Proposal: Rocket.Chat ReactJS fullstack Component Create a ready-to-go easy to embed mini-chat React component.

Google Summer of Code 2022

About Me

Name:	Sidharth Mohanty
Rocket.Chat:	sidharth.mohanty
Email:	sidmohanty11@gmail.com
GitHub:	sidmohanty11
LinkedIn:	sidmohanty11
Website:	https://sidmohanty11.github.io/
Country:	India
Phone Number:	+91 9438277475
Time zone:	UTC +5:30 (IST - India)
Pincode:	753012

Abstract

The ultimate goal of the project is to create a full-stack React component node module of the RocketChat application that would be fully configurable, extensible, and flexible for use. It could be easily installed in React-based applications.

Benefits to the Community

It has been more than 7 years and we are still counting, RocketChat has proved to serve over 12 million users in over 150 countries. It has now become the go-to open-source chat solution for many organizations. With that said, when it comes to integrating RocketChat into their applications it is a little painful setup and with the iframe solution also we can see UI leaks which isn't a good experience for users. So, with this full-stack RocketChat component, I aim to solve this issue which will make it ever so simple to integrate RocketChat into their applications.

React has become the most popular front-end framework of choice. Be it a startup or an organization, everyone prefers to use react in most cases. For that reason, I think it just makes sense to create a React.js mini-chat component that can be useful for a large number of RocketChat users.

This will be a simple and effective solution for anyone who is planning to have a chat solution in their application. With this RocketChat component, they can get started right away. For instance, if someone plans to make a chat solution for their application, it is more likely that it will take years to build a robust chat app (keeping in mind that it must be real-time, handle efficient database requests and provide a sturdy server which can scale). This ensures organizations use their v*aluable time building their product* and leave RocketChat with the stress to provide them with a powerful chatting solution.

The RocketChat component, in my opinion, will skyrocket in downloads due to its easily configurable architecture and it will simultaneously increase users of RocketChat as it is a two-way binding (RocketChat component being closely attached to the RocketChat environment). This will provide a standard community discussion environment, increase user engagement, service provider communications, marketplace communications, communities that need to educate their clients and assist them with their product usage, etc.

Goals

During the GSoC period, I would be focused on:

1. Making a ready-to-use react mini-chat component that can be installed in any React-based web application through the npm registry.
2. Making this component configurable and giving flexibility to users to choose from.
3. Integrating real-time chat functionality using the RocketChat-SDK.
4. Integrating native RocketChat functionalities like pinning messages, announcements, reacting to messages, sending emojis, etc.
5. Making the component fully responsive for both large and smaller screens.
6. Providing a native RocketChat feel to the component by using RocketChat's Fuselage design system.
7. Documenting and writing tests for the component.

8. Publishing the end product to the npm registry.

Deliverables

By the end of my GSoC period, I plan to deliver an npm ReactJS library module of RocketChat with the following features:

1. Creation of UI using RocketChat Fuselage design system. [NEW]
2. Providing real-time chat functionality using RocketChat node.js SDK. [NEW]
3. Authentication using RocketChat's Google SSO with an additional choice for <username, password> login (if the user already has an account). [NEW]
4. Adding EmojiOne Emoji Picker to the component to ensure cross-platform functioning of emojis.[NEW]
5. Using Rocket.Chat's REST API to: [NEW]
    - Get channel details
    - Get the channel's pinned messages
    - Get the channel's starred messages
    - Get the channel attachments
    - Send a message to the channel
    - Get messages of the channel
6. Providing the functionality to pin/star/react to any message and mention users. [NEW]

Extras

The following are the improvements/features that I plan to build within the GSoC period if I finish before my planned schedule, and if not then definitely after the GSoC period completion.

These are the other features that are described in the Future Work/After GSoC Ends section:

1. Implementation of thread functionality which is provided by RocketChat.
2. Login using email and password.
3. Migrating the whole codebase to typescript

Implementation Details

1. Overall Component View

Once the whole component is assembled, any React-based web application can just do `npm i @rocket.chat/reactjs-component` to install the full-stack ReactJS RocketChat component into their application.

Significance of each prop:

- **host** - host is the RocketChat instance URL that the in-app chat developer has hosted, it is required for authentication requests, API calls, and real-time socket connection. <STRING>
- **height** - total component height prop <NUMBER | STRING> (analogous to CSS height)
- **Width** - total component width prop <NUMBER | STRING> (analogous to CSS width)
- **isClosable** - boolean to show/hide the close button and also throw an error if it is set to true and [closableState, setClosableState] are not provided to it. <BOOLEAN>
- **closableState** - the state which will track the opening and closing of the RocketChat component. <STATE>
- **setClosableState** - this can be used in both the user's custom button component onClick event and also can be passed through to the RocketChat component for toggling state. <SETSTATE>
- **authType** - this prop acts as a conditional to change the type of authentication the in-app chat developer wants to proceed with. <"emailpass" | "sso" | "both">
- **GOOGLE_SECRET** - required when authType is set to "sso" <STRING>
- **GOOGLE_CLIENT_ID** - required when authType is set to "sso" <STRING>
- **moreOpts** - to show a dropdown full of other functionalities such as pinned messages of the channel, starred messages of the channel and attachments of the channel. <BOOLEAN>
- **className** - to provide extra styling functionality to message component <STRING>
- **roomId** - id of the channel which will be opened and listened to in the RocketChat component <STRING> or,
- **roomName** - name of the channel which will be opened and listened to in the RocketChat component <STRING>

```
<RocketChat
  host={host}
  height={height}
  width={width}
  isClosable={true}
  closableState={toggleCloseState}
  setClosableState={setToggleCloseState}
  authType="both"
  GOOGLE_SECRET={googleSecret}
  GOOGLE_CLIENT_ID={googleClientId}
  moreOpts={true}
  className={className}
  roomId={roomId}
  roomName={roomName}
/>
```

2. Setting up the environment

There are two main bundlers in the market to develop this npm module, both are equally suitable bundlers for development. They offer some perks and come with some drawbacks.

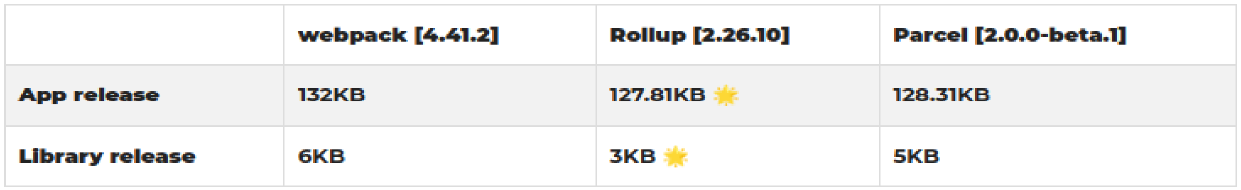
Methods:

1. Using Rollup
2. Using Webpack

For the RocketChat component, I would like to implement it using Rollup as it has become the defacto while developing libraries. Some of the main reasons can be highlighted here in these blogs:

- Benchmarking-rollup-webpack-parcel
- webpack-and-rollup-RichHarris

One of the best features is it produces a lightweight library release.



	webpack [4.41.2]	Rollup [2.26.10]	Parcel [2.0.0-beta.1]
App release	132KB	127.81KB 🌟	128.31KB
Library release	6KB	3KB 🌟	5KB

I've tested out both the bundlers and have set up sample configuration files for creating a component library and they can be found here.

