News Aggregation Rocket.Chat 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

News - Aggregation Rocket.Chat App

Discover the latest news from your favorite sources, right within Rocket.Chat, with the News Aggregation App.

Abstract

The News-Aggregation project aims to create a Rocket.Chat app that aggregates news from top websites like TechCrunch, using different methods to collect data from sources with APIs or RSS feeds. The app will display news in a user-friendly format, with complete configurability of sources and categories. The project focuses on Rocket.Chat app development, requiring expertise in this area. The goal is to deliver a fully functional app that provides users with a daily news feed sourced from the Internet. The technical challenge involves implementing logic to collect news from various sources, categorizing them, and presenting them in an organized format within Rocket.Chat. This project aims to enhance the Rocket.Chat ecosystem by offering a valuable tool for accessing curated news content seamlessly within the platform.

Benefits to the community

The News-Aggregation App offers significant benefits to the Rocket.Chat community by providing a centralized platform for seamless access to curated news content within the chat environment. It will streamline the process of staying updated on current events without leaving the Rocket.Chat platform, eliminating the need to visit multiple news websites and saving time.

The app will promote community engagement by allowing users to customize their news feed according to their interests, enhancing user satisfaction and retention. Its development and popularity will contribute to the growth and recognition of Rocket.Chat, strengthening its position as a versatile communication platform.

Goals

During the GSoC period, I will focus on:

1. Developing a news collection mechanism for gathering articles from sources like TechCrunch and BBC News, using different logic for APIs and RSS feeds.
2. Implementing a categorization system for organizing articles based on predefined or user-defined tags, ensuring flexibility and customization.
3. Designing a user-friendly interface for presenting articles in a "newspaper" or "news feed" format within Rocket.Chat, including article summaries and links to full content.
4. Ensuring complete configurability of sources and categories for user customization.

5. Documenting the development process for easy adoption and integration by the Rocket.Chat community.

Deliverables

By the end of the GSoC period, I aim to deliver the following features within the News Aggregation App:

1. Implementation of a robust news collection mechanism capable of gathering articles from various sources such as TechCrunch and other top websites, utilizing different logic to collect news from sources with APIs and RSS feeds.

2. Development of a user-friendly interface for presenting news articles in a "newspaper" or "news feed" format within the Rocket.Chat environment, including summaries of articles and links to full content.

3. Incorporation of a categorization system to organize news articles based on predefined categories or user-defined tags, providing flexibility and customization for users.

4. Complete configurability of news sources and categories, allowing users to customize their news feed according to their preferences and interests.

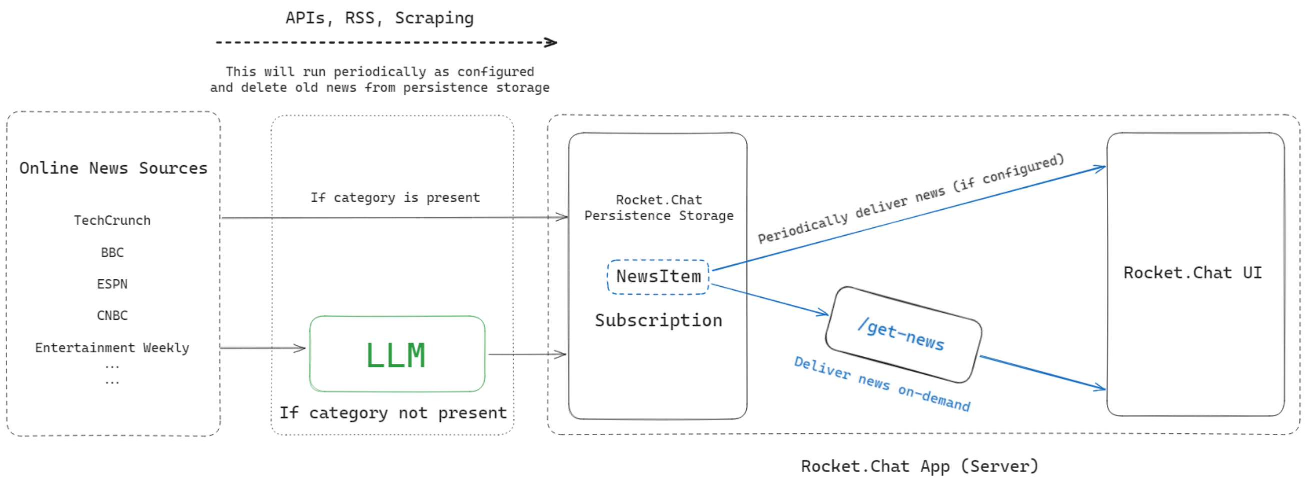
These deliverables aim to provide users with a comprehensive News Aggregation App that offers convenient access to daily news from various sources within the Rocket.Chat ecosystem.

