rocket.chat

Embedded Chat 2024

Google Summer of Code 2024 - Project Proposal

About me

- **Name -** Umang Utkarsh
- **Rocket.Chat ID -** umang.utkarsh
- **Email -** umangutkarsh0024@gmail.com
- **Github -** umangutkarsh
- **Linkedin -** umangutkarsh
- **Country -** India
- **Time zone -** UTC +5:30 (IST - India)
- **Institute -** National Institute of Technology, Karnataka (NITK)
- **Company -** Jio Platforms Limited

Introduction

I am Umang Utkarsh, a recent graduate from National Institute of Technology, Karnataka, India, and a Software Development Engineer at Jio Platforms Limited. My journey into tech began with machine learning projects during college, leading me to explore web development and open-source software. I love working on projects that solve real-world problems, and want to grow in the tech community.

Why do I wish to participate in the Google Summer of Code?

Participating in GSoC offers me a unique opportunity to immerse myself in the open-source community and contribute to meaningful projects. It aligns perfectly with my aspiration to enhance my coding skills, learn from experienced developers, and make a significant impact on open-source projects. I'm eager to apply my knowledge about technology to real-world problems, and GSoC provides the perfect platform for me to do so.

Why do I want to apply to RocketChat in particular?

My interest in applying to Rocket.Chat stems from several key factors. The alignment of Rocket.Chat's tech stack with my experience and interests makes it an ideal platform for me to apply my skills and expand my knowledge. The welcoming and supportive nature of the Rocket.Chat community has left a lasting impression on me, fostering a sense of belonging and motivating me to contribute and grow.

My journey with Rocket.Chat has been both enriching and transformative. I've spent considerable time immersing myself in the codebases of various projects, learning from the community, and making meaningful contributions. This experience has not only honed my technical skills but also taught me valuable lessons in teamwork, problem-solving, and effective communication. Beyond the tech stack I initially joined with, I've gained a deeper understanding of Rocket.Chat app development and other related technologies, enhancing my overall skill set.

Regularly attending the weekly app-workshops has been instrumental in staying connected with the community members and keeping abreast of the latest developments. These workshops have provided me with opportunities to learn from others, share my knowledge, and collaborate on projects.

I'm excited about the prospect of further contributing to Rocket.Chat and becoming a more integral part of this incredible community. My goal is to use this opportunity to enhance the platform and contribute to its growth, while also learning and growing alongside the community. Therefore, I have decided to apply only for RocketChat for GSoC 2024.

Project Description

Project Title

Embedded Chat 2024

EmbeddedChat is a customizable, extensible React component module for integrating Rocket.Chat's chat functionality into applications, offering a flexible and user-friendly interface for real-time communication.

Abstract

The EmbeddedChat 2024 project aims to significantly enhance the integration of Rocket.Chat's chat functionality into web and mobile applications, focusing on creating a seamless and customizable chat experience. By leveraging the power of Rocket.Chat's REST and real-time APIs, this project seeks to elevate the chat experience beyond the basic capabilities, introducing advanced features such as enhanced security measures, a headless UI library for greater flexibility, and pre-built templates for quicker development. The project will also focus on making EmbeddedChat more accessible and user-friendly, with a particular emphasis on simplifying the configuration process for both developers and administrators. The ultimate goal is to provide a robust, scalable, and customizable solution that empowers developers to easily integrate real-time chat into their applications, fostering better communication and collaboration among users.

Benefits to the community

The EmbeddedChat 2024 project aims to revolutionize the way developers integrate chat functionality into their applications, offering a comprehensive, customizable, and efficient solution that leverages the power of Rocket.Chat. This project will address the challenges and limitations of traditional chat integration methods, providing a seamless and user-friendly chat experience that is both robust and scalable.

For the community, the EmbeddedChat 2024 project will offer several key advantages:
Simplified Integration: By offering a full-stack React component node module, EmbeddedChat 2024 will significantly simplify the process of integrating chat functionality into web and mobile applications. This will save developers valuable time and resources, allowing them to focus on building their core application features.
Enhanced User Experience: The project will introduce advanced chat features such as enhanced security measures, a headless UI library for greater flexibility, and pre-built templates. These features will ensure a seamless and engaging chat experience for users, improving overall user satisfaction and engagement.
Customization and Flexibility: EmbeddedChat 2024 will provide developers with the flexibility to customize the chat experience to match the look and feel of their application. This includes theming options and the ability to configure various chat functionalities, ensuring that the chat feature is an integral part of the application's identity.

Scalability and Performance: By leveraging Rocket.Chat's robust infrastructure, EmbeddedChat 2024 will offer a scalable and high-performance chat solution. This will ensure that applications can handle a growing number of users without compromising on performance or user experience.
Community Growth and Recognition: The development and adoption of EmbeddedChat 2024 will contribute to the growth and recognition of the Rocket.Chat project, reinforcing its position as a leading open-source chat solution. This will also encourage more developers to explore and contribute to the Rocket.Chat ecosystem, fostering a vibrant and supportive community.

Goals and Deliverables

Rocket.Chat App for Configuration: Develop a Rocket.Chat app to remotely configure EmbeddedChat instances, enhancing user control and customization.
Security Enhancements: Implement security improvements, including shifting from localStorage to browser cookies for authentication, to enhance data security and user privacy.
Headless UI Library: Transform the UI library into a headless component, offering flexibility and customization for developers to design their chat interfaces.
Pre-built Templates: Introduce a variety of pre-built templates for the headless UI library, simplifying the development process and allowing users to quickly integrate EmbeddedChat into their applications.
UI Kit Rendering Improvements: Enhance the rendering of the UI kit inside EmbeddedChat, ensuring a seamless and visually appealing chat experience.
Documentation and Tutorials: Create comprehensive documentation and tutorials for the EmbeddedChat app, facilitating easy adoption and integration by the Rocket.Chat community.
These goals aim to significantly enhance the EmbeddedChat project, making it more secure, customizable, and user-friendly, while also fostering a supportive community around its development.

