rocket.chat

Extended LLM Prompt Editor/Explorer App

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

Extended LLM Prompt Editor/Explorer App

Empowering Rocket.Chat users to freely converse with open-source LLMs, manage, and share prompts seamlessly.

Abstract

The LLM Prompt Editor/Explorer project aims to create a comprehensive Rocket.Chat app designed to facilitate interactive and efficient development with open-source Large Language Models (LLMs) like Mistral, Llama 2, and Phi. This app will enable users to engage in free-form conversations with LLMs, manage their prompts, and share insights across the Rocket.Chat platform. The goal is to deliver a Rocket.Chat app that allows users to explore, experiment, and collaborate with LLMs directly within the chat environment, enhancing productivity and innovation in AI/ML development.

Benefits to the community

The LLM Prompt Editor/Explorer App will offer substantial benefits to the Rocket.Chat community, particularly for users interested in AI/ML development and experimentation.
For the community, the LLM Prompt Editor/Explorer App will offer several key advantages:

1. **Enhanced Development Efficiency:** The app will enable users by integrating a prompt editor directly into Rocket.Chat, allowing them to interact with LLMs and manage prompts within the chat environment.
2. **Collaboration and Sharing:** The app will facilitate knowledge sharing and peer learning by enabling users to share their prompts and insights with the community, fostering innovation in AI/ML solutions.
3. **Personalized Learning and Experimentation:** The configurability of the app will allow users to tailor their interactions with LLMs, supporting a more engaging learning experience.

4. **Promotion of AI/ML Literacy:** By providing a user-friendly interface for LLM interaction, the app will promote AI/ML literacy.
5. **Community Growth and Recognition:** The app's development and adoption will contribute to the growth and recognition of the Rocket.Chat project, reinforcing its position as a platform that supports innovation and collaboration in AI/ML.

Goals

During the GSoC period, I will focus on:

1. The app will allow users to converse with LLMs like Mistral, Llama 2, and Phi, manage conversation history, and share prompts with external applications.
2. The app will be designed for easy interaction with LLMs, including intuitive controls for managing prompts and sharing content.
3. Users will be able to customize their interaction with LLMs, including selecting the LLMs to converse with and configuring the response format.
4. Detailed documentation will be provided to facilitate the app's adoption and integration by the Rocket.Chat community.

Deliverables

By the end of the GSoC period, the Extended LLM Prompt Editor/Explorer App aims to deliver the following features:

1. The app will feature a fully functional prompt editor that enables free conversation with open-source Large Language Models (LLMs) such as Mistral, Llama 2, and Phi. This editor will support the management of conversation history, including the ability to save, delete, and rename prompts.
2. The app will be designed with an intuitive interface for easy interaction with LLMs, including controls for managing prompts and sharing content with external applications. We can build on top of Golem's existing solution.
3. Users will be able to customize their interaction with LLMs, including selecting the LLMs to converse with and configuring the response format.
4. The app will allow prompt editors to be configured in different channels, ensuring that a prompt editor configured in one channel is not visible in other channels. This feature will enhance privacy and customization for users.

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 (prompt editors and information related to them 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^

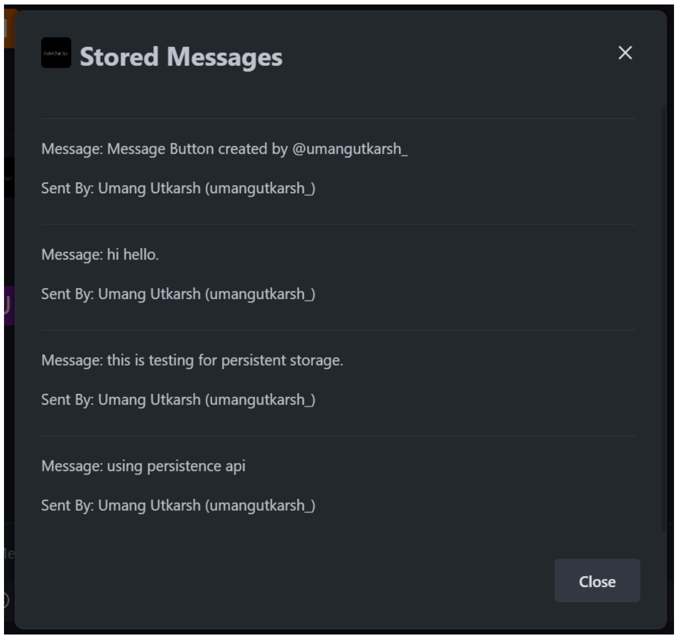
```
public async findByName(
        room: IRoom,
    ): Promise<Array<string>> {
        const associations: Array<RocketChatAssociationRecord> = [
            new RocketChatAssociationRecord(RocketChatAssociationModel.MISC,
'message'),
            new RocketChatAssociationRecord(RocketChatAssociationModel.ROOM,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

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.



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

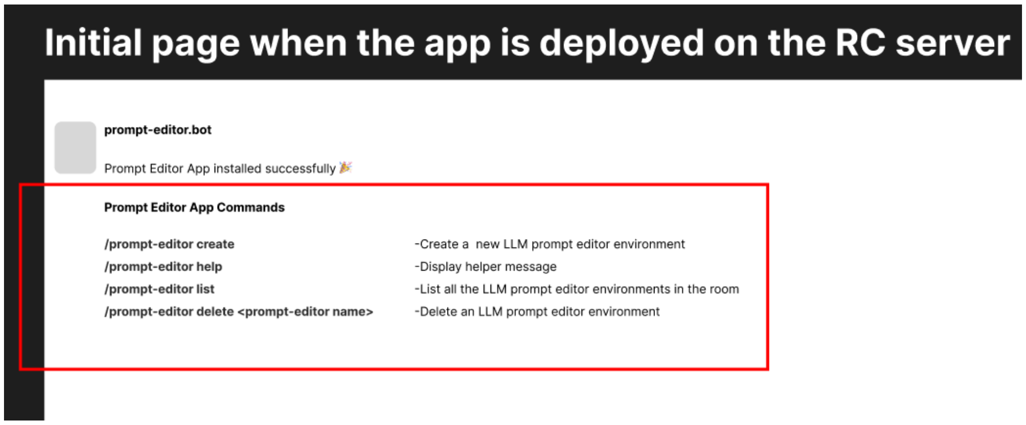
The following link provides the code for the approach I have followed - **Link**

Extended LLM Prompt Editor/Explorer App Flow

To outline the basic flow of the Extended LLM Prompt Editor/Explorer App for Rocket.Chat, including the configuration through the app's settings, we'll follow a structured approach.

1. Installation and Initial Setup

Upon Installation: The app will send a direct message to the admin of the workspace, providing basic information and features of the app, along with instructions on how to use it with slash commands.



prompt-editor.bot

Prompt Editor App installed successfully 🎉

Prompt Editor App Commands

/prompt-editor create	-Create a new LLM prompt editor environment
/prompt-editor help	-Display helper message
/prompt-editor list	-List all the LLM prompt editor environments in the room
/prompt-editor delete <prompt-editor name>	-Delete an LLM prompt editor environment

More information about the app will be visible to the user when installed using these methods.

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

    }
```