Implementation Details

Proof of Concept

For the proof of concept, I have created a demo of the news-aggregator app to fetch, categorize, and display the news on the rocket.chat instance.
The UI will be improved using Rocket.Chat UiKit during the GSoC tenure.
The demo-code can be accessed here - **News-Aggregation Rocket.Chat App**
The demo video can be accessed here - **Demo**

App Architecture



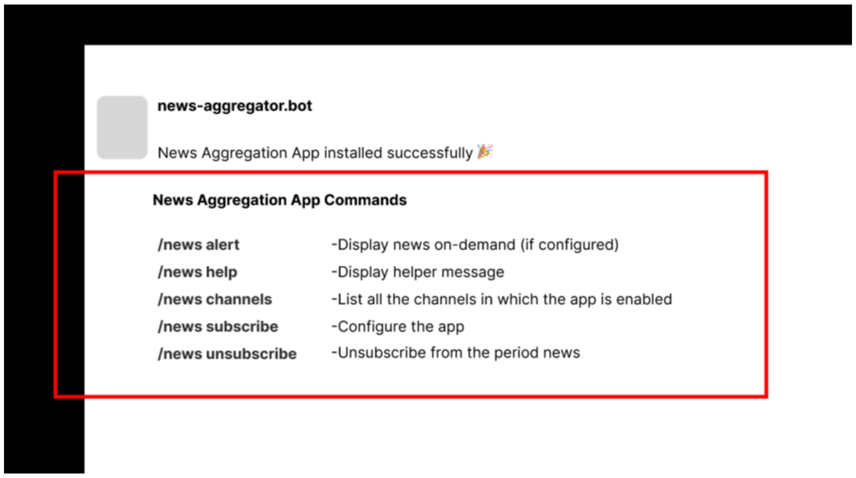
This **link** presents the architecture of the app.

News Aggregator App Flow

The basic flow of the News Aggregation App for Rocket.Chat is detailed below.

1. Installation and Initial Setup

Upon Installation: The app sends 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.



news-aggregator.bot

News Aggregation App installed successfully 🎉

News Aggregation App Commands

/news alert	-Display news on-demand (if configured)
/news help	-Display helper message
/news channels	-List all the channels in which the app is enabled
/news subscribe	-Configure the app
/news unsubscribe	-Unsubscribe from the period news

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

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

    }
```

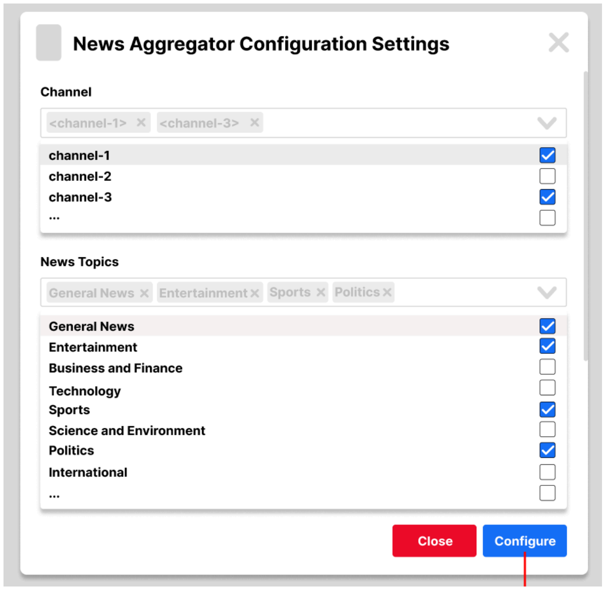
```
public async initialize(configurationExtend: IConfigurationExtend, environmentRead:
IEnvironmentRead): Promise<void> {