Rocket.Chat Demo App

I have built a basic rocket.chat app to showcase the use of some basic functionality which can be incorporated into the app, like - ***Notifying the user, direct message to the user, integrating modals/contextual-bars using ui-kit, using the persistence api to store data on the Rocket.Chat server and fetching data from the persistence storage.***

The code of demo app can be accessed here - **BasicDemo-RCApp**
The demo video to showcase the features of the app - **Demo**

Sending message, Notifying user, direct message to the user

The following code snippets demonstrate how various actions such as sending messages, notifying users, and direct messages would be handled.

```
export async function sendMessage(modify: IModify, room: IRoom, sender: IUser,
message: string, blocks?: Array<Block>): Promise<string> {
    const msg = modify.getCreator()
        .startMessage()
        .setSender(sender)
        .setRoom(room)
        .setParseUrls(true)
        .setText(message);

    if (blocks !== undefined) {
        msg.setBlocks(blocks);
    }

    return await modify.getCreator().finish(msg);
}
```

```
export async function sendNotification(modify: IModify, room: IRoom, sender: IUser,
message: string): Promise<void> {
    let msg = modify.getCreator()
        .startMessage()
        .setRoom(room)
        .setText(message);

    return await modify.getNotifier().notifyUser(sender, msg.getMessage());
}
```

```
export async function sendDirectMessage(context: SlashCommandContext, read: IRead,
modify: IModify, message: string): Promise<void> {
    const messageStructure = modify.getCreator().startMessage();
    const sender = context.getSender();
    const appUser = await read.getUserReader().getAppUser();
    if (!appUser) {
        throw new Error(`Error getting the app user`);
    }
    let room = (await getOrCreateDirectRoom(read, modify, [
        sender.username,
        appUser.username,
    ])) as IRoom;

    messageStructure.setRoom(room).setText(message);
    await modify.getCreator().finish(messageStructure);
}
```

App Data Persistence

Some code snippets are shown below to demonstrate the use of persistence storage. We will make use of **`RocketChatAssociationRecord`** and **`RocketChatAssociationModel`** for storing, retrieving and removing the data (news in this case, and other user related information).

```
public async persist(
        room: IRoom,
        id: string,
    ): Promise<boolean> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
            new RocketChatAssociationRecord(RocketChatAssociationModel.ROOM,
room.id),
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC, id),
        ];

        try {
            await this.persistence.updateByAssociations(associations, {id}, true);
        } catch (err) {
            console.warn(err);
            console.log(err);
            return false;
        }
        return true;
    }
```

Store data^

```
public async findAll(): Promise<Array<string>> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
        ];
        let result: Array<string> = [];
        try {
            const records: Array<{id: string}> =
                (await this.persistenceRead.readByAssociations(associations)) as
Array<{id: string}>;

            if (records.length) {
                result = records.map(({id}) => id);
            }
        } catch (err) {
            console.warn(err);
        }

        return result;
    }
```
Get all data^

```
public async findByName(
        room: IRoom,
    ): Promise<Array<string>> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
            new RocketChatAssociationRecord(RocketChatAssociationModel.ROOM,
room.id),
        ];

        let result: Array<string> = [];
        try {
            const records: Array<{id: string}> =
                (await this.persistenceRead.readByAssociations(associations)) as
Array<{id: string}>;

            if (records.length) {
                result = records.map(({id}) => id);
            }
        } catch (err) {
            console.warn(err);
        }
        return result;
    }
```
Get a single desired data^

```
public async removeById(
        id: string,
    ): Promise<boolean> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
            new RocketChatAssociationRecord(RocketChatAssociationModel.ROOM, id),
        ];

        try {
            await this.persistence.removeByAssociations(associations);
        } catch (err) {
            console.warn(err);
            return false;
        }
        return true;
    }
```

Remove a data^

```
public async clear(): Promise<boolean> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
        ];

        try {
            await this.persistence.removeByAssociations(associations);
        } catch (err) {
            console.warn(err);
            return false;
        }

        return true;
    }
```

Clear all data^

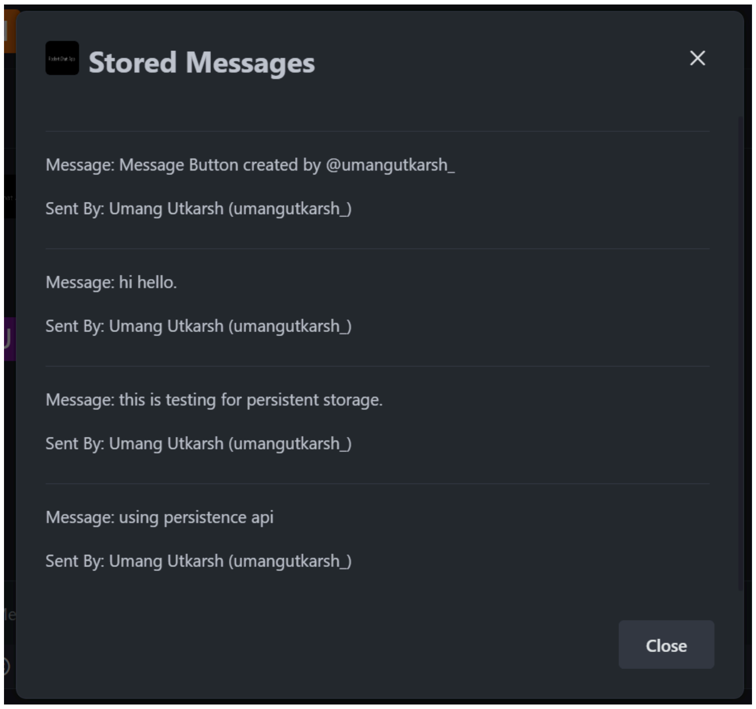
All of these functionality could be accessed through a class name `MessagePersistence`

```
let messageStorage = new MessagePersistence(persistence,
read.getPersistenceReader());
```

OR

```
let messageStorage = new MessagePersistence(persistence,
app.getAccessors().reader.getPersistenceReader());
```

Below is a small demonstration of getting the stored data from persistent storage.


Stored Messages

Message: Message Button created by @umangutkarsh_

Sent By: Umang Utkarsh (umangutkarsh_)

Message: hi hello.

Sent By: Umang Utkarsh (umangutkarsh_)

Message: this is testing for persistent storage.