Other essential setups which are going to help in the future:

➢ Setting up prettier with Eslint
➢ Adding "scripts" for tests, build, lint, etc.
➢ Husky pre-commit git hook to run tests, and lint before pushing to GitHub.
➢ A playground directory, to run and test out the application (which will be a simple react application generated from *create-react-app*).
➢ CI pipeline using GitHub actions to check code quality and build/tests.
➢ Add a CD pipeline to automate the process of publishing to npm.

3. Designing the UI using Fuselage Design System

Designing the UI will involve the implementation of the following components

- Box, Message, Message Toolbox, Icons, Button, etc from Fuselage
- Creating an Emoji Picker using Joypixels emojis
- Header portion using Fuselage (<Box />, <Dropdown />, etc)
- The overall component will be divided into roughly three sub-components to ensure minimum re-rendering of the application and increase performance. These components can be visualized as
    - ➢ ChatHeader: Provide details about the channel and a kebab menu which onClick will open up a dropdown suggesting if the user wants to check the attachments, pinned messages, starred messages, etc.
    - ➢ ChatBox: The component to map out all the messages and update UI when a new message is received.
    - ➢ MessageBox: This will be the Input component responsible to send messages and emojis.

Visual representation/Mockups for the RocketChat component:-

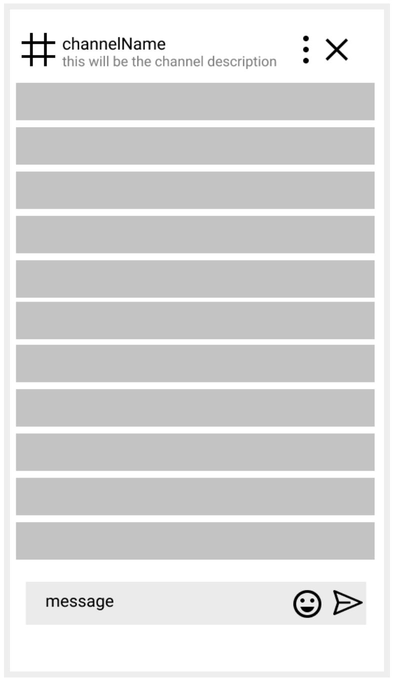
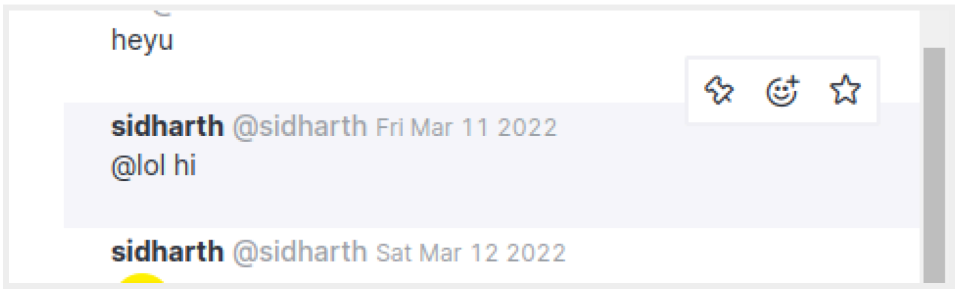


Fig. 1. Visualization of the RocketChat component

We use **Fuselage's own Message component** to show the messages.


heyu

sidharth @sidharth Fri Mar 11 2022
@lol hi

sidharth @sidharth Sat Mar 12 2022

Fig. 2. Visualization of the Chat Message component

More on this can be found on my Figma workflow here.

The design-related props that the component will receive-

1. height and width
2. className for customizations/theming

To make it fully configurable, we can add a prop - **isClosable**, this will help the in-app chat developer to configure it really well with the design as there can be broadly **two scenarios**:

i. When a user wants to keep the RocketChat component hidden and can open/close it with the use of state management.

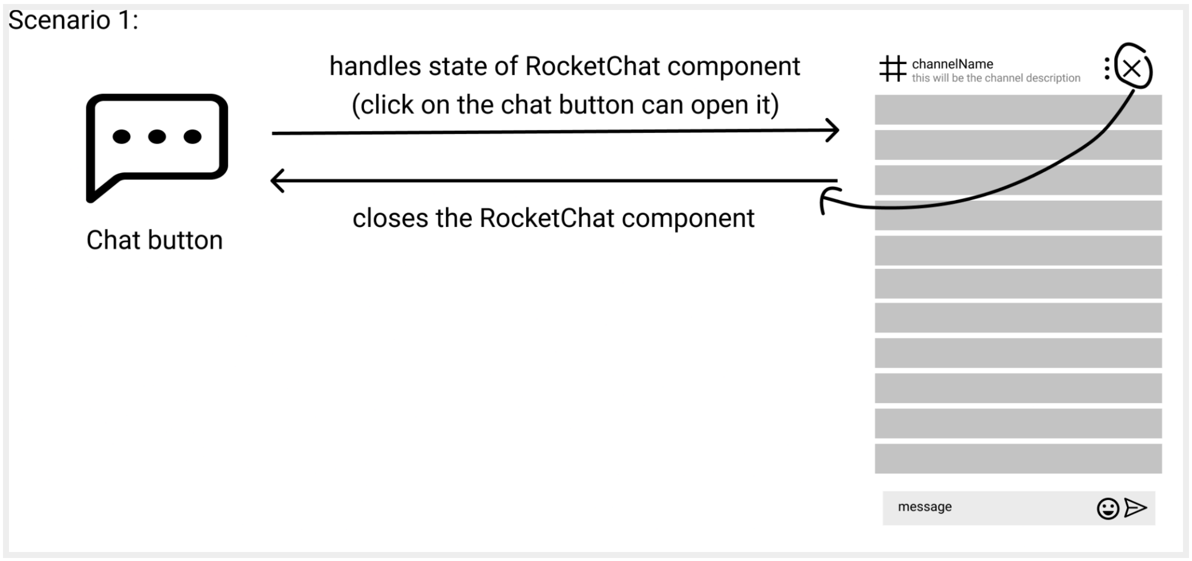

Scenario 1:
handles state of RocketChat component
(click on the chat button can open it)
closes the RocketChat component
Chat button
channelName
this will be the channel description
message

Fig. 3. Visualization of closable functionality of RocketChat component

ii. When a user wants to show the RocketChat component all the time.