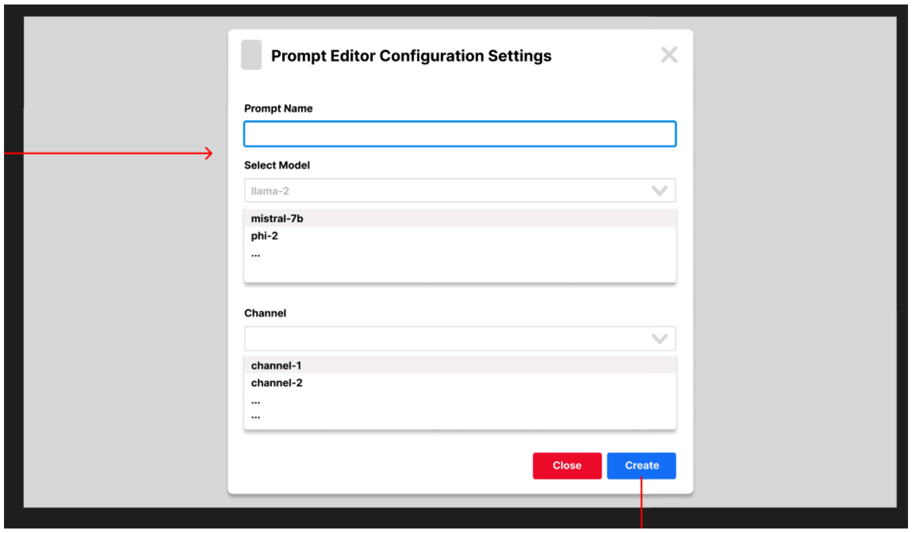
```
public async initialize(configurationExtend: IConfigurationExtend, environmentRead:
IEnvironmentRead): Promise<void> {

    }
```

2. Slash Commands Overview

/prompt-editor create: Allows users to create a new prompt editor in the current room, specifying the LLM they wish to converse with.
/prompt-editor help: Offers information about the app, its features, and how to configure it.
/prompt-editor list: Lists all prompt editors configured in the current room.
/prompt-editor delete: Enables users to delete a specific prompt editor by name.
/prompt-editor configure: Allows users to configure the prompt editor.

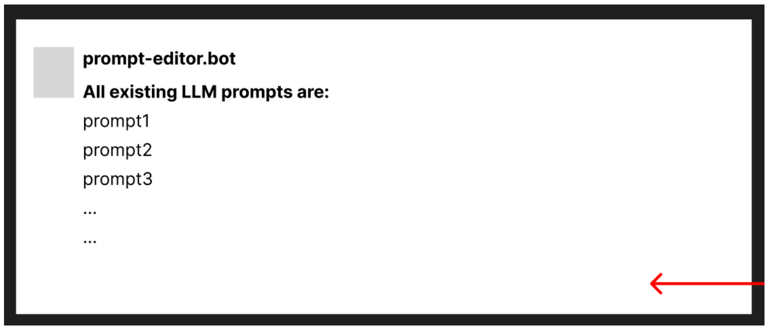
3. Configuring the App

Configuration Modal: After installation, the app requires configuration. Users can configure the app through a configuration modal, or other methods.



Settings in Rocket.Chat: Users can also configure the app through the Rocket.Chat interface by navigating to *Menu -> Installed -> Private Apps -> LLM Prompt Editor -> Settings*. This provides an additional layer of convenience and control for users.

More configuration settings can be provided as the app progresses.



prompt-editor.bot

All existing LLM prompts are:

prompt1
prompt2
prompt3
...
...

After configuring, the ***/prompt-editor list*** slash command will provide the list prompts configured in the current room.

All the features to interact with large language models, and interacting with the prompt editor can be provided using the golem's existing solution which is described in [features of the prompt editor] section.

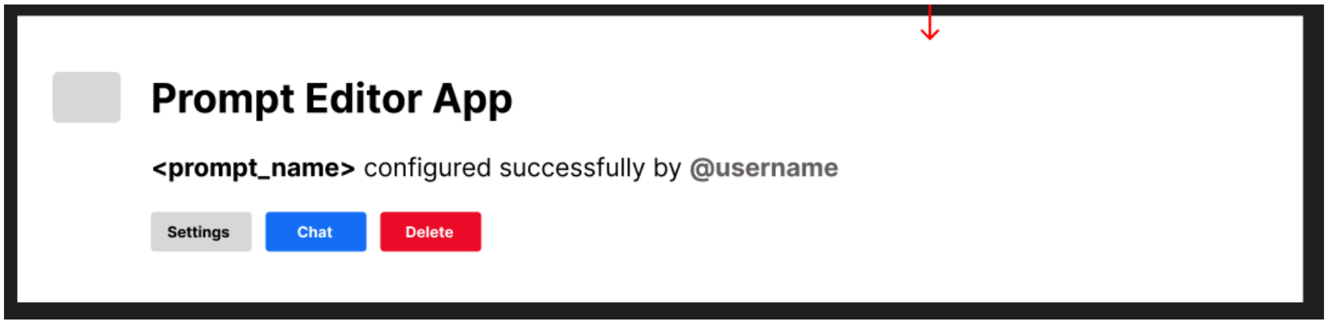
4. Daily Use

Interacting with LLMs: Users interact with the configured LLMs through the prompt editor, enabling free conversation, and management of prompts.

