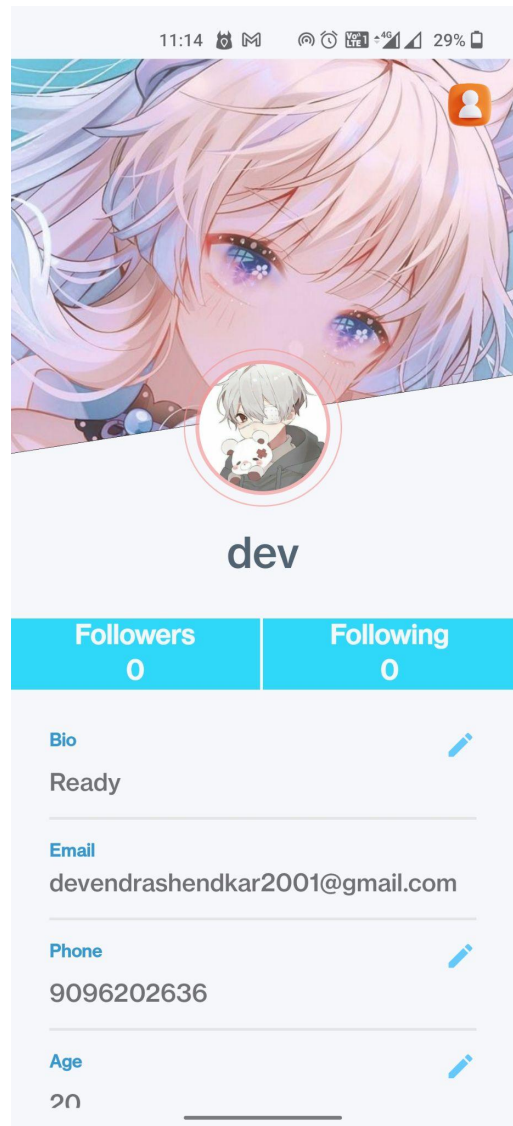
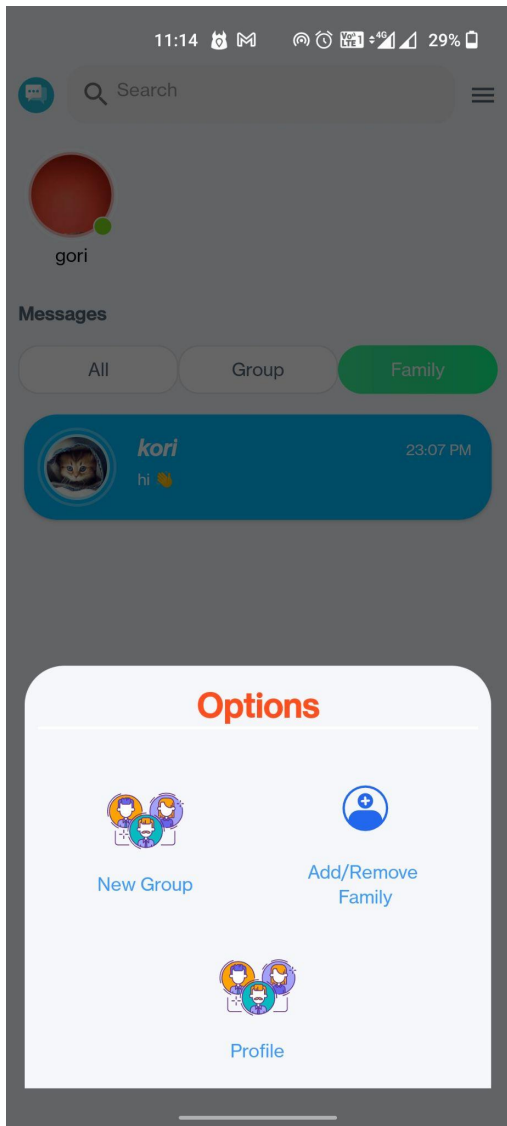