Sent By: Umang Utkarsh (umangutkarsh_)

Message: using persistence api

Sent By: Umang Utkarsh (umangutkarsh_)

Building UI Blocks

This is demonstrated above in the Ui-Kit section

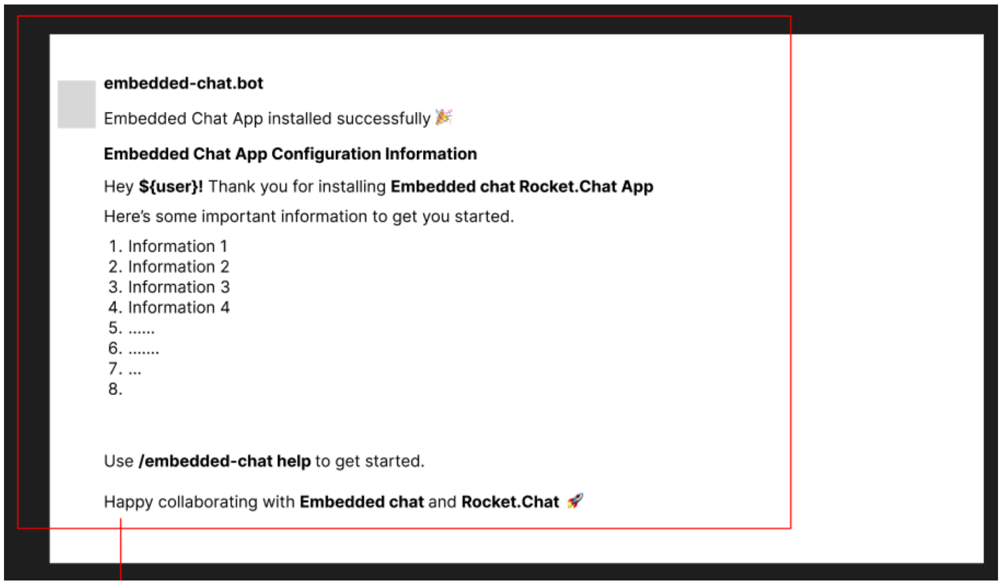
Implementation Details

Embedded Chat 2024 App Flow

To outline the basic flow of the Embedded Chat 2024 app for Rocket.Chat, including the configuration through the app's settings, we'll follow a structured approach that integrates seamlessly with the Rocket.Chat platform, ensuring a user-friendly experience from installation to daily use.

1. **Installation and Initial Setup**

Upon Installation: The app sends a direct message to the admin of the workspace, providing basic information about the app, its features, and instructions on how to configure it using slash commands.


embedded-chat.bot

Embedded Chat App installed successfully 🎉

Embedded Chat App Configuration Information

Hey **${user}!** Thank you for installing **Embedded chat Rocket.Chat App**

Here's some important information to get you started.

1. Information 1
2. Information 2
3. Information 3
4. Information 4
5. ......
6. ....
7. ...
8.

Use **/embedded-chat help** to get started.

Happy collaborating with **Embedded chat** and **Rocket.Chat** 🚀

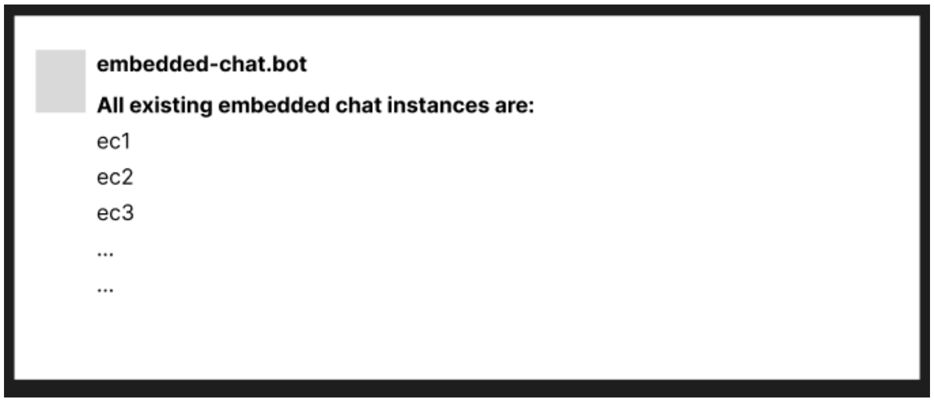
More information about the app would be provided to the user when installed.

```
public async onInstall(context: IAppInstallationContext, read: IRead, http: IHttp,
persistence: IPersistence, modify: IModify): Promise<void> {

    }
```

```
public async initialize(configurationExtend: IConfigurationExtend, environmentRead:
IEnvironmentRead): Promise<void> {

    }
```

2. Slash Commands Overview

/embedded-chat list: Lists all configured EmbeddedChat instances.


embedded-chat.bot

All existing embedded chat instances are:

ec1

ec2

ec3

...

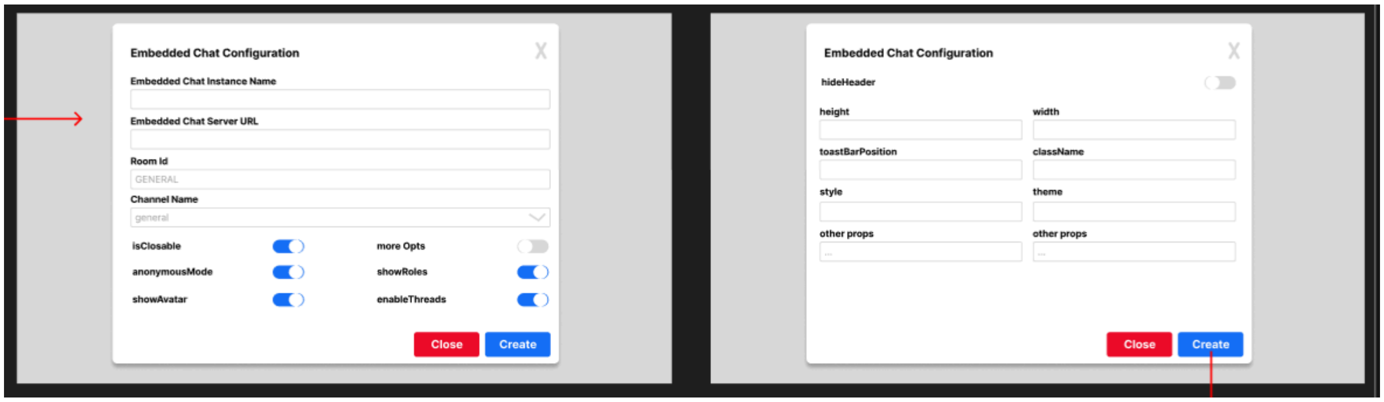
...