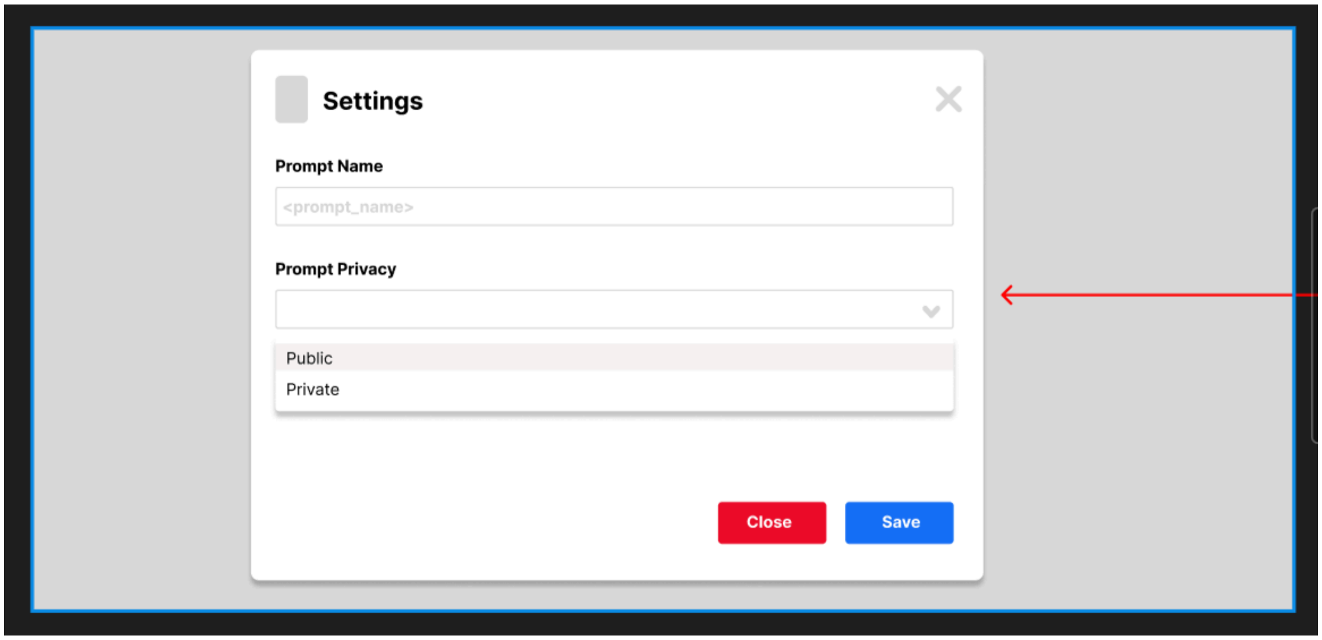
Managing Prompts: Users can manage their prompts within the editor, including saving, deleting, and renaming prompts.

5. User Interaction

Slash Commands for Control: Users interact with the app through slash commands, enabling them to create, list, delete, and configure prompt editors.


Prompt Editor App

<prompt_name> configured successfully by **@username**

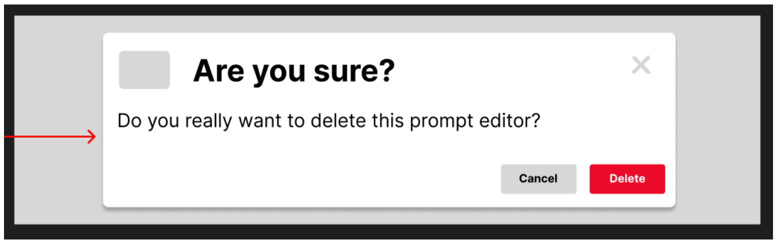

Settings

Prompt Name

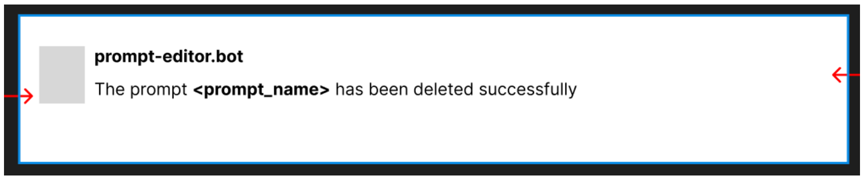
<prompt_name>

Prompt Privacy

Public
Private


Are you sure?

Do you really want to delete this prompt editor?

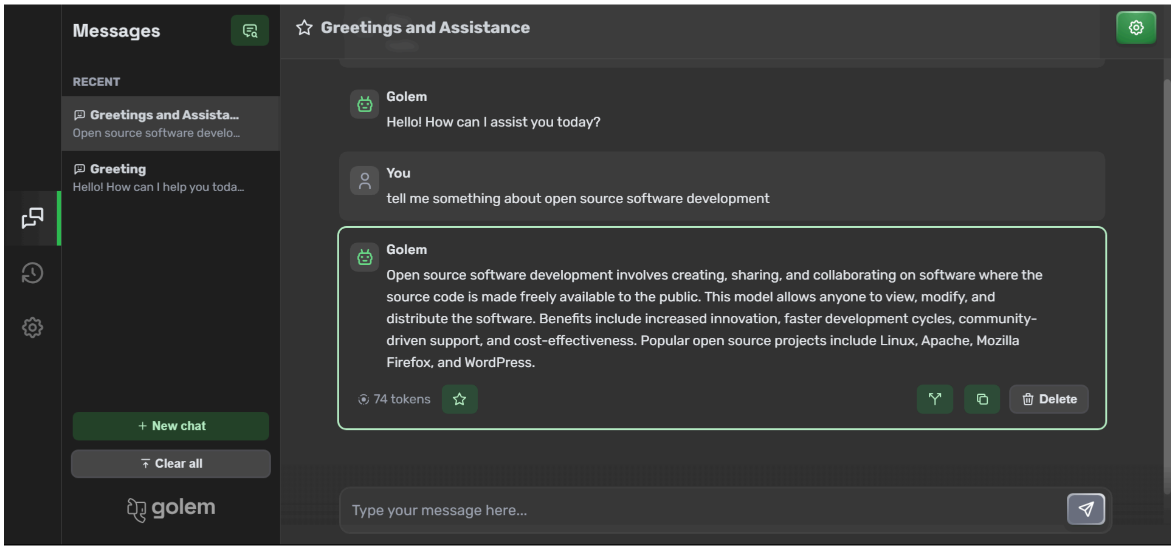

prompt-editor.bot

The prompt **<prompt_name>** has been deleted successfully

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

6. Design Integration

Golem UI Integration: The app leverages Golem, an open-source conversational UI, providing a seamless and intuitive interface for interacting with LLMs.