    }
```

2. Slash Commands Overview

/news alert: Allows users to request news on-demand, providing them with the latest news summaries.
/news help: Offers information about the app, its features, and how to configure it.
/news channels: Lists all channels where the app is configured.
/news subscribe: Enables users to configure the app through a modal by specifying channels and news topics of interest.
/news unsubscribe: Allows users to unsubscribe from receiving periodic news updates.
The commands can be changed / added as the app progresses.

3. Configuring the App

Configuration Modal: After installation, the app requires configuration. Users can specify channels and news topics they're interested in through a configuration modal.



News Aggregator Configuration Settings

Channel

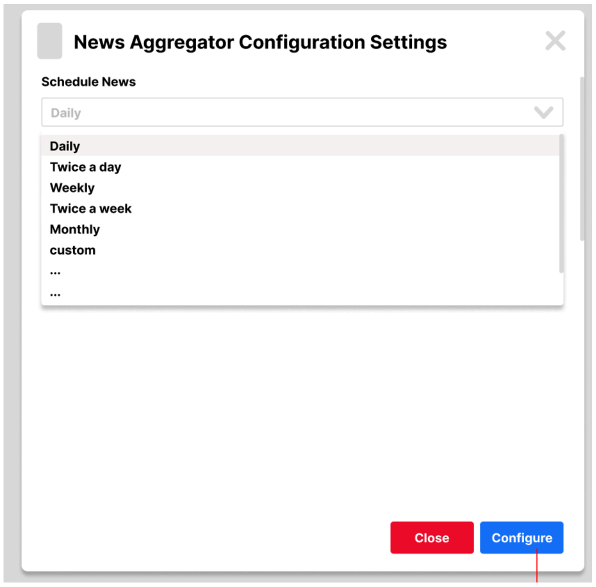
<channel-1> × <channel-3> ×

channel-1 ☑
channel-2 ☐
channel-3 ☑
... ☐

News Topics

General News × Entertainment × Sports × Politics ×

General News ☑
Entertainment ☑
Business and Finance ☐
Technology ☐
Sports ☑
Science and Environment ☐
Politics ☑
International ☐
... ☐



News Aggregator Configuration Settings

Schedule News

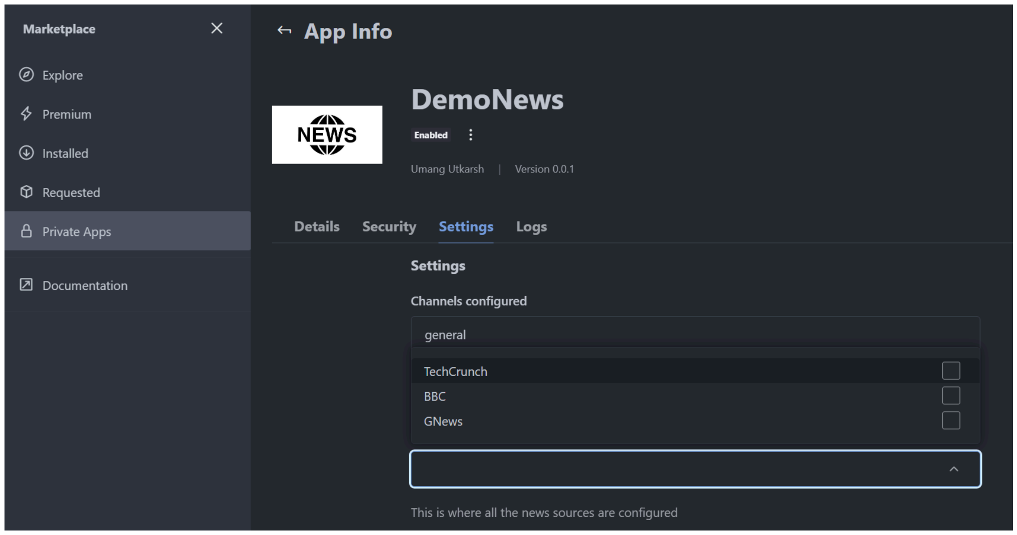
Daily

Daily
Twice a day
Weekly
Twice a week
Monthly
custom
...
...

Code snippets are shown to demonstrate how a modal would be configured.

```
export async function SettingsModal(
    appId: string,
    messageId: string
): Promise<IUIKitSurfaceViewParam> {
    const block: Block[] = [];

    /* For Settings Text block */
    let settingsTextBlock = getSectionBlock(UtilityEnum.MESSAGE_LABEL);
    block.push(settingsTextBlock);

    /* For Board Input block */
    let boardInputBlock = getInputBox(
        UtilityEnum.INPUT_LABEL,
        UtilityEnum.INPUT_PLACEHOLDER,
        UtilityEnum.MESSAGE_INPUT_BLOCK_ID,
        UtilityEnum.MESSAGE_INPUT_ACTION_ID,
        appId
    );
    block.push(boardInputBlock);

    let options: Array<Option> = [
        {
            text: {
                type: "plain_text",
                text: "Message",
            },
            value: UtilityEnum.MESSAGE,
        },
        // other options...
    ];


    let MultiStaticSelectElement = getMultiStaticSelectElement(
        UtilityEnum.MESSAGE_SELECT_LABEL,
        options,
        ...other_params,
    );

    let MultiChoiceBlock: InputBlock = {
        type: "input",
        label: {
            type: "plain_text",
            text: UtilityEnum.MESSAGE_TYPE_LABEL,
        },
        element: MultiStaticSelectElement,
    };
    block.push(MultiChoiceBlock);

    let closeButton = getButton(
        UtilityEnum.CANCEL,
        UtilityEnum.CLOSE_BLOCK_ID,
        UtilityEnum.CLOSE_ACTION_ID,
        ...other_params,
    );