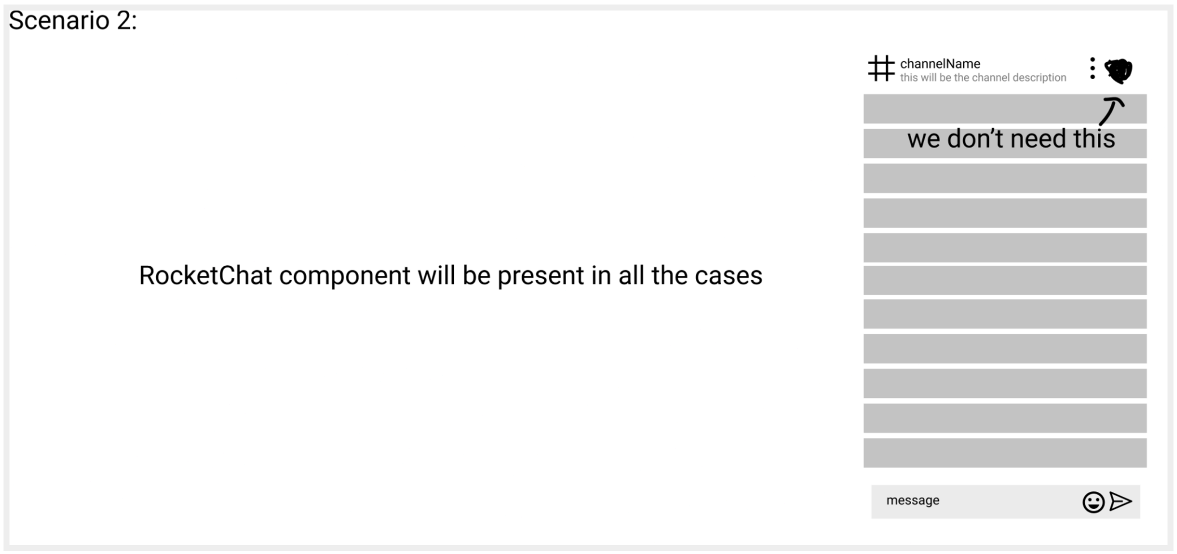

Scenario 2:
channelName
this will be the channel description
we don't need this
RocketChat component will be present in all the cases
message

Fig. 4. When the RocketChat component doesn't need to be closed

4. Authentication using RocketChat's Google SSO or Email/Password

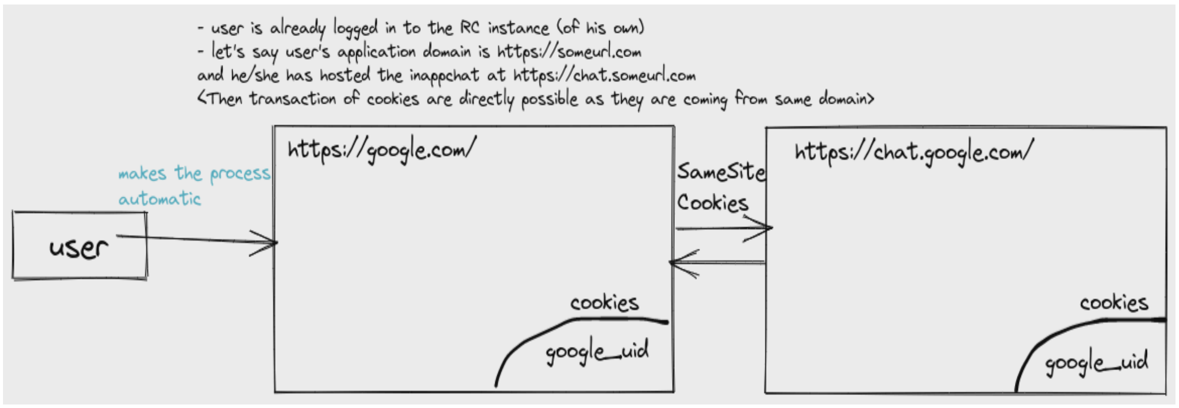
Handling authentication will be one of the greatest challenges of the RocketChat component. We can't even proceed to get messages of the channel if we don't have two things (from the user):

- rc_token and,
- rc_uid

After that only we can think about getting/sending messages from a room by providing the roomId. With that said, how can we get these?

Some ways/situations:

1. <u>Case 1:</u> The In-app chat developer uses the same domain for hosting his/her RocketChat instance. For example, if a user has a shopping site hosted on *myshoppingsite.com,* and he/she hosts the RocketChat instance at *chat.myshoppingsite.com* then every user who joins the RocketChat instance will already have the cookie in the browser. We can leverage this and authenticate the user directly by using the cookies. So that we can directly call API requests and authenticate users.



- user is already logged in to the RC instance (of his own)
- let's say user's application domain is https://someurl.com
and he/she has hosted the inappchat at https://chat.someurl.com
<Then transaction of cookies are directly possible as they are coming from same domain>

Fig. 5. SameSite Cookies mechanism

2. Case 2 (EXTRAs): Most of the developers don't have the patience right away to set up all the things from the start and want to take things in a progressive manner. Out of the box, RocketChat instances support email/password login which can be integrated into the RocketChat component for simpler use cases where in-app chat developers don't want to set up a full-fledged OAuth system and just want to try out things. This will only need the users to put in their email and password (if and only if they already have an account in that RocketChat instance) and with this, we can successfully log in a user and get the rc_token and rc_uid. The endpoint that will be used to authenticate users with login and password is Login endpoint of RocketChat.