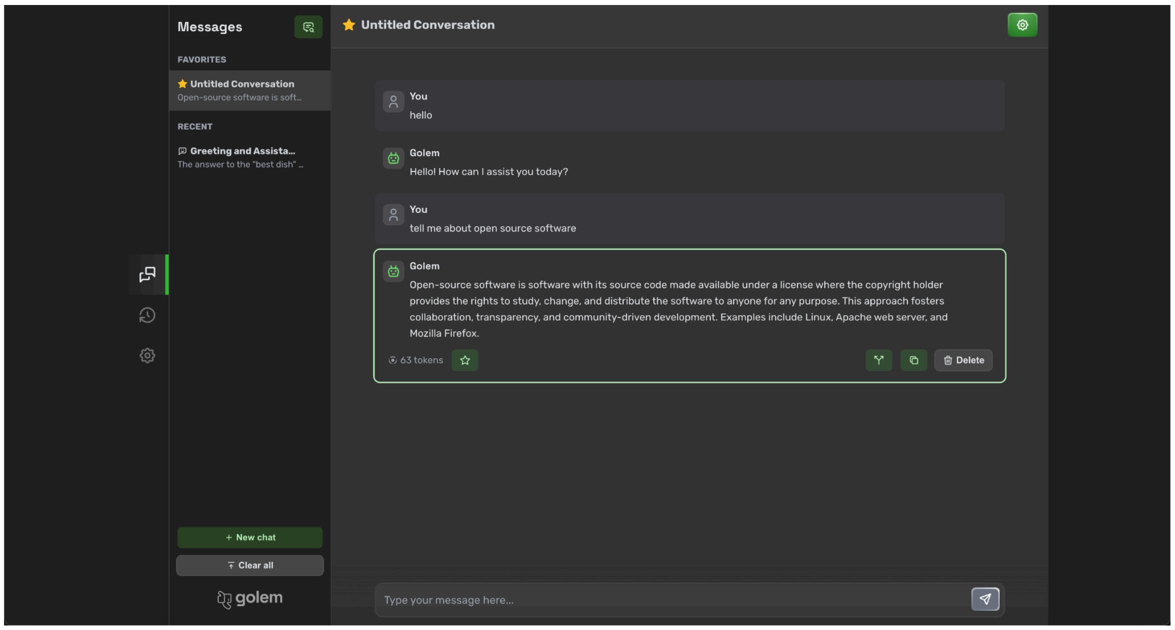
Greetings and Assista...
Open source software develo...

Greeting
Hello! How can I help you toda...

Golem
Hello! How can I assist you today?

You
tell me something about open source software development

Golem
Open source software development involves creating, sharing, and collaborating on software where the source code is made freely available to the public. This model allows anyone to view, modify, and distribute the software. Benefits include increased innovation, faster development cycles, community-driven support, and cost-effectiveness. Popular open source projects include Linux, Apache, Mozilla Firefox, and WordPress.



Untitled Conversation
Open-source software is soft...

Greeting and Assista...
The answer to the "best dish" ...

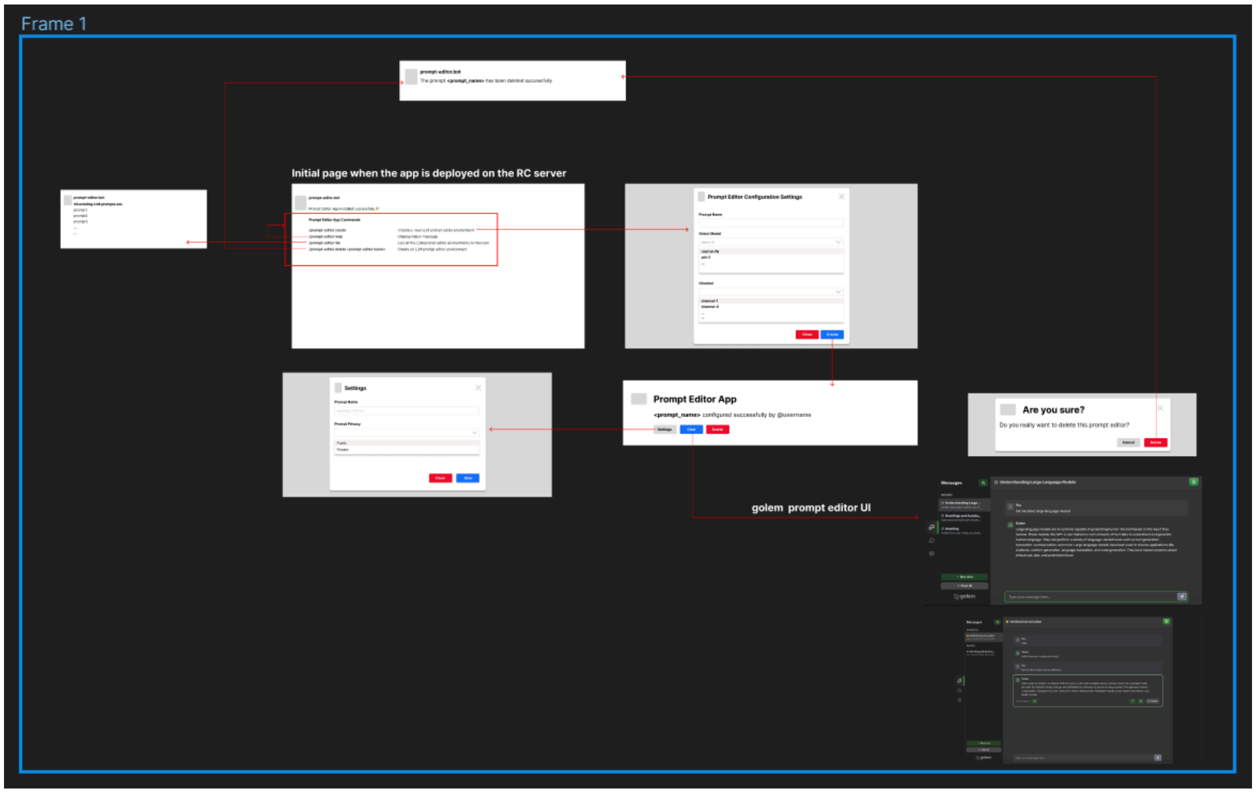
You
hello

Golem
Hello! How can I assist you today?

You
tell me about open source software

Golem
Open-source software is software with its source code made available under a license where the copyright holder provides the rights to study, change, and distribute the software to anyone for any purpose. This approach fosters collaboration, transparency, and community-driven development. Examples include Linux, Apache web server, and Mozilla Firefox.

Figma Designs: The app's user interface is designed with Figma, which aligns with Rocket.Chat's aesthetics and the Golem conversational UI.