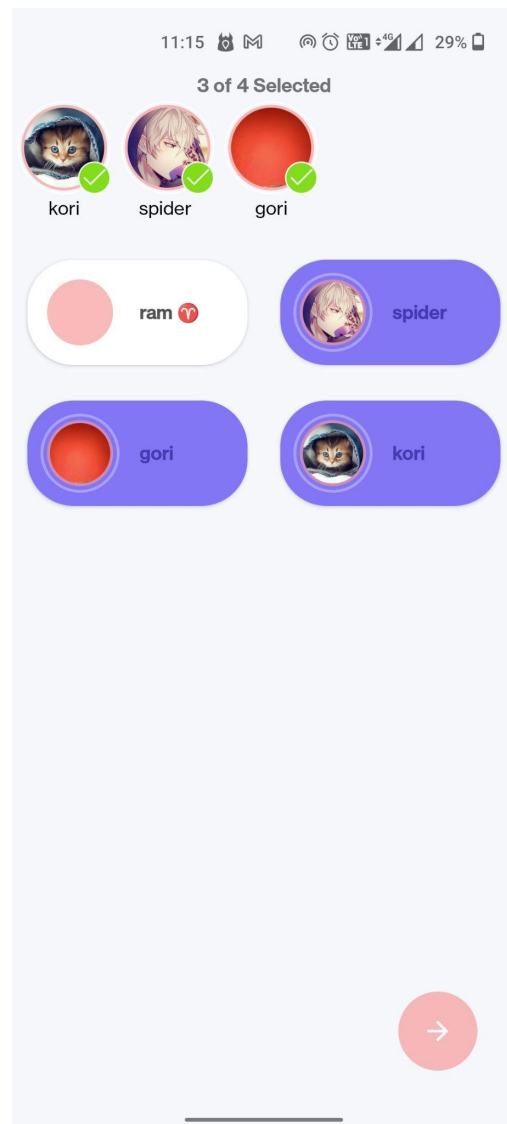
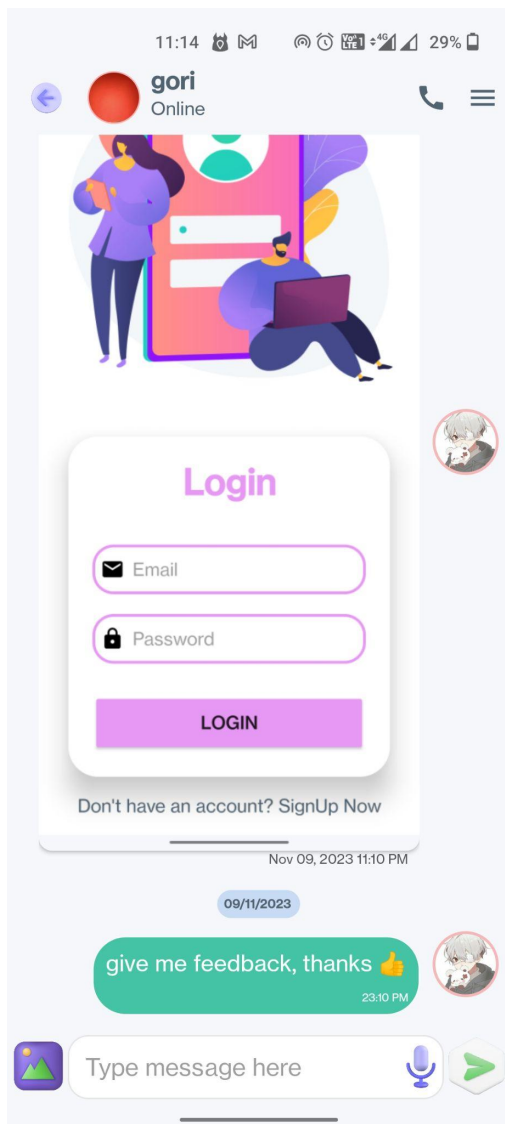














Gaming corner

Members



kori



spider



gori



Advantages and Disadvantages

Advantages:

- Users can enjoy a unified platform that covers various aspects, including communication, social networking, multimedia sharing, and health-related features.
- Integration of health and wellness features, such as exercise videos, meditation, and health-related articles, contributes to users' overall well-being.
- Integration of a GPT-powered chatbot provides users with intelligent and engaging conversational experiences.
- Users can share not only text messages but also photos and videos, creating a more dynamic and expressive communication environment.
- Incorporating offline functionality for exercises, meditation, and content consumption enhances accessibility, particularly in areas with limited internet access.

Disadvantages:

- Integrating a wide range of features may make the development process complex, requiring more resources and effort.
- Handling sensitive health data and personal information requires robust security measures to prevent privacy breaches.
- A multitude of features may lead to a complex user interface, potentially overwhelming some users.
- Features like video posting and multimedia sharing could lead to increased data storage and bandwidth requirements.

- Users accustomed to simpler chat applications may resist adopting a more feature-rich platform.
- Integrating and maintaining various external services (e.g., GPT-powered chatbot, video hosting) may pose technical challenges.

Future Scope

- Continuous improvement and expansion of the GPT-powered chatbot to offer more advanced and personalized conversational experiences.
- Incorporation of gamification elements to encourage users to stay active, meditate regularly, or achieve health-related goals.
- Expansion of social networking features, such as event planning, location-based meetups, or interest-based groups.

- Continuous improvement of security protocols to address evolving privacy concerns and ensure the safety of user data.
- Adapting the application to cater to the cultural preferences and languages of a global audience.
- Implementation of a robust feedback system to collect user insights and preferences for continuous improvement.
- Collaborating with telehealth providers to integrate telehealth services within the app, allowing users to access healthcare professionals seamlessly.

Conclusion

In conclusion, the development of this comprehensive chat application represents a multifaceted solution to address existing challenges and cater to the evolving needs of users. By amalgamating communication, social networking, health and wellness features, and advanced conversational AI, the project aspires to create a holistic user experience. The identification of fragmented user experiences in current chat applications, along with the limited integration of health and wellness functionalities, serves as the impetus for this project. Through careful design and implementation, we aim to bridge these gaps and offer users a seamless platform that not only facilitates communication but also fosters personal well-being. The

integration of a GPT-powered chatbot marks a step toward intelligent and engaging interactions, providing users with information, assistance, and a natural conversational experience. This innovation is pivotal in elevating the user experience and setting the application apart in a competitive landscape. Moreover, the incorporation of features like multimedia sharing, offline functionality, and personalized content recommendations adds depth and versatility to the user experience. These features are intended to adapt to the varied preferences and needs of users, creating a dynamic and user-centric platform. As we move forward, the project's future scope envisions continuous improvement, embracing emerging technologies such as augmented reality, wearable device integration, and advanced security measures. These enhancements are aimed at ensuring the application remains at the forefront of innovation and user satisfaction. In essence, this project is not just a chat application; it is a comprehensive lifestyle platform that strives to enhance the way users communicate, engage with their health and well-being, and find value in a single, integrated space. By addressing existing challenges and anticipating future trends, the project aspires to leave a lasting impact on user experiences and set new standards for versatile and user-friendly applications in the digital landscape.

Appendix

Source Code Link :-

<https://github.com/smartinternz02/SI-GuidedProject-587805-1697028645>

Project Demo Link:-

<https://github.com/smartinternz02/SI-GuidedProject-587805-1697028645>

Note

Source Code is inside Chatting app folder

Project Demo video is inside demo folder