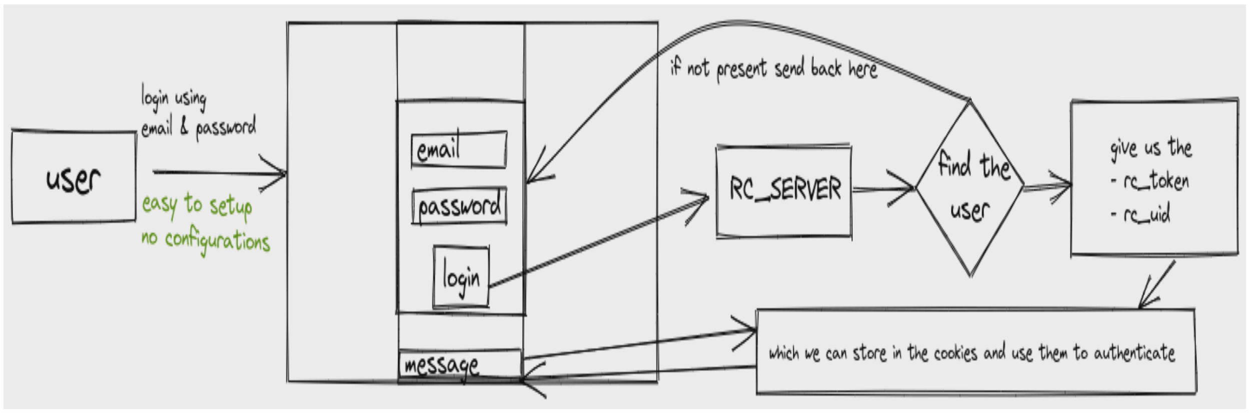
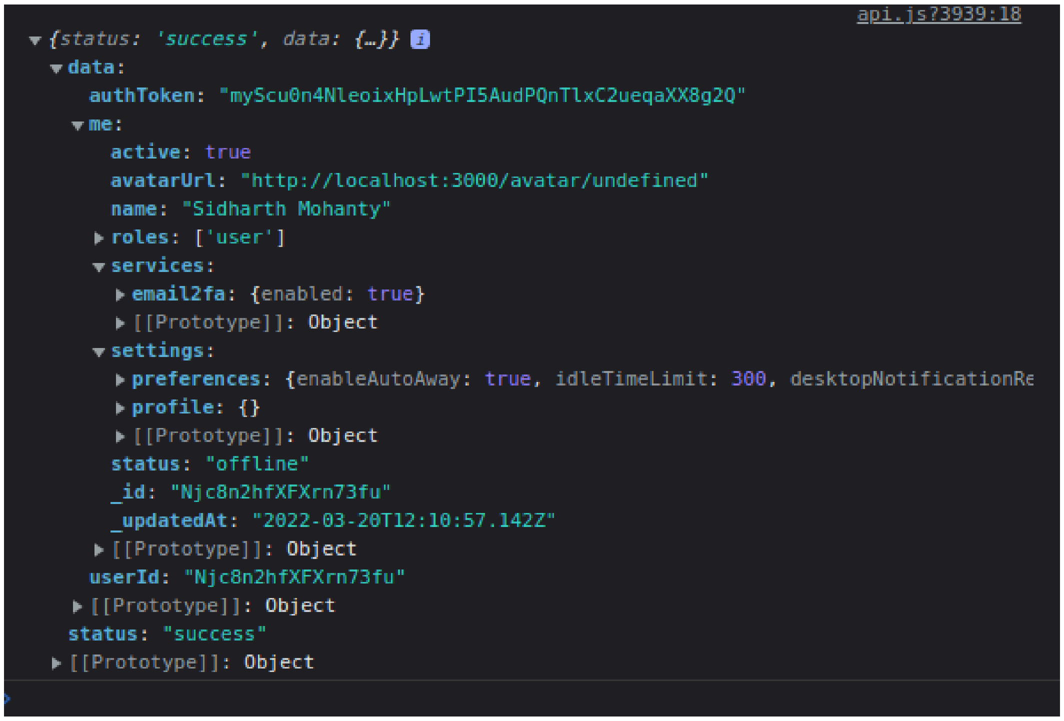

user
login using email & password
easy to setup
no configurations
email
password
login
message
if not present send back here
RC_SERVER
find the user
give us the
- rc_token
- rc_uid
which we can store in the cookies and use them to authenticate

Fig. 6. Workflow: Login with email and password in the RocketChat component

3. Case 3: This will be the most preferable way to set up authentication in the RocketChat component for any production application. For setting up Google sign-in into RocketChat instance, (as almost everyone uses a Gmail account and it is just a click to sign in or signup) we need to configure Google OAuth in RocketChat. This process will need us to set up a google OAuth App in Google's developer console. For setting up Google SSO in the RocketChat component, we need to get the CLIENT_ID and CLIENT_SECRET as props from the RocketChat component user and everything else will be handled in the RocketChat component itself. How you may ask? In a nutshell, we need to set up the google popup (which appears when we click on "Login with Google") and after a successful login into google, we will receive the idToken and accessToken from google which we can use to authenticate the user using RocketChat's Login with Google endpoint, as we require only those two parameters. And we will be getting a user response back from the RocketChat server from which rc_token and rc_uid can be extracted out and we are again good to go in making API calls and requests.

   A sample response would look like this:



```
                                                              api.js?3939:18
▼ {status: 'success', data: {…}} i
  ▼ data:
      authToken: "myScu0n4NleoixHpLwtPI5AudPQnTlxC2ueqaXX8g2Q"
    ▼ me:
        active: true
        avatarUrl: "http://localhost:3000/avatar/undefined"
        name: "Sidharth Mohanty"
      ▶ roles: ['user']
      ▼ services:
        ▶ email2fa: {enabled: true}
        ▶ [[Prototype]]: Object
      ▼ settings:
        ▶ preferences: {enableAutoAway: true, idleTimeLimit: 300, desktopNotificationRe
        ▶ profile: {}
        ▶ [[Prototype]]: Object
        status: "offline"
        _id: "Njc8n2hfXFXrn73fu"
        _updatedAt: "2022-03-20T12:10:57.142Z"
      ▶ [[Prototype]]: Object
      userId: "Njc8n2hfXFXrn73fu"
    ▶ [[Prototype]]: Object
    status: "success"
  ▶ [[Prototype]]: Object
```

Fig. 7. Login with Google SSO through RocketChat component Response

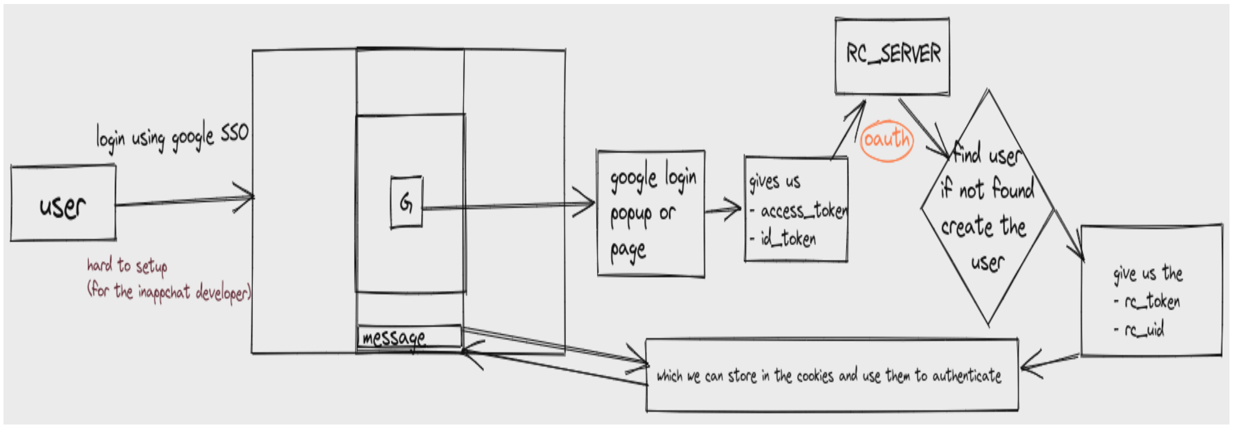


Fig. 8. Workflow: Login with Google SSO through RocketChat component

5. Providing real-time chat functionality using RocketChat node.js SDK

Rocket.Chat's node.js SDK provides us with a DDP(Distributed Data Protocol) driver. DDP is a stateful WebSocket protocol that meteor uses for client-server communication. Underneath the hood, RocketChat SDK uses its Realtime API to connect to a publication endpoint, receive data and listen for updates. This subscription model ensures that real-time messaging can be done without relying on 3rd-party providers or by re-writing socket implementations.