Frame 1
Initial page when the app is deployed on the RC server
Prompt Editor Configuration Settings
Settings
Prompt Editor App
Are you sure?
Do you really want to delete this prompt editor?
golem prompt editor UI

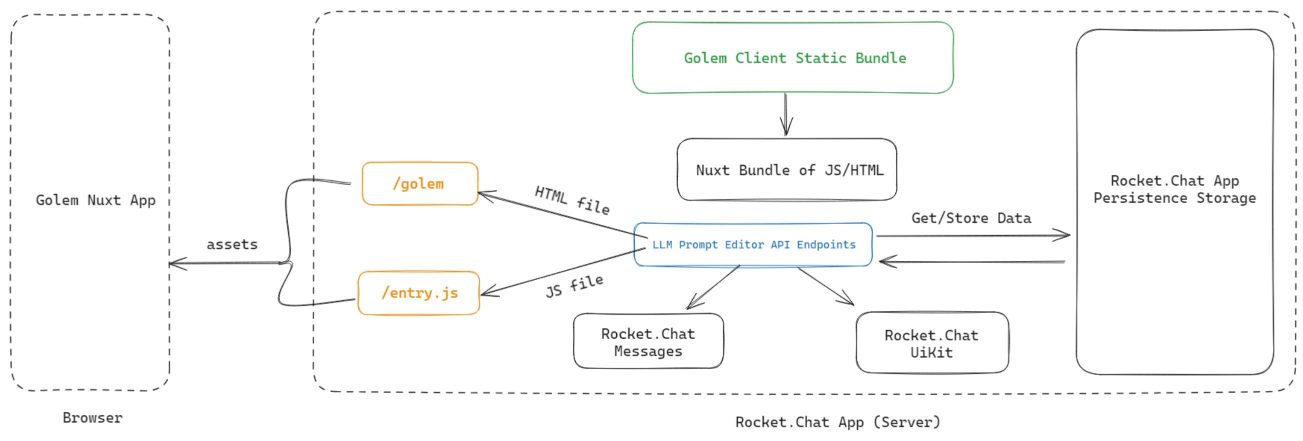
The basic flow of the extended LLM prompt editor/explorer app through figma design can be accessed here - [Extended LLM Prompt Editor/Explorer Figma design]

Prompt Editor Chat User Interface

To integrate a User Interface for the extended LLM prompt editor within the user's workspace, we can build on top of ***Golem***, which will be the client app. So that the user doesn't have to rely on any third party sources, this MIT-Licensed open source conversational UI is considered.
The main code can be accessed here - **Golem Prompt Editor UI**
Using this existing solution, no quick setup is required.

Architecture of App

The following **link** presents the architecture of the app.



Golem Open-Source Conversational UI

Golem Chat Solution helps to share conversations, secure data, and customize experience. It is built using the Nuxt framework which is an open source JavaScript library based on Vue.js, Nitro, and Vite.
It is an open-source conversational UI designed to provide an alternative to ChatGPT. It aims to offer a more interactive and engaging user experience for conversing with AI models. Golem focuses on creating a platform that allows users to interact with AI in a more natural and intuitive way, making it easier to engage with AI-driven applications and services.

Bundling the client app

The bundling of the client app can be done with module bundlers available. This will be one of the major tasks for this app since after then only it can be compressed.
I Researched about module bundlers **here**

After researching I found out that ***webpack*** would be the most suitable one. Detailed info **here**
But since Nuxt already uses webpack behind the scenes, It can be configured in its ***nuxt.config.ts*** file just like how ***webpack.config.js*** is configured. More on **Nuxt documentation**.

```
const path = require('path');
const webpack = require('webpack');
const { VueLoaderPlugin } = require('vue-loader');
const BundleAnalyzerPlugin = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  mode: 'development',
   entry: {
      bundle: path.resolve(__dirname, './src/main.js'),
   },
   output: {
      path: path.resolve(__dirname, 'dist'),
      filename: '[name].js',
      clean: true,
   },
   module: {
      rules: [
         {
            test: /\.vue$/,
            use: [
               'vue-loader',
            ],
         },
         {
            test: /\.scss$/,
            use: ['style-loader', 'css-loader', 'sass-loader'],
         },
         {
            test: /\.css$/,
            use: [
               'style-loader',
               'css-loader',
            ],
         },
         {
            test: /\.js$/,
            exclude: /node_modules/,
            use: {
               loader: 'babel-loader',
               options: {
                  presets: ['@babel/preset-env'],
               },
            },
         },
         {
            test: /\.(png|svg|jpg|jpeg|gif)$/i,
            type: 'asset/resource',
         },
      ],
   },
   plugins: [
      new VueLoaderPlugin(),
      new BundleAnalyzerPlugin(),
      new HtmlWebpackPlugin({
         title: 'Vue Webpack App',
         filename: 'index.html',
         template: './public/index.html',
         templateParameters: {
            BASE_URL: process.env.BASE_URL || 'http://localhost:9000',
         },
      }),
      new webpack.DefinePlugin({
         'process.env.BASE_URL': JSON.stringify(process.env.BASE_URL ||
'http://localhost:9000'),
      }),
   ],
   devServer: {
      static: {
         directory: path.resolve(__dirname, 'dist'),
      },
      compress: true,
      hot: true,
      port: 9000,
   },
}
```
This is how webpack.config.js would be configured.

```
export default defineNuxtConfig({
    pwa: {
        icon: {
            fileName: 'android-chrome-512x512.png',
        },
        manifest: {
            background_color: '#f5f5f5',
            name: 'Golem',
            categories: ['productivity', 'education'],
            description: 'Golem is an open-source, amazingly crafted conversational
UI and alternative to ChatGPT.',
            display: 'standalone',
            lang: 'en',
            id: `golem-${new Date().getTime()}`,
            theme_color: '#3f3f3f',
        },
        meta: {
            ogDescription: 'Golem is an open-source, amazingly crafted conversational
UI and alternative to ChatGPT.',
            ogTitle: 'Golem',
            ogHost: 'https://golem.chat/',
            ogImage: '/og-image.png',
            ogUrl: 'https://golem.chat/',
            title: 'Golem',
            author: 'Henrique Cunha',
            description: 'Golem is an open-source, amazingly crafted conversational
UI and alternative to ChatGPT.',
            lang: 'en',
            ogSiteName: 'app.golem.chat',
            twitterCard: 'summary_large_image',
            twitterSite: 'golem.chat',
            twitterCreator: '@henrycunh',

        },
    },

    css: ['~/assets/css/main.css'],
    experimental: {
        reactivityTransform: true,
    },
    modules: [
        '@unocss/nuxt',
        '@vueuse/nuxt',
        '@nuxtjs/color-mode',
        '@kevinmarrec/nuxt-pwa',
    ],
    colorMode: {
        classSuffix: '',
    },
    unocss: {
        attributify: true,
        uno: true,
        icons: true,
        webFonts: {
            fonts: {
                code: 'DM Mono:400',
                text: 'Rubik:400,700',
                title: 'Space Grotesk:400,700',
            },
        },
    },
    vite: {
        define: {
            'process.env.VSCODE_TEXTMATE_DEBUG': 'false',
            '__DEV__': process.env.NODE_ENV === 'development',
        },
    },
    hooks: {
        'vite:extendConfig': function (config, { isServer }) {
            if (isServer) {
                const alias = config.resolve!.alias as Record<string, string>
                for (const dep of ['shiki-es', 'fuse.js']) {
                    alias[dep] = 'unenv/runtime/mock/proxy'
                }
            }
        },
    },
    app: {
        pageTransition: { name: 'page', mode: 'out-in' },
    },
    runtimeConfig: {
        detaKey: process.env.DETA_PROJECT_KEY,
        apiKey: process.env.OPENAI_API_KEY,
        password: process.env.GOLEM_PASSWORD,
    },
    build: {
        transpile: ['trpc-nuxt', 'dexie'],
    },
})
```