/embedded-chat help: Offers information about the app, its features, and how to configure it.

Additional commands can be added or modified as the app progresses.

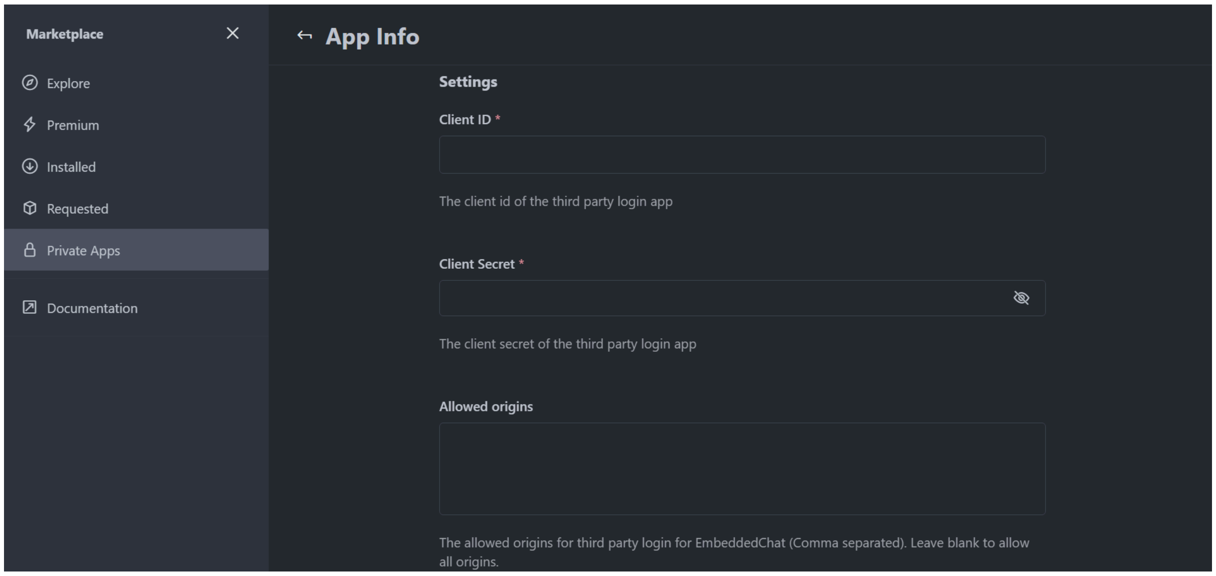
3. Configuring the App

Configuration Modal: After installation, the app requires configuration. Admins can specify secret values and other configurations through a configuration modal.



isClosable: on
anonymousMode: on
showAvatar: on
more Opts: off
showRoles: on
enableThreads: on
hideHeader: off

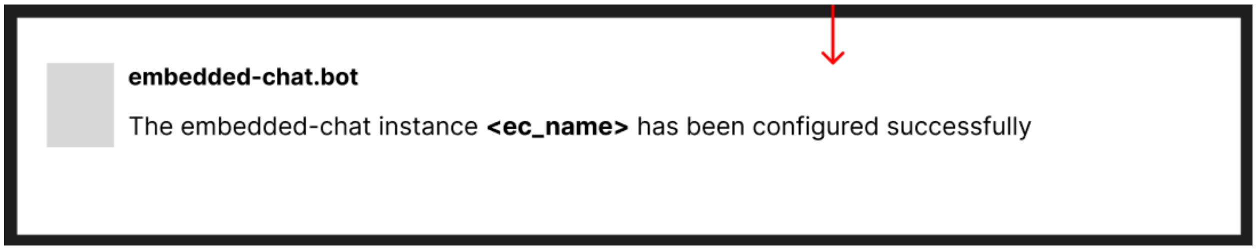
Settings in Rocket.Chat: Admins can also configure the app through the Rocket.Chat interface by navigating to Menu -> Installed -> Private Apps -> EmbeddedChat -> Settings. This provides an additional layer of convenience and control for admins.



The client id of the third party login app

The client secret of the third party login app

The allowed origins for third party login for EmbeddedChat (Comma separated). Leave blank to allow all origins.



embedded-chat.bot

The embedded-chat instance **<ec_name>** has been configured successfully

Security Improvements: Create endpoints to securely store tokens, enhancing the app's security.

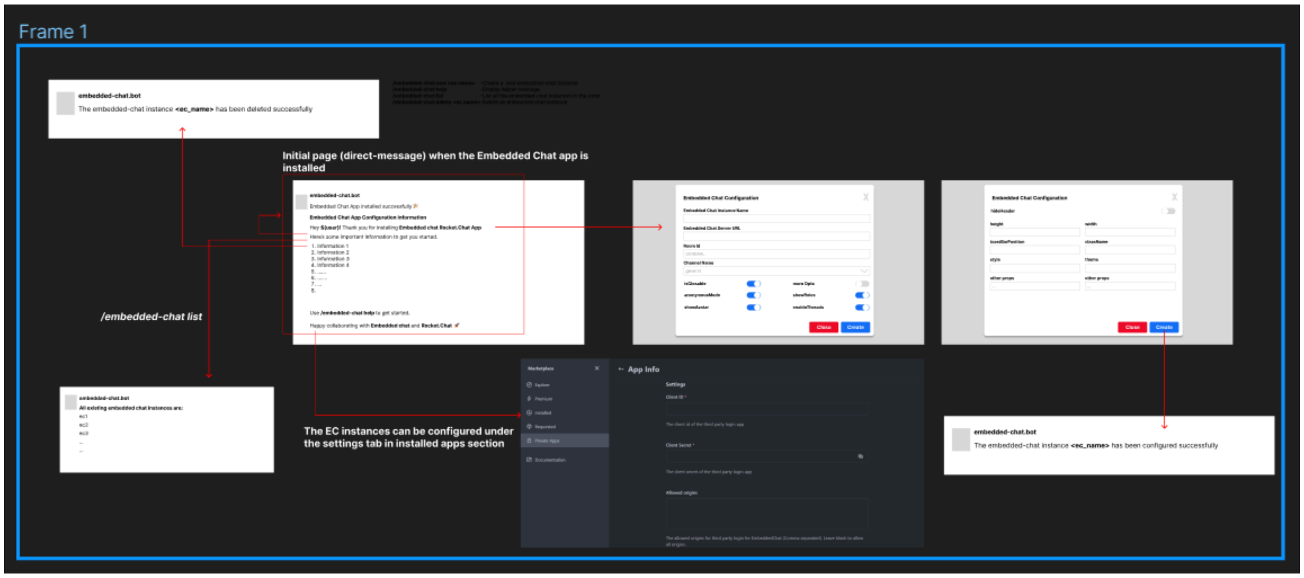
Theming and Pre-built Templates: Introduce a theming layer where admins can configure changes and access pre-built templates for a customized chat experience.