Implementing this will require us to use the latest SDK release (used by the mobile client). We can import a RocketChat client instance that will handle the socket connection. This instance implements socket connection methods that are being used by RocketChat in their LiveChat and Mobile App.

So, in this case, most of the heavy lifting will be done by the SDK and we just need to connect it to the frontend. A **sample code** will look something like this:

```
import { Rocketchat } from "@rocket.chat/sdk";

const client = new Rocketchat({
  logger: console,
  protocol: "ddp",
  host,
  useSsl
});

const runRealtime = async (token, rid) => {
    try {
       await client.connect();
       // login to the RC-server
       await client.resume({ token });
       // subscribe to the room's messages
       await client.subscribe("stream-room-messages", rid);
       client.onMessage(() => {
        // update data real-time
       });
     } catch(err) {
       // handle error
    }
 };
```

6. The Message Component in Depth

There is a whole Message component in RocketChat's Fuselage which can be found here.

- It provides us with a className prop which can be useful to change its themes but it requires them to set an `!important` tag to the CSS.
- MessageHeader, MessageName, MessageUsername will be used to display the user details.
- MessageBody is the section where we will be rendering the messages sent. For this a simple and re-usable Markdown component is written here in RocketChat MarkdownText Component.
- Toolbox will come in handy to show 3 interactive options to the users which are,
    1. Pin that message
    2. React to that message
    3. Star that message

```
<Message className="customclass" clickable key={m._id}>
   <MessageContainer>
      <MessageHeader>
         <MessageName>{m.u.name}</MessageName>
         <MessageUsername>@{m.u.username}</MessageUsername>
         <MessageTimestamp>
           {new Date(m.ts).toDateString()}
         </MessageTimestamp>
       </MessageHeader>
       <MessageBody>
          {/* formats the message (legacy) */}
          <MarkdownText body={m.msg} />
       </MessageBody>
    </MessageContainer>
    <MessageToolboxWrapper>
       <MessageToolbox>
         {/* onClick will pin the message */}
         <MessageToolboxItem icon="pin" />
           {/* onClick will open the EmojiPicker for reacting to the
message*/}
         <MessageToolboxItem icon="emoji" />
         {/* onClick will star the message */}
         <MessageToolboxItem icon="star" />
       </MessageToolbox>
     </MessageToolboxWrapper>
 </Message>
```

7. Using RocketChat REST API and building a solid foundation for RocketChat component API requests

Below are the mentioned endpoints, the reason why to use that endpoint, and a sample code snippet that gives us an overview.

For all these requests, we need to make a request to the RocketChat server. For each endpoint, a separate helper function can be created which will come in handy and we can maintain a modular design for the codebase.

```
export const handler = async (params) => {
  try {
    const response = await
axios.get(`${params.host}/api/v1/<<endpoint>>?roomId=${params.ri
d}`, { headers: {
      "Content-Type": "application/json",
          "X-Auth-Token": cookies.rc_token,
          "X-User-Id": cookies.rc_uid,
    } })
    return handleResponse(response);
  } catch (err) {
   // handle error
  }
};
```

Send Message endpoint is the most crucial and basic one, that will be used to send messages and as we have subscribed to the room messages, we can update the UI in real-time.

/POST request

★ Send message - this endpoint will be used for sending messages to the channel.

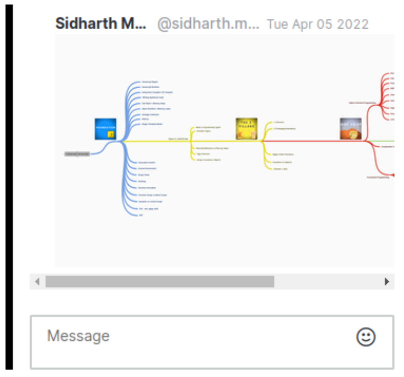
The endpoints listed below are going to be used all over the application.

/GET requests to fetch our desired response.

★ Channel Info - this endpoint will give us all necessary channel information like room name, announcement (if any), and description.

★ Get messages - this endpoint will give us the latest 50 messages, which we can modify if we like (but it will be more than enough to show in the in-app chat component). This will get iterated in the `<ChatBox />` component using the `<Message />` component of RocketChat's Fuselage Design system.

★ Get attachments - this endpoint will be used to get all the attachments sent in the channel and we can preview them from the in-app chat. For starters, it will show videos, photos, mp3 and mp4 files and for the rest, it will appear as a file download button with the filename.ext so that the user can download and use the file.



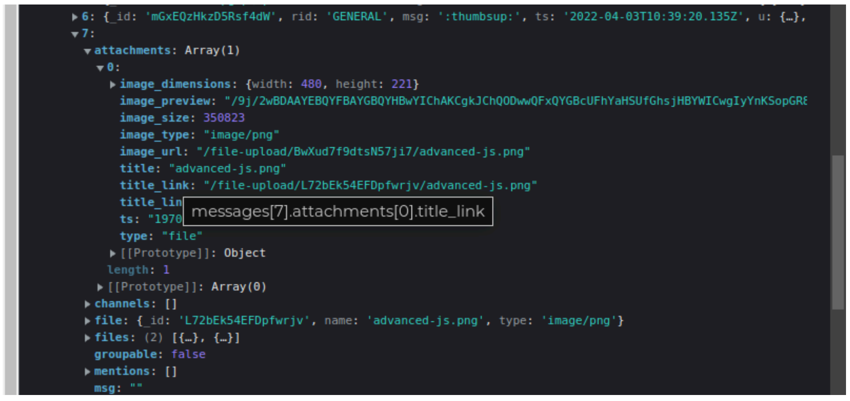

```
▸6: {_id: 'mGxEQzHkzD5Rsf4dW', rid: 'GENERAL', msg: ':thumbsup:', ts: '2022-04-03T10:39:20.135Z', u: {…},
▾7:
  ▾attachments: Array(1)
    ▾0:
      ▸image_dimensions: {width: 480, height: 221}
       image_preview: "/9j/2wBDAAYEBQYFBAYGBQYHBwYIChAKCgkJChQODwwQFxQYGBcUFhYaHSUfGhsjHBYWICwgIyYnKSopGR
       image_size: 350823
       image_type: "image/png"
       image_url: "/file-upload/BwXud7f9dtsN57ji7/advanced-js.png"
       title: "advanced-js.png"
       title_link: "/file-upload/L72bEk54EFDpfwrjv/advanced-js.png"
       title_lin messages[7].attachments[0].title_link
       ts: "1970
       type: "file"
      ▸[[Prototype]]: Object
     length: 1
    ▸[[Prototype]]: Array(0)
  ▸channels: []
  ▸file: {_id: 'L72bEk54EFDpfwrjv', name: 'advanced-js.png', type: 'image/png'}
  ▸files: (2) [{…}, {…}]
   groupable: false
  ▸mentions: []
   msg: ""
```