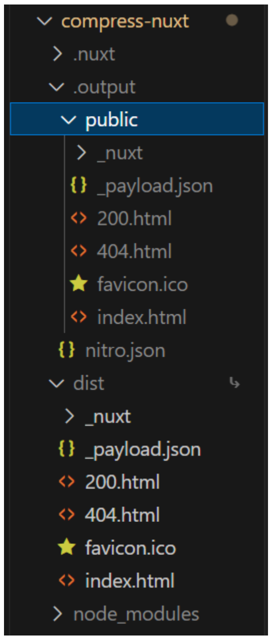
This is how nuxt.config.ts is configured in the golem app.

The script sections in ***package.json*** provide various functions:

1. **build (*nuxt build*):** Compiles the Nuxt.js app for production, optimizing it for performance.
2. **dev (*nuxt dev*):** Starts the app in development mode with hot module replacement for live reloading.
3. **generate (*nuxt generate*):** Creates a static version of the app, useful for static site generation.
4. **preview (*nuxt preview*):** Serves the static site generated by generate for previewing.
5. **postinstall (*nuxt prepare*):** Intended to run after package installation, but in this case, it seems to be a custom command.
6. **prepare (*esno scripts/prepare.ts*):** Runs a TypeScript script to copy assets from *node_modules* to the project's public directory, ensuring they are included in the build.

The ***'pnpm run generate'*** creates a static version of the app, and those statically generated files will then be compressed.

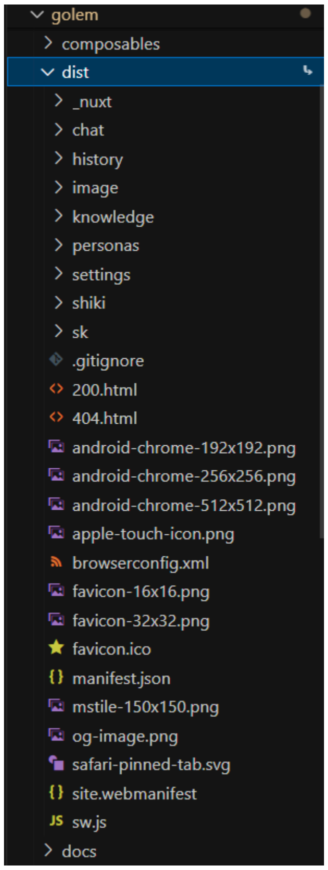
I tried bundling both - **‘A sample nuxt app’**, and the **‘golem nuxt app’.**
These are the results:



```
compress-nuxt
  .nuxt
  .output
    public
      _nuxt
      _payload.json
      200.html
      404.html
      favicon.ico
      index.html
    nitro.json
  dist
    _nuxt
    _payload.json
    200.html
    404.html
    favicon.ico
    index.html
  node_modules
```

Static build files of the sample nuxt app.

Static build files of golem app



```
golem
  composables
  dist
    _nuxt
    chat
    history
    image
    knowledge
    personas
    settings
    shiki
    sk
    .gitignore
    200.html
    404.html
    android-chrome-192x192.png
    android-chrome-256x256.png
    android-chrome-512x512.png
    apple-touch-icon.png
    browserconfig.xml
    favicon-16x16.png
    favicon-32x32.png
    favicon.ico
    manifest.json
    mstile-150x150.png
    og-image.png
    safari-pinned-tab.svg
    site.webmanifest
    sw.js
  docs
```

Compressing the bundled app

After bundling the client app, it needs to be compressed in a string to decode through the Rocket.Chat endpoint.
Brotli algorithm is used to compress the files to a string of *base-64* format.

Why Brotli?
Compression Ratio: *Brotli generally offers better compression ratios than Gzip, which can lead to faster download times and reduced bandwidth usage.*
Browser Support: *Brotli is supported by most modern browsers, including Chrome, Firefox, and Edge. However, it's not as widely supported as Gzip, especially among older browsers.*
Use Case: *Brotli is suitable for compressing text-based content, such as HTML, CSS, and JavaScript files, as well as binary files like images and fonts.*

Compressing Sample nuxt app

A ***build.js*** file in the ***./scripts*** directory is written to compress the desired bundled files.

```
import { readFileSync, writeFileSync } from "fs";
import { compress } from "brotli";

let jsCode = readFileSync("./dist/_nuxt/D3rLC7TF.js", "utf8");

let htmlBuffer = Buffer.from(jsCode, "utf8");

let compressed = compress(htmlBuffer, {
   mode: 0,
   quality: 11,
   lgwin: 22,
});

let base64Compressed = `export const compressedNuxtString=\`${Buffer.from(
   compressed
).toString("base64")}\``;

let compressedHtml = `export const compressedNuxtContent=\`${readFileSync(
   "./dist/index.html",
   "utf8"
)}\``;

writeFileSync("../prompteditor/assets/compressedNuxtContent.ts", compressedHtml);
writeFileSync("../prompteditor/assets/compressedNuxt.ts", base64Compressed);
```

```
console.log("Built at", new Date().toLocaleString());
```

Here's a breakdown of what each part of the script does:

1. The script imports *readFileSync* and *writeFileSync* from the *fs* module for reading and writing files, and compresses from the ***brotli*** module for compressing data.
2. It reads the content of a JavaScript file located at *./dist/{bundle}.js* using *readFileSync*. The content is stored in the jsCode variable.
3. The JavaScript code is converted into a Buffer object using *Buffer.from(jsCode, "utf8")*. This is necessary because the compress function expects a Buffer as input.
4. The JavaScript code is compressed using the Brotli algorithm with specific compression settings. The compressed data is stored in the compressed variable.
5. This data is converted to a Base64 string using *Buffer.from(compressed).toString("base64")*. This string is then wrapped in a TypeScript export statement and stored in the *base64Compressed* variable.
6. The script reads the content of the index.html file located at *./dist/index.html* using *readFileSync*. The content is stored in the compressedHtml variable.
7. The HTML content is wrapped in a TypeScript export statement and stored in the compressedHtml variable.
8. The script writes the Base64-encoded compressed JavaScript code to a TypeScript file named *compressedNuxt.ts* located at *../prompteditor/assets/*.
9. The script writes the HTML content wrapped in a TypeScript export statement to a TypeScript file named *compressedNuxtContent.ts* located at *../prompteditor/assets/*.
10. Finally, the script logs the current date and time to the console, indicating when the script finished executing.