4. User Interaction

Direct Messages for Support: For any queries or assistance, admins can reach out to the app through direct messages, ensuring a smooth and efficient user experience.

5. Design Integration

Figma Designs: The app's user interface and user experience are designed with Figma, aligning with Rocket.Chat's aesthetics.


Frame 1
Initial page (direct-message) when the Embedded Chat app is installed
/embedded-chat list
The EC instances can be configured under the settings tab in installed apps section
Embedded Chat Configuration
Close
Create
Marketplace
App Info
Settings
Client ID *
Client Secret *
Allowed origins

embedded-chat.bot
The embedded-chat instance **<ec_name>** has been deleted successfully

embedded-chat.bot
Embedded Chat App installed successfully 🎉
Embedded Chat App Configuration Information
Hey **${user}!** Thank you for installing **Embedded chat Rocket.Chat App**
Here's some important information to get you started.
1. Information 1
2. Information 2
3. Information 3
4. Information 4
5. ......
6. ......
7. ....
8.

Use **/embedded-chat help** to get started.
Happy collaborating with **Embedded chat** and **Rocket.Chat** 🚀

embedded-chat.bot
All existing embedded chat instances are:
ec1
ec2
ec3
...
...

embedded-chat.bot
The embedded-chat instance **<ec_name>** has been configured successfully

The basic flow of the Embedded Chat 2024 app through Figma design can be accessed here - **[Embedded Chat 2024 Figma design]**

This flow ensures that the Embedded Chat 2024 app for Rocket.Chat is not only functional but also user-friendly, providing a seamless experience from installation to daily use, with the added convenience of configuration through the Rocket.Chat interface.

The flow can be modified as the app development progresses.

Creation of Endpoints

1. Configure EmbeddedChat props

```
/props-endpoint/:id
```

- The dynamic ***id*** can be passed from Embedded Chat, so that this endpoint gives the list of all the props (or a json) instead of passing all the props in the main embedded-chat component. (i.e., - ***isClosable, theme, host, roomId, etc.***). These props can then be used to populate an embedded chat instance.

```
export class PropsEndpoint extends ApiEndpoint {
    public path = "props";

    public async get(
        request: IApiRequest,
        endpoint: IApiEndpointInfo,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persis: IPersistence,
    ): Promise<IApiResponse> {
        // Some logic to fetch the props to populate EC
        return this.json({
            status: 200,
            headers: {
                'Content-Type': "application/json",
            },
            content: await getProps(),
        });
    }
}
```

- To pass the props in the ***url*** of embedded chat, some sort of like ***query-params***

```
/chat.${url-of-embedded-chat}/?isClosable=${value}&host=${someHost}&roomId={id}&....
```

2. **To support cookie-based authentication instead of storing the token in localStorage**

```
export class AuthEndpoint extends ApiEndpoint {
    public path = "auth";

    public async post(
        request: IApiRequest,
        endpoint: IApiEndpointInfo,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persis: IPersistence,
    ): Promise<IApiResponse> {
        const code = request?.query?.content;
        const userOrEmail = request?.user?.username;
        const password = request?.content; // fetch password from request;
        let reqBody;
        if (!code) {
            reqBody = `{ "user": "${userOrEmail}", "password": "${password}"}`;
        } else {
            reqBody = `{"user": "${userOrEmail}","password": "${password}","code":
"${code}"}`;
        }
        try {
            const response = await http.post(`${someHost}/api/v1/login`, {
                headers: {
                    'Content-Type': 'application/json',
                },
                content: reqBody.toString(),
            });

            if (response.statusCode !== 200) {
                return {
                    status: response.statusCode,
                    content: 'Not authorized',
                };
            }

            return {
                status: HttpStatusCode.OK,
                content: response,
            }
        } catch (err) {
            this.app.getLogger().info('Error: ', err);
            console.log('Error: ', err);
            return this.json({
                status: HttpStatusCode.NOT_FOUND
            })
        }

    }
}
```

```
document.cookie = 'name=John; expires=' + new Date(9999, 0, 1).toUTCString();
```

Something like this can be used, or some library to manage cookies like localStorage, for example *'js-cookie'*.

User-Interface Library

Currently, the Embedded Chat uses component style overrides which get overridden at the component level to provide theming. Every component has also *classNames* added so that customization can be done using css style sheet, or other also.
I am planning to extend the theming, to have sort of a layer where users can do some changes and have some grid looking Embedded Chat instances. Template ideas can be taken from ***element by matrix***. Which is also an in-app chat solution and has a delivered UI.
Some pre-built templates will be provided to the user to quickly get started with Embedded Chat.