Fig. 9. Example Image Attachment Object

- ★ Get pinned messages - this endpoint will be used to get all the pinned messages of the channel and will be presented in the pinned messages section.
- ★ Get starred messages - this endpoint will be used to get all the starred messages of the channel and will be presented in the starred messages section.

These functionalities are going to be used in the message toolbox -

/POST requests

- ★ Pin message - Endpoint used to pin a message in the channel.
- ★ Star message - Endpoint used to star a message in the channel.
- ★ React to a message - Endpoint used to react to a message in a channel.

8. Emoji Picker

For emojis, RocketChat uses **Emojione's emoji pack**. To implement this a custom Emoji Picker component will be made (a working example is shown below in the rollup environment). This component will be used in both the react to a message part and also for sending emojis from the `<InputBox />` component.

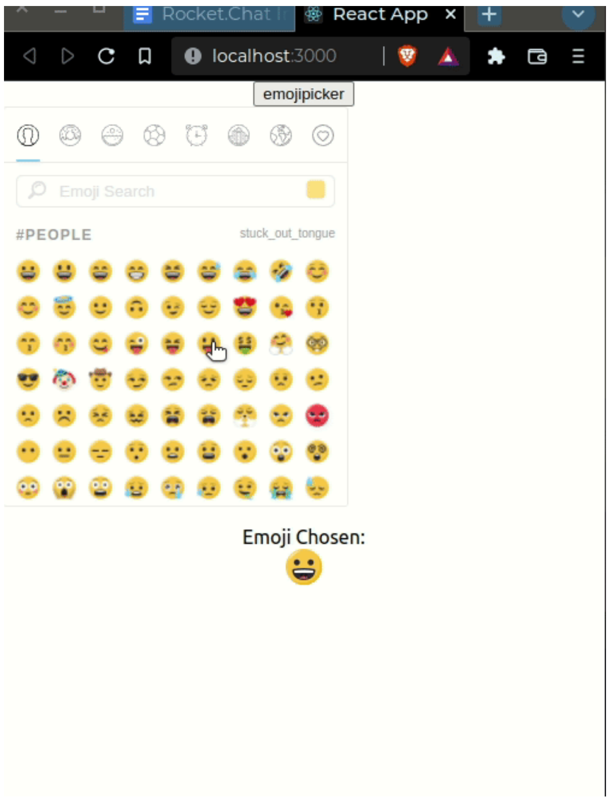


Fig. 10. Example EmojiPicker component for RocketChat component

An example code snippet for implementation is attached below:

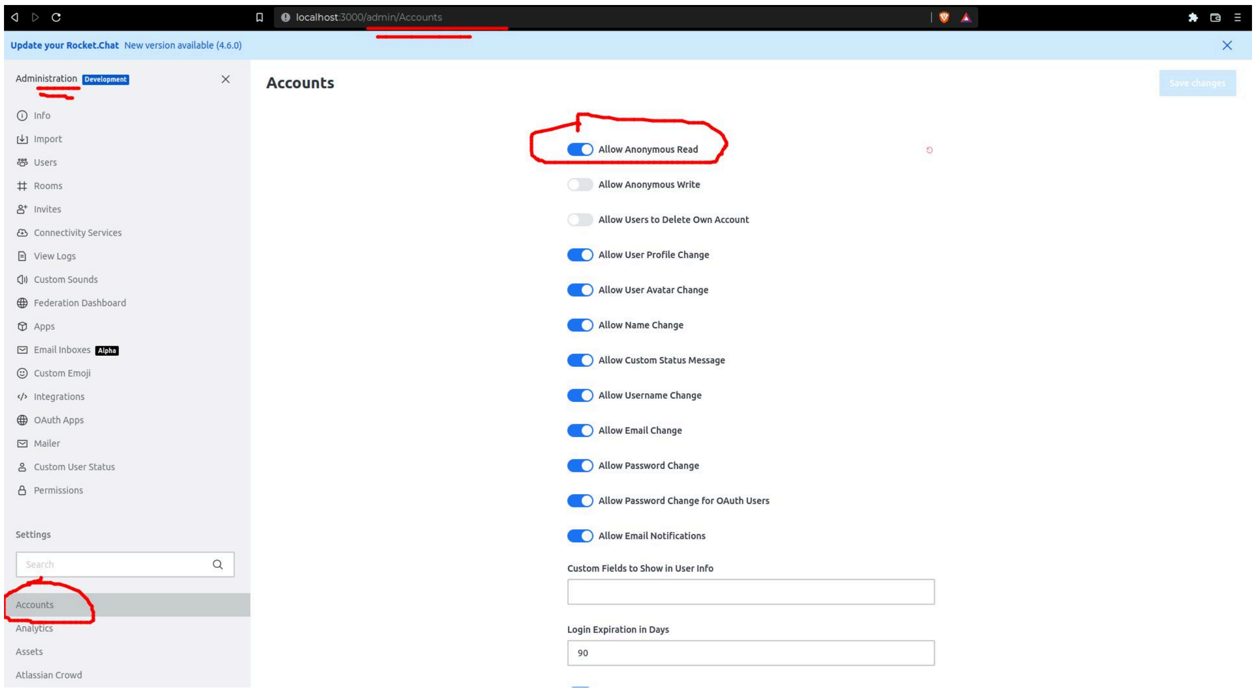
Code:

```
import React, { useState } from 'react'
import EmojiPicker from "emoji-picker-react";
import JSEMOJI from "emoji-js";

let jsemoji = new JSEMOJI();
jsemoji.img_set = "emojione";
jsemoji.img_sets.emojione.path =
  "https://cdn.jsdelivr.net/emojione/assets/3.0/png/32/";

jsemoji.supports_css = false;
jsemoji.allow_native = false;
jsemoji.replace_mode = "unified";

const App = () => {
  const [emoji, setEmoji] = useState()
  const [openEmoji, setOpenEmoji] = useState(false)

  const handleClick = (n, e) => {
    let emoji = jsemoji.replace_colons(`:${e.name}:`);
    setEmoji(emoji);
  }
  return (
    <div>
            <button  onClick={()  =>  setOpenEmoji(prevState  =>
!prevState)}>emojipicker</button>
      {openEmoji && <EmojiPicker onEmojiClick={handleClick} /> }
       <p>Emoji  Chosen:  <div  dangerouslySetInnerHTML={{  __html:
emoji }}></div></p>
    </div>
  )
}

export default App
```

9. Ensuring read-only functionality when the user is not logged in?

To provide read-only messages functionality to users while they are not logged in to the RocketChat component, there are some steps the in-app chat developer should take. Firstly, he/she must allow Anonymous Read in `Administration -> Accounts -> Enabling`

Anonymous Read.