Then after bundling, this file is executed through the scripts section

```
"build:nuxt": "node scripts/build.js",
```

Compressing Golem app

A ***build.js*** file in the ***./scripts*** directory is written to compress the desired bundled files of the golem app.

```
import { readFileSync, writeFileSync } from "fs";
import { compress } from "brotli";

let jsCode = readFileSync("./dist/_nuxt/entry.ea1d6c59.js", "utf8");
let htmlBuffer = Buffer.from(jsCode, "utf8");

let compressed = compress(htmlBuffer, {
   mode: 0,
   quality: 11,
   lgwin: 22,
});

let base64Compressed = `export const compressedGolemString=\`${Buffer.from(
   compressed
).toString("base64")}\``;

let compressedHtml = `export const compressedGolemContent=\`${readFileSync(
   "./dist/404.html",
   "utf8"
)}\``;

writeFileSync("../prompteditor/assets/compressedGolemContent.ts", compressedHtml);
writeFileSync("../prompteditor/assets/compressedGolem.ts", base64Compressed);

console.log("Built at", new Date().toLocaleString());
```

1. The script imports *readFileSync* and *writeFileSync* from the fs module, which are used for reading from and writing to files, respectively. It also imports the compress function from the *brotli* module, which is used for Brotli compression.
2. It reads a JavaScript file located at *./dist/_nuxt/{entry}.{hash}.js* using *readFileSync* and stores its content in the jsCode variable.
3. The content of the JavaScript file is converted into a Buffer object using *Buffer.from(jsCode, "utf8")*. This is done to prepare the data for compression.
4. The compress function from the *brotli* module is used to compress the buffer. The compression parameters are set. These parameters control the compression algorithm's behavior, with mode specifying the compression mode, quality controlling the compression level, and *lgwin* determining the sliding window size.
5. The compressed data is then converted into a base64 string using *Buffer.from(compressed).toString("base64")*. This string is embedded into a JavaScript export statement and stored in the *base64Compressed* variable.
6. The script reads an HTML file located at *./dist/{index}.html* using *readFileSync* and stores its content in the compressedHtml variable.
7. The content of the HTML file is embedded into a JavaScript export statement and stored in the compressedHtml variable.

8. The script writes the *base64-encoded* compressed JavaScript data to a file named compressedGolem.ts in the *../prompteditor/assets/* directory using *writeFileSync*. It also writes the HTML content to a file named *compressedGolemContent.ts* in the same directory.
9. Finally, the script logs the current date and time to the console, indicating when the script was executed.

Then after bundling, this file is executed through the scripts section

```
"build:golem": "node scripts/build.js",
```

Serving the compressed files to Rocket.Chat endpoints

These compressed files are served to the Rocket.Chat endpoint to decode.

Serving compressed 'Sample Nuxt app'

1. **Serving JS file -**

```
export class NuxtBundleJsEndpoint extends ApiEndpoint {
   public path = "bundle.js";

   public async get(
      request: IApiRequest,
        endpoint: IApiEndpointInfo,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persis: IPersistence,
   ): Promise<IApiResponse> {
      const content = Buffer.from(compressedNuxtString, "base64");
      return {
        status: 200,
        headers: {
            "Content-Type": "text/javascript",
            "Content-Encoding": "br",
        },
        content,
      }
   }
}
```

2. **Serving HTML files -**

```
export class NuxtEndpoint extends ApiEndpoint {
    public path = "nuxt";

    public async get(): Promise<IApiResponse> {
        const content = compressedNuxtContent;
        return {
            status: 200,
            headers: {
                "Content-Type": "text/html",
                "Content-Security-Policy":
                        "default-src 'self' http: https: data: blob: 'unsafe-inline'
'unsafe-eval'",
            },
            content,
        };
    }
}
```

Serving compressed 'Golem app'

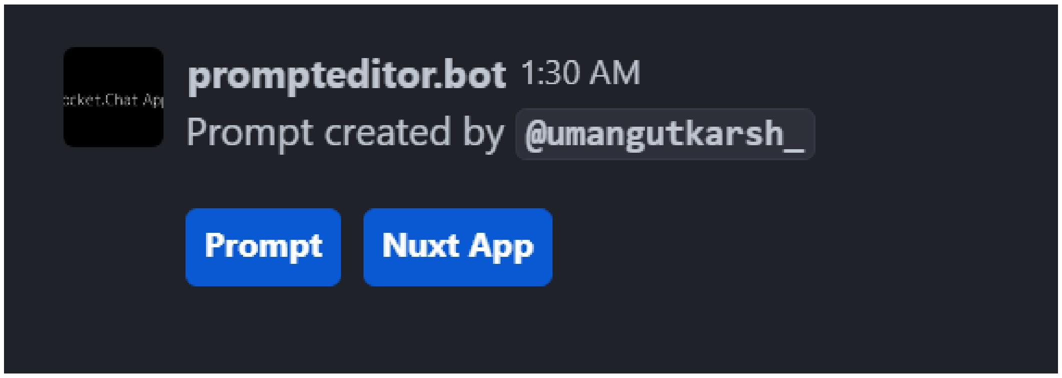
1. **Serving JS file -**

```
export class GolemBundleJsEndpoint extends ApiEndpoint {
   public path = "entry.js";

   public async get(
      request: IApiRequest,
        endpoint: IApiEndpointInfo,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persis: IPersistence,
   ): Promise<IApiResponse> {
      const content = Buffer.from(compressedGolemString, "base64");
      return {
        status: 200,
        headers: {
            "Content-Type": "text/javascript",
            "Content-Encoding": "br",
        },
        content,
      }
   }
}
```

2. Serving HTML files

```
export class GolemEndpoint extends ApiEndpoint {
    public path = "golem";

    public async get(): Promise<IApiResponse> {
        const content = compressedGolemContent;
        return {
            status: 200,
            headers: {
                "Content-Type": "text/html",
                "Content-Security-Policy":
                        "default-src 'self' http: https: data: blob: 'unsafe-inline'
'unsafe-eval'",
            },
            content,
        };
    }
}
```

These are then provided to the required action button.


promptteditor.bot 1:30 AM
Prompt created by @umangutkarsh_

A small demonstration can be viewed **here**.