Further customization of the templates could be done through ***EmbeddedChatTheme.stories.js***, which takes a theme object for customizations.

```
import { EmbeddedChat } from '..';

// More on how to set up stories at:
https://storybook.js.org/docs/react/writing-stories/introduction
export default {
  title: 'EmbeddedChat/Theme',
  component: EmbeddedChat,
};

// More on writing stories with args:
https://storybook.js.org/docs/react/writing-stories/args
export const WithTheme = {
  args: {
    host: process.env.STORYBOOK_RC_HOST || 'http://localhost:3000',
    roomId: 'GENERAL',
    GOOGLE_CLIENT_ID: '',
    isClosable: true,
    setClosableState: true,
    moreOpts: true,
    channelName: 'general',
    anonymousMode: true,
    headerColor: 'white',
    toastBarPosition: 'bottom right',
    showRoles: true,
    showAvatar: false,
    enableThreads: true,
    theme: {
      breakpoints: {
        xs: 0,
        sm: 600,
        md: 900,
        lg: 1200,
        xl: 1536,
      },
      components: {
        ChatInput: {
          styleOverrides: {
            fontWeight: 400,
            color: 'gray',
            border: '1px solid black',
          },
        },
        Message: {
          classNames: 'myCustomClass',
        },
        Box: {
          styleOverrides: {
            background: '#f1f1f1',
          },
          classNames: 'custom-box',
        },
      },
      palette: {
        mode: 'light',
        primary: {
          main: '#007FFF',
          light: '#66B2FF',
          dark: '#0059B2',
          contrastText: '#fff',
        },
        divider: '#E7EBF0',
        text: {
          primary: '#1A2027',
          secondary: '#3E5060',
        },
        grey: {
          main: '#E7EBF0',
          contrastText: '#6F7E8C',
        },
        error: {
          main: '#EB0014',
          light: '#FF99A2',
          dark: '#C70011',
          contrastText: '#fff',
        },
        success: {
          main: '#1AA251',
          light: '#6AE79C',
          dark: '#1AA251',
          contrastText: '#fff',
        },
        warning: {
          main: '#DEA500',
          light: '#FFDC48',
          dark: '#AB6800',
          contrastText: 'rgba(0, 0, 0, 0.87)',
        },
        secondary: {
          main: '#9c27b0',
          light: '#ba68c8',
          dark: '#7b1fa2',
          contrastText: '#fff',
        },
        info: {
          main: '#0288d1',
          light: '#03a9f4',
          dark: '#01579b',
          contrastText: '#fff',
        },
        background: {
          surface: '#fff',
          default: '#fff',
          modal: '#fff',
        },
      },
      typography: {
        default: {
          fontFamily:
            '"IBM Plex Sans",-apple-system,BlinkMacSystemFont,"Segoe
UI",Roboto,"Helvetca Neue",Arial,sans-serif,"Apple Color Emoji","Segoe UI
Emoji","Segoe UI Symbol"',
          fontSize: 14,
          fontWeightLight: 300,
          fontWeightRegular: 400,
          fontWeightMedium: 500,
          fontWeightBold: 700,
        },
        h1: {
          fontSize: 'clamp(2.625rem, 1.2857rem + 3.5714vw, 4rem)',
          fontWeight: 800,
          lineHeight: 1.1142857142857143,
          color: '#0A1929',
        },
        h2: {
          fontSize: 'clamp(1.5rem, 0.9643rem + 1.4286vw, 2.25rem)',
          fontWeight: 800,
          lineHeight: 1.2222222222222223,
          color: '#132F4C',
        },
        h3: {
          fontSize: '2.25rem',
          lineHeight: 1.2222222222222223,
          fontWeight: 400,
        },
        h4: {
          fontSize: '1.75rem',
          lineHeight: 1.5,
          fontWeight: 400,
        },
        h5: {
          fontSize: '1.5rem',
          lineHeight: 1.5,
          color: '#007FFF',
          fontWeight: 400,
        },
        h6: {
          fontSize: '1.25rem',
          lineHeight: 1.5,
          fontWeight: 500,
        },
        button: {
          fontWeight: 700,
          fontSize: '0.875rem',
          lineHeight: 1.75,
        },
      },
      shadows: [
        'none',
        '0px 2px 1px -1px rgba(0,0,0,0.2),0px 1px 1px 0px rgba(0,0,0,0.14),0px 1px 
3px 0px rgba(0,0,0,0.12)',
        '0px 3px 1px -2px rgba(0,0,0,0.2),0px 2px 2px 0px rgba(0,0,0,0.14),0px 1px 
5px 0px rgba(0,0,0,0.12)',
        '0px 3px 3px -2px rgba(0,0,0,0.2),0px 3px 4px 0px rgba(0,0,0,0.14),0px 1px 
8px 0px rgba(0,0,0,0.12)',
        '0px 2px 4px -1px rgba(0,0,0,0.2),0px 4px 5px 0px rgba(0,0,0,0.14),0px 1px 
10px 0px rgba(0,0,0,0.12)',
        '0px 3px 5px -1px rgba(0,0,0,0.2),0px 5px 8px 0px rgba(0,0,0,0.14),0px 1px 
14px 0px rgba(0,0,0,0.12)',
        '0px 3px 5px -1px rgba(0,0,0,0.2),0px 6px 10px 0px rgba(0,0,0,0.14),0px 1px 
18px 0px rgba(0,0,0,0.12)',
        '0px 4px 5px -2px rgba(0,0,0,0.2),0px 7px 10px 1px rgba(0,0,0,0.14),0px 2px 
16px 1px rgba(0,0,0,0.12)',
      ],
      zIndex: {
        header: 1100,
        popup: 1200,
        modal: 1300,
        snackbar: 1400,
        tooltip: 1500,
      },
    },
  },
};
```

A demonstration can be viewed here - **theming**

In this way, custom themes of various components can be pre-built. For this, the components need to be separated out. For example:: A ***`<Button />`*** *used in* ***`<ChatHeader />`*** *might have a different theme than the one used in* ***`<ChatInput />`****.*

Making components headless

To make the components of Embedded chat headless, we can have the functionalities provided as hooks, and then some context provider will provide those functionality to other components. Basically, separate the functional components to hooks and keep the UI separate.

For the UI also, the components need to be classified into two -

1. **Basic Components -** Like **`<Button />`**, **`<InputElement />`**, **`<Box />`**.
2. **Complex Components -** Using these basic components, like **`<ChatHeader />`**, **`<ChatInput />`**, **`<Modal />`**, **`<Sidebar />`**.

First priority would be to completely separate out those components which don't have any logic with them related to the app. Secondly, the components which have some functionality will be provided as hooks.

For example RCInstance functionality provided by <RCInstanceProvider />

```
const RCContext = createContext();

export const RCInstanceProvider = RCContext.Provider;

/**
 * @typedef {Object} ECOptions
 * @property {boolean} enableThreads
 * @property {string} authFlow
 * @property {string} width
 * @property {string} height
 * @property {string} host
 * @property {string} roomId
 * @property {string} channelName
 * @property {boolean} showRoles
 * @property {boolean} showAvatar
 * @property {boolean} hideHeader
 * @property {boolean} anonymousMode
 *
 * @typedef {Object} RCContext
 * @property {import('@embeddedchat/api').EmbeddedChatApi} RCInstance
 * @property {ECOptions} ECOptions
 * @returns {RCContext}
 */
export const useRCContext = () => useContext(RCContext);

export default RCContext;
```

```
<RCInstanceProvider value={RCContextValue}>
      <Components />
</RCInstanceProvider>
```

Making the UI library completely headless means that, if a user wants a very basic component then he/she can just use the templates. For example - If a user doesn't want the bottom tab to have bold, italic options, but just the message-box. So the particular template would be used, and that will remove those particular options.