- [✓] Allow Anonymous Read
- [ ] Allow Anonymous Write
- [ ] Allow Users to Delete Own Account
- [✓] Allow User Profile Change
- [✓] Allow User Avatar Change
- [✓] Allow Name Change
- [✓] Allow Custom Status Message
- [✓] Allow Username Change
- [✓] Allow Email Change
- [✓] Allow Password Change
- [✓] Allow Password Change for OAuth Users
- [✓] Allow Email Notifications

Custom Fields to Show in User Info:

Login Expiration in Days: 90

Fig. 11. Enabling "Allow Anonymous Read"

After that, we can send a request to the server to send us all the messages of the channel using anonymous-read endpoint.

10. Documentation

As the RocketChat component has so many facets to it, it is not possible for the in-app chat developer to predict an outcome and work on that basis. So, I will be maintaining full-fledged documentation that will help the in-app chat developer.

From setting up the RocketChat component's host to authentication to the anonymous read part, every portion will be super descriptive and full of diagrams so that it will take a maximum of one day to set up the RocketChat component.

I will be adding both quickstart video tutorials and also tutorials explaining how I made the RocketChat component so that it can be developed with any frontend framework very quickly.

Also, I will be answering all the FAQs that might arise while developing the RocketChat component.

11. Publishing the npm package and testing

Testing:

- React testing library along with jest will be used to test the component.
- I will be testing the component prop-wise to ensure that it works properly and as expected in every situation.

Publishing to npm (steps):

- Login into npm, npm login
- Then, add all the necessary details in the package.json
- npm publish
- To update the package and handle the versioning part efficiently, np can be used and we can start a consistent release cycle.

GitHub Actions:

- Add config files to run npm test and npm build to every push to develop branch
- This will ensure our application stays working in all cases.
- Add a CD pipeline to publish the application to the npm registry when there is a push to the prod branch.

Detailed Work Plan

Application Review Period (Upto 20th May)

During the application review period, I plan to dive deep into the RocketChat node.js SDK code and figure out all the possibilities that could make the RocketChat component more extensible in the future. I would also be researching a lot through the RocketChat codebase, about how they handle certain situations and take notes to implement a similar robust structure in the RocketChat component.

Community Bonding Period (May 20th - June 12th)

During the community bonding period, I intend to set up clear communication with my mentor. Discussion regarding any blockers or edge case scenarios that we might encounter will be a priority.

Also, I would like to know the community better by connecting to more members of the organization and participating in discussions to know the future plans for the SDK and RocketChat so that I can develop the RocketChat component in a better way.

[coding period starts]

Week 1 (13th June - 19th June)

- Setting up the development environment
- Creating the UI of the RocketChat component with some dummy data, Fuselage icons, and components.

Week 2 (20th June - 26th June)

- UI refinements
- Testing all possible scenarios for theming the RocketChat component
- Working on the responsiveness for both smaller and larger screens

Week 3 (27th June - 3rd July)

- Setup Authentication with Google SSO Login

Week 4 (4th July - 10th July)

- Authentication portion continued, if not finished
- Setup sending and receiving messages in real-time
- Handling any possible errors

Week 5 (11th July - 17th July)

- Getting channel details (Chat Header)
- Handling announcements
- Providing more options (channel's pinned, starred, attachments)

Week 6 (18th July - 24th July)

- Legacy Markdown Component setup for the rendering of messages

[1st phase evaluation]

Week 7 (25th July - 31st July)

- EmojiPicker Component
- Successfully sending and receiving emojis

Week 8 (1st August - 7th August)

- Handling attachments

Week 9 (8th August - 14th August)

- Reacting to messages
- Handling mentions to other users

Week 10 (15th August - 21st August)

- Pinning messages
- Starring messages

Week 11 (22nd August - 28th August)

- Rigorous testing
- CI/CD pipeline setup

Week 12 (29th August - 4th September)

- Buffer period and bug fixes
- Handling unpredictable edge case scenarios

[final week]

Week 13 (5th September - 12th September)

- Documentation
- Creating a summary of work done during the GSoC period
- Final work product launch

Future Work/After GSoC Ends Section

I would love to contribute more to the RocketChat component by adding more interesting features to it, making it a more effective component that can provide a great engagement level to the web apps which decide to use this component.

I would also try to implement the features that are listed in the Extras section.

1. **Threads Functionality** - threads are one of the most used and useful features of RocketChat which separates itself from its competitors. Providing thread functionality will be a lot beneficial for the users as it will decrease a lot of clutter. It can be implemented by Get Channel Threads endpoint which will list all the thread's "start" messages. By this, we can get the id of every thread which is called "tmid" in RocketChat, and send another request to the Get Thread Messages endpoint which will give us all the follow-up messages inside that thread.
2. **Login with email and password** - This section has been described in the Authentication using RocketChat's Google SSO or Email/Password implementation section(case-ii).
3. **Migrating from JS to TS** - This will help us in the long run when the RocketChat component's codebase grows, new developers can just go through type definitions and understand the codebase easily. This will also help us provide auto-suggestions in VS Code and our code will be less vulnerable to unpredictable results.

Related Work Done

I have already made myself familiar with the project because I was working on integrating a mini in-app chat for the RocketChat 2022 Alumni Summit Conference in RC4Community.

I have developed the barebone structure of the in-app chat application. It was developed for the purpose of the Alumni conference and it supports all the basic features like sending/receiving messages, emojis, etc.

1. [Merged] **RC4Community #78**: Working Demo of the application without real-time functionality (only REST API calls) and cookies for authentication.
2. [Merged] **RC4Community #89**: Working Demo of the application with real-time functionality (with RocketChat's node.js SDK).

I have also built several wireframes for various deliverables, features, and the overall workflow of the RocketChat component. The wireframes used in this proposal as well as several more for in-detailed flow can be found in this Figma file.