ISSUES:

1. **The sample nuxt app is served and decoded and apart from some css issues (which might be due to decoding issues or bundling/compression issues) is served,**

2. **The Golem app is served properly since after building, it is not clear which files need to be compressed, and what is the main entry point of the bundle. Hence the golem client is not opened after the Action-Button is clicked.**

Here, I have also used 'http-server' which is a simple zero-configuration command-line HTTP-server to serve the static site locally.

This approach can be followed if the proper files are built and bundled, then the correct compression algorithm is used to compress the nuxt golem app and server to Rocket.Chat endpoints.

This is the code link to my approach - **code**

I have also worked on learning Vue.js, and Nuxt framework to work and make any changes to the code if required - here

Building UI Blocks with Ui-Kit

The User Interface (modals, contextual bars, buttons) of the Rocket.Chat app would be built using the *@rocket.chat/ui-kit.* Some example blocks are:

```
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 getChannelSelectElement(
    appId: string,
    blockId: string,
    actionId: string,
) {
    const block: ChannelsSelectElement = {
        type: "channels_select",
        appId,
        blockId,
        actionId,
    };
    return block;
}
```

```
export function getButton(
    labelText: string,
    blockId: string,
    actionId: string,
    appId: string,
    value?: string,
    style?: ButtonStyle.PRIMARY | ButtonStyle.DANGER,
    url?: string,
) {
    const button: ButtonElement = {
        type: "button",
        text: {
            type: "plain_text",
            text: labelText,
            emoji: true,
        },
        appId,
        blockId,
        actionId,
        url,
        value,
        style,
        secondary: false,
    };
    return button;
}
```

Interacting with large language models

To interact with open source LLMs, some changes would be required in some files of the nuxt golem app. Specifically in the files - *language-model.ts, message.post.ts* to modify the CHAT_COMLETION_ENDPOINT

```
        const CHAT_COMPLETION_ENDPOINT = 'https://api.openai.com/v1/chat/completions'
```

Features of Prompt Editor

The basic features mentioned in the project idea like saving, deleting, renaming the prompts, and copying the text to clipboard are provided by the golem's open source solution.
A demo of the features can be viewed here - **Features of Prompt Editor**

Workflow Timeline

Application Review Period (April 2 - May 1)
During this period, my focus will be on:

1. Understanding best practices for app development with Rocket.Chat. Leveraging TypeScript for the Extended LLM Prompt Editor App.
2. Identifying correct files for bundling and compression.
3. Research on Compressing Nuxt Apps
4. Exploring LLM APIs.
5. Actively participating in the Rocket.Chat community, sharing insights, and assisting others.
6. Contributing to other Rocket.Chat projects.

Community Bonding Period (May 1 - May 26)
During this period, my focus will be on:

1. Establishing communication with mentors for app development strategies.
2. Deepening engagement with the Rocket.Chat community to understand user needs.
3. Conducting research on the Rocket.Chat core codebase and Nuxt apps for best practices.
4. Identifying the proper compression procedure for the Golem Nuxt app.
5. Exploring large language models for effective interaction.

6. Reviewing the Golem Nuxt app codebase for integration.
7. Refining app features and user experience based on feedback.

Week 1 (May 27 - June 2)

- Setting up the basic structure of the LLM prompt editor app, including initializing the app and preparing for further development.
- Work on basic building blocks

Week 2 (June 3 - June 9)

- Work on building and bundling the static build files of the *nuxt golem app*.
- Work on compressing the built files using an appropriate compressor.
- Serving the build to a static-server to test out.

Week 3 (June 10 - June 16)

- Work on serving the compressed files to a Rocket.Chat server endpoint.
- Work on building blocks for the user to interact with the app.

Week 4 (June 17 - June 23)

- Keep this week to make changes to the user interface, Building the modals, contextual bars, and other block elements.

Week 5 (June 24 - June 30)

- Creating different commands for the users to interact, for example - *slash commands.*
- Building of various handlers to interact with the prompt editor app.

Week 6 (July 1 - July 7)

- Work on creating different utility functions for example - *sendMessage(), notifyUser(), sendDirect().*
- Setting up basic persistence storage.

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)

- Will work further on the persistence storage to store and retrieve various prompts created by the user. Also work on providing configuration settings
- Work on the permission mechanism of the app, so that not all users are able to access or view a prompt created by another user.
- Setting up endpoints and serving those endpoints to integrate the final prompt editor (golem).

Week 9 (July 22 - July 28)

- Make modifications in the code base of the *golem nuxt app* so as to suit the needs of the Rocket.Chat prompt-editor app.
- Use open source LLMs, so that the prompt editor is able to interact with them.

Week 10 (July 29 - August 4)

- Work more on the codebase of golem to make it suited to work with the app.

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

- Will work on the documentation part, and create a summary of the work done during the GSoC period.
- If the above work gets completed in the stipulated time-frame, then will work on some of the extra features for the app.
- Complete any incomplete (if any) and extra work required for the app.

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.

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. **Cross-Platform Compatibility:** Expand the app's compatibility beyond the RC Web App and Electron to include mobile platforms (iOS and Android). This would make the prompt editor accessible to a wider audience, regardless of their device.
2. **Integration with additional LLMs:** Enhance the app to support additional open-source LLMs beyond the initially mentioned ones (Mistral, Llama 2, Phi). This could include integrating with new LLMs as they become available or popular.

Relevant Experiences

I have been consistently contributing to Rocket.Chat, and engaging with community members for the past 6 months now, and so I have become attached with Rocket.Chat 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. This web app provides all functionality to set up a datacenter in a city, and configuring servers and creating clusters within those datacenters.
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

The reason these projects are relevant is because each of them have different techstacks and working on different technologies prove that I will be able to handle the development of the Rocket.Chat app and the golem nuxt app, which uses a completely different technology.
'

Why do I find myself suited for this project?

Over the past few months, I've dedicated my time to contribute to Rocket.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.

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. Specifically, I've explored various Rocket.Chat apps like **EmbeddedChat**, **Apps.Whiteboard**, **Apps.Notion**, **Apps.Github22**, **Rocket.Chat.Demo.App**, and **Apps.RocketChat.Tester**.

My growing affinity for Rocket.Chat app development, fueled by consistent contributions, has deepened my familiarity with Rocket.Chat's API, SDK, and **developer documentation**, including the **Rocket.Chat Apps TypeScript Definition**, rocket.chat apps-cli, and apps-engine, equipping me to drive improvements in the 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 myself for 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.