Using Ui-Kit

Various blocks from the *@rocket.chat/ui-kit* will be used to integrate buttons and modals for the *Rocket.Chat* app.
Following are some demonstrations:

```
export function getInputBox(
    labelText: string,
    placeholderText: string,
    blockId: string,
    actionId: string,
    appId: string,
    intitialValue?: string,
    multiline?: boolean,
) {
    const block: InputBlock = {
        type: "input",
        label: {
            type: "plain_text",
            text: labelText,
            emoji: true,
        },
        element: {
            type: "plain_text_input",
            placeholder: {
                type: "plain_text",
                text: placeholderText,
                emoji: true,
            },
            appId,
            blockId,
            actionId,
            initialValue: intitialValue,
            multiline: multiline
        },
    };
    return block;
}
```

```
export function getStaticSelectElement(
    placeholder: string,
    options: Array<Option>,
    appId: string,
    blockId: string,
    actionId: string,
    initialValue?: Option['value'],
) {
    const block: StaticSelectElement = {
        type: "static_select",
        placeholder: {
            type: "plain_text",
            text: placeholder,
            emoji: true,
        },
        options: options,
        appId,
        blockId,
        actionId,
        initialValue,
    };
    return block;
}
```

Project Demo Link

The following is a link to project demo - **Link**

Workflow Timeline

Application Review Period (April 2 - May 1)

During this, my focus will be on:

1. Understanding best practices for app development with Rocket.Chat.
2. Leveraging TypeScript for the EmbeddedChat app.
3. Documenting key insights, best practices, and potential challenges for the EmbeddedChat project.
4. Actively participating in the Rocket.Chat community, sharing insights, and assisting others.
5. Contributing to other Rocket.Chat projects.

Community Bonding Period (May 1 - May 26)

During this, my focus will be on:

1. Establishing regular communication with my mentor to discuss challenges and strategies.
2. Deepening engagement with the Rocket.Chat community to understand future plans.
3. Familiarizing myself with Rocket.Chat documentation for app development.
4. Deciding on the frequency of communication with my mentor for updates and feedback.

Week 1 (May 27 - June 2)

- Setting up the basic structure of the Embedded Chat Rocket.Chat app, including initialization and preparing for further development.
- Work on building basic blocks for the app.

Week 2 (June 3 - June 9)

- Research ways to fetch *rc_uid, rc_token* and fetch them to remove them from localStorage dependency, and use cookies based authentication.

Week 3 (June 10 - June 16)

- Research if url-params or webhooks would be more beneficial in this case (for the managed configured props), and implement the desired one.
- Work more on webhooks.

Week 4 (June 17 - June 23)

- Work on using ec-app url, instead of the */api/v1/login,* where the user will hit. And after that the login api(*/api/v1/login*) will be hit.
- Store the user credentials of the user, though this the details of the user is sent.

Week 5 (June 24 - June 30)

- Work on passing other endpoints through this, since they would require *rc_token.* (if feasible)
- Work on getting your own token (by getting the rc_token and hash it or encode or do some processing based on requirement), and store this as ec_token. (if feasible).

Week 6 (July 1 - July 7)

- Work on getting an intermediate endpoint, where auth is handled.
- If required, work on the APIs in the embedded chat app.

Week 7 (July 8 - July 14 - Midterm Evaluations)

- Will keep this week as a buffer since it will include the mid-term evaluations.
- Will work on some of the feedback received.
- Will complete any incomplete work (if any) left from the previous weeks

Week 8 (July 15 - July 21)

- Analyze if the proposed auth works. If the token is fetched, then one hit to the url will fetch the *rc_token* and *rc_uid*.

- Start to work on custom theming.

Week 9 (July 22 - July 28)

- Work on separating out the components by making them headless (have the functional part separate from the UI)

Week 10 (July 29 - August 4)

- Work on building pre-defined templates for users to get started with embedded chat.

Week 11, 12 (August 5 - August 11, August 12 - August 18)

- Integrate the theming with the app and provide an option to have pre-built themes applied using *styleOverrides* and *theme{}* objects.

Week 13 (July 19 - August 25 - Final product and Evaluation submission period)

- Will keep this week as a buffer to work on the feedback received from my mentor's evaluation.
- Will complete any incomplete work (if left) from previous weeks.

Future Development

I would love to keep contributing to RocketChat projects and be an active member of the community. With that being said, I plan to add the following if I complete the required deliverables before the timeline. If not then I will definitely work on these after GSoC tenure ends.

1. Quote Message (Reply to Message) feature

I was working on this feature before when I was contributing, so I will continue to integrate this and improve this.

2. Work on improving React Native

I am also planning to improve the react-native repo of embedded chat and so that embedded chat can be accessed on mobiles.

Relevant Experiences

I have been consistently contributing to Rocket.Chat and specifically Embedded Chat the most, and engaging with community members for the past few months now, and so I have become attached with Rocket.Chat, and its projects now. My contributions include bug-fixes, adding features, raising issues, and helping out other community members.

I am also among the top contributors in the **Rocket.Chat GSoC 2024 leaderboard**. The GSoC leaderboard could be accessed at https://gsoc.rocket.chat.

A list of my contributions include:

Pull Requests:

1. [Merged] **EmbeddedChat #253** - fix: #148-Improvements on SearchMessage.js
2. [Merged] **EmbeddedChat #262** - Fix/#238 replace message emoji
3. [Merged] **EmbeddedChat #277** - Feat/#274 replace message components with custom components
4. [Merged] **EmbeddedChat #335** - Fix/#334 source image fix
5. [Merged] **EmbeddedChat #339** - Fix/#338 menu closing issue
6. [Merged] **EmbeddedChat #342** - Feat/#341 Delete Confirmation Modal
7. [Merged] **EmbeddedChat #354** - Fix/#353 Lazy Load Fix
8. [Merged] **EmbeddedChat #360** - Fix/#358 drag drop upload error handling
9. [Merged] **EmbeddedChat #362** - Warning 3 Resolved
10. [Merged] **EmbeddedChat #363** - Warning 4 Resolved
11. [Merged] **EmbeddedChat #364** - Warning 5,6 Resolved
12. [Merged] **EmbeddedChat #367** - Typing-text fixed
13. [Merged] **EmbeddedChat #370** - Fix/#166 scroll (focus) to new message
14. [Merged] **EmbeddedChat #390** - Fix/#389 enhance drop box
15. [Merged] **EmbeddedChat #445** - Feat/#414 message notification
16. [Merged] **EmbeddedChat #475** - [Fix/#457]: Invite link appearance fixed/Copy to clipboard button added/Improvements in the slash command panel UI/Bug-fixes
17. [Merged] **EmbeddedChat #476** - Fix/#468 user mention fixed
18. [Merged] **EmbeddedChat #491** - Feat/#485 files menu option - sidebar