Relevant Experiences

I have been a consistent and active contributor to RocketChat for the past five months now. I have also remained the **top contributor** with the most number of PRs in the GSoC 2022 Leaderboard. I have contributed to numerous projects within the organization.

A list of my contributions is listed below:

Issues:

RocketChat/Rocket.Chat

- Issues (Total: 3)

RocketChat/Rocket.Chat.Fuselage

- Issue (Total: 1)

RocketChat/RC4Community

- Issue (Total: 1)

Pull Requests:

Total: 30 **Merged**: 18 **Under-review**: 12

1. [Merged] **RC4Community #47** **-** [NEW] RocketChatLinkButton Component
2. [Merged] **RC4Community #66** - wip: github components kit
3. [Merged] **RC4Community #78** - wip: inappchat component for virtual conference
4. [Merged] **RC4Community #83** - Add Mainstage Speakers Conference Page
5. [Merged] **RC4Community #85** - [NEW] inappchat setup docs
6. [Merged] **RC4Community #86** - [NEW] inappchat in greenroom
7. [Merged] **RC4Community #89** - [NEW] Realtime InAppChat using RocketChat-SDK
8. [Merged] **RC4Community #93** - [FIX] Re-rendering of InAppChat
9. [Merged] **RC4Community #97** - [IMPROVE] mainstage design with coutdown, local date showtime and speakers
10. [Merged] **RC4Community #107** - [FIX] handling error and not showing InputBox when not authenticated
11. [Merged] **RC4Community #112** - [IMPROVE] Inappchat improvements and fixes
12. [Merged] **RC4Community #118** - [IMPROVE] realtime emoji animation
13. [Merged] **RC4Community #139** - chore: changing from community.liaison to open
14. [Merged] **Rocket.Chat #23605** - [IMPROVE] Add Rocket.Chat version to User-Agent header for oembed requests
15. [Merged] **Rocket.Chat #23882** - [FIX] Custom emoji route in admin
16. [Merged] **Rocket.Chat #23970** - [FIX] Filter ability for admin room checkboxes
17. [Merged] **Rocket.Chat #24117** - [FIX] Custom Emoji Image preview
18. [Merged] **Rocket.Chat #24933** - [FIX] Deactivating user breaks if user is the only room owner

19. [Open] **Rocket.Chat #24440** - [IMPROVE] Cursor pointer to all the action buttons
20. [Open] **Rocket.Chat #24135** - [NEW] omnichannel delete contact functionality
21. [Open] **Rocket.Chat #24116** - [IMPROVE] Omnichannel contact updates after addition or edit of any contact
22. [Open] **Rocket.Chat #24063** - [IMPROVE] showing role names instead of random ids everywhere
23. [Open] **Rocket.Chat #23764** - [IMPROVE] Optional Ability to delete DMs for users
24. [Open] **Rocket.Chat #23685** - [IMPROVE] Adding a jump to message button to quote messages
25. [Open] **Rocket.Chat #23655** - [IMPROVE] Adding a separate "Mark Threads Read" for reading all unread threads
26. [Open] **Rocket.Chat #23650** - [IMPROVE] Drag and Drop UI for files only
27. [Open] **Rocket.Chat #23632** - [IMPROVE] Adding a noscript tag for letting users know if they have JS disabled
28. [Open] **RC4Community #51** - [On Hold - June 22] Pre commit hooks
29. [Open] **Rocket.Chat.Fuselage #699** - fix: handling visibility of options in multiselect
30. [Open] **Rocket.Chat.Fuselage #700** - docs: Fuselage usage docs and npm links fix

In addition to contributing to RocketChat projects, I always have been looking for opportunities to help out the community in any way possible, this can be demonstrated by RocketChat.GSoCLeaderboard App which I made in order to simplify the process of addition of potential contributors to the GSoC leaderboard.

Prior to contributing to RocketChat, I have also published an open-source product that can help developers document their code called jsnotion which has been receiving consistent downloads(peak downloads reached: 299/day).

Projects that I've worked on

➢ **jsnotion**

Description: A code documentation application which is an installable npm package, that can run code in the browser itself without needing any backend service.

Highlight: Made a custom esbuild plugin to handle code bundling in the browser. Receiving consistent downloads daily (93+ per month) and peak downloads reached is 299 per day.

Technologies Used: TypeScript, React, Redux, Express, Commander.js

Why is this relevant to RocketChat ReactJS full stack project?

Both of them are developed in ReactJS and possess full-stack capabilities.

Project Link: https://github.com/sidmohanty11/jsnotion

- ### convo

  Description: A full-stack chatting application that has a similar system design as WhatsApp.

  Highlight: Working with WebSockets and real-time messaging functionality.

  Technologies Used: Vue.js, Golang, MongoDB

  Why is this relevant to RocketChat ReactJS full stack project?
  A clear understanding of how REST APIs, Websockets, and Subscription-based models work.

  Project Link: https://github.com/sidmohanty11/convo

- ### Next-ecom

  Description: A full-fledged e-commerce application with Shopify API integration.

  Highlight: Scalable code and Shopify API integration.

  Technologies Used: Next.js, GraphQL, Shopify API, TypeScript

  Why is this relevant to RocketChat ReactJS full stack project?
  Integrating a foreign API by going through its documentation.

  Project Link: https://github.com/sidmohanty11/next-ecom

- ### mingo

  Description: An HTTP client library similar to axios, developed in golang.

  Highlight: Client requests and API calls.

  Technologies Used: Golang

  Why is this relevant to RocketChat ReactJS full stack project?
  Making API calls, knowledge of request/response headers, and cookies. Comfortable with language switches and willingness to learn and adapt.

  Project Link: https://github.com/sidmohanty11/mingo

Why do I think I’m well suited for this project?

I am well versed with React.js and while working on the InAppChat component for RC4Community, this has enhanced my knowledge and ideas about how I can build it in the best way.

Staying the top contributor this season shows my hard work and love for the RocketChat community. I am a good team player and I actively lookout to help my peers, for this, I have made a lot of friends in the open-source community and was nominated by all the team members of RC4Community to add this milestone **commit**.

I have been contributing to RocketChat for several months now, and I have familiarized myself with the base model on which RocketChat works. I have gathered a lot of knowledge about its API and also about the SDK.

Moreover, I love the RocketChat community. I have learned a lot from my peers and mentors during this period of time and would like to do more of this in the future.

Time availability

How much time do you expect to dedicate to this project?

As my end-semester exams will be over on April 10th, I don’t have any major academic commitments throughout the GSoC timeframe. I would be able to dedicate 40hrs per week to this project.

Please list jobs, summer classes, and/or vacations that you’ll need to workaround.

I haven’t joined any summer classes or internships. I don't have any plans to go on a vacation anytime between the GSoC period. I will be available for calls from 10 am IST to 11 pm IST on both weekdays and weekends.