19. [Merged] **Apps.Whiteboard #56** - Fix/#55 settings dropdown issue

20. [Merged] **google-summer-of-code #2** - removed-listing-twice

21. [Open] **EmbeddedChat #263** - Feat/#120 dark mode
22. [Open] **EmbeddedChat #264** - fix/#132(task-9) - Refactor-useEffects
23. [Open] **EmbeddedChat #279** - Revert "Revert "Feat/#274 replace message components with custom components""
24. [Open] **EmbeddedChat #293** - Feat/#237 css to emotion styling
25. [Open] **EmbeddedChat #333** - Feat/#332 ui-kit block element renders
26. [Open] **EmbeddedChat #347** - fix/#336-Button-Misalignment
27. [Open] **EmbeddedChat #381** - Typing-display-fix
28. [Open] **EmbeddedChat #422** - Feat/draft quote message
29. [Open] **EmbeddedChat #453** - Resolve merge conflicts

30. [Open] **RC4Community #229** - fix/#228: fixed-marginBottom-footer
31. [Open] **RC4Community #231** - Fix/#230: fixed-logo-redirection
32. [Open] **RC4Community #233** - Feat/#232: style navbar items

33. [Open] **Apps.Whiteboard #77** - remove-imports

34. [Open] **Docker.Official.Image #203** - update-readme

Below I have listed all the open and merged Pull Requests by me -

1. **Open**
2. **Merged**

I have also opened several issues out which some have been fixed and some will be fixed, or under review. All of my issues could be found here -

1. **Open**

2. Closed

Projects I have worked on

1. DataCenter as a Service - WebApp

Description: This is one of my on-going projects at Jio Platforms Limited (an Indian Technology company), where I work as a Software Developer.
Techstack used: Next.js, Redux-toolkit, Golang (Gin framework), REST APIs, MongoDB aggregation pipelines

2. UrbanShop-The_eCommerce_app

Description: This project is part of my MERN Stack from scratch - The eCommerce Platform. It is a full-featured app with shopping cart and PayPal & credit/debit payments.
Techstack used: HTML, CSS, JavaScript, React, Nodejs, MongoDB, Bootstrap, Redux-toolkit
Project Link: https://github.com/umangutkarsh/UrbanShop-The_eCommerce_app

3. JobLabs

Description: Job Portal built using MERN stack. Vite is used instead of the create-react-app method for creating the project.
Techstack used: HTML, CSS, JavaScript, React, Nodejs, MongoDB, Styled Components
Project Link: https://github.com/umangutkarsh/JobLabs

4. Recipe-Book

Description: The Recipe Book is a web application built using Angular for the frontend client and Firebase as a complete backend solution.
Techstack used: HTML, CSS, JavaScript/TypeScript, Angular, Bootstrap, Firebase
Project Link: https://github.com/umangutkarsh/recipe-book

Why do I find myself suited for this project?

Over the past few months, I've dedicated my time to contributing to Embedded Chat, and it has been an immensely rewarding experience. Engaging with the talented individuals within this community has not only fostered new friendships but also provided me with invaluable opportunities for personal and professional growth.

Despite coming from a non-technical background, my journey into the tech domain has been one of independent exploration and learning. I firmly believe that Rocket.Chat offers an ideal platform for continuous learning and development. Moreover, being a part of such an incredible open-source software community presents boundless opportunities for both contribution and enrichment.

I am deeply committed to contributing my part to this community and I am eager to further engage in its collaborative endeavors.

As one of the top contributors on the Rocket.Chat GSoC 2024 leaderboard, I've gained a deep understanding of the platform. Through hours of dedicated effort, I've honed my skills by delving into its intricacies, hacking, and debugging to solve challenges. Specifically, I've explored various Rocket.Chat apps like **EmbeddedChat**, **Apps.Whiteboard**, **Apps.Notion**, **Apps.Github22**, **Rocket.Chat.Demo.App**, and **Apps.RocketChat.Tester**, comprehending their implementations in depth.

My affinity for Rocket.Chat app development has only grown stronger over time, motivating me to continue enhancing it. Through several months of consistent contributions, I've become intimately familiar with Rocket.Chat's API and SDK. I've extensively studied the **Rocket.Chat developer documentation**, including the **Rocket.Chat Apps TypeScript Definition**, rocket.chat apps-cli, and apps-engine, and also some app development tutorials on the **Rocket.Chat youtube channel**. This immersion has equipped me with the knowledge and skills necessary to drive meaningful improvements within the Rocket.Chat ecosystem.

Working as a software developer at an Indian technology company, I am accustomed to working under tight deadlines and meeting project requirements on time. This skill set, coupled with my technical expertise and experience with Rocket.Chat, positions me as a well-suited candidate to further advance the platform through this project.

Time Availability

(How much time would I be able to dedicate to this project ?)

I can commit 3-4 hours per day on weekdays and 7-8 hours per day on weekends (Saturday, Sunday), totaling approximately 36 hours per week. Given that I don't have any significant commitments apart from my full-time job and I'm at the early stages of my career with minimal job-related workload, I can dedicate ample time to this project. Over the 13-week GSoC period, I estimate I'll be able to invest approximately 450 hours into the project.

This timeframe allows me flexibility to accommodate unforeseen circumstances and take a few days off if needed. Typically, medium projects require around 175 hours to complete, so I believe I'll have sufficient time to fulfill project requirements and even handle potential setbacks.

During the GSoC period, I don't have any vacation plans, ensuring my availability throughout. I'm open to calls between 6:30 pm IST and 11 pm IST on weekdays, and from 9 am IST to 11 pm IST on weekends, facilitating communication and collaboration as needed.