    // Event handling for closing modal
    closeButton.actionId = UtilityEnum.CLOSE_ACTION_ID;

    let submitButton = getButton(
        UtilityEnum.SEND,
        UtilityEnum.SEND_BLOCK_ID,
        UtilityEnum.SEND_ACTION_ID,
        appId,
        messageId,
        ButtonStyle.PRIMARY
    );

    const value = {
        id: UtilityEnum.SETTINGS_MODAL_ID,
        type: UIKitSurfaceType.MODAL,
        appId: appId,
        title: {
            type: "plain_text" as const,
            text: UtilityEnum.MESSAGE_MODAL_TITLE,
        },
        close: closeButton,
        submit: submitButton,
        blocks: block,
    };
    return value;
}
```

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



DemoNews

Umang Utkarsh | Version 0.0.1

Settings

Channels configured

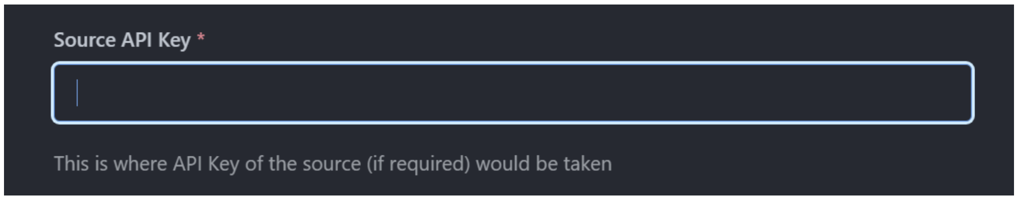
general

TechCrunch

BBC

GNews

This is where all the news sources are configured



Source API Key *

This is where API Key of the source (if required) would be taken

A demo code snippet of configuring settings and API Keys (if required), and accessing the provided key in settings through the `IRead` and providing in the main entry point of the app as shown below.

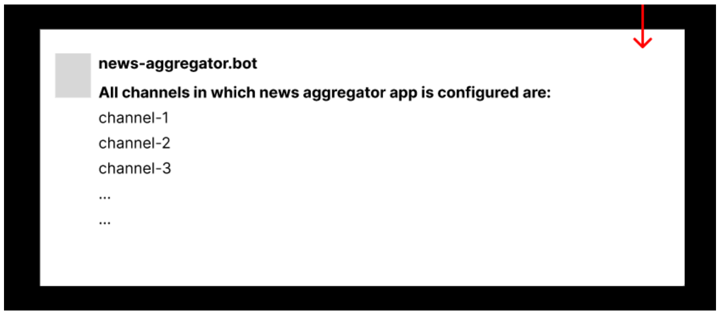
```
export const settings: Array<ISetting> = [
    {
        id: NewsSetting.API_KEY,
        section: "NewsSettings",
        public: false,
        type: SettingType.STRING,
        value: "",
        packageValue: "",
        hidden: false,
        i18nLabel: "Source API Key",
        i18nDescription: "This is where API Key of the source (if required) would be 
taken",
        required: true,
    },
    {
        id: NewsSetting.NewsChannel,
        section: "NewsSettings",
        public: false,
        type: SettingType.STRING,
        value: "general",
        packageValue: "",
        hidden: false,
        i18nLabel: "Channels configured",
        i18nDescription: "This is where all the channels are configured in which the
news-app will work",
        i18nPlaceholder: "Enter allowed channels (Comma Separated)",
        required: false,
    },
    {
        id: NewsSetting.NewsSource,
        section: "NewsSettings",
        public: false,
        type: SettingType.MULTI_SELECT,
        values: [
            {"key": "techcrunch", "i18nLabel": "TechCrunch"},
            {"key": "bbc", "i18nLabel": "BBC"},
            {"key": "gnews", "i18nLabel": "GNews"},
        ],
        packageValue: "",
        hidden: false,
        i18nLabel: "Sources",
        i18nDescription: "This is where all the news sources are configured",
        i18nPlaceholder: "Select news sources",
        required: false,
    },
]
```

```
await Promise.all([
            settings.map((setting) => {
                configuration.settings.provideSetting(setting)
            }),
        ]);
```

```
const apiKey = (await
read.getEnvironmentReader().getSettings().getValueById(NewsSetting.API_KEY)) as
string;
```

Now we can get the required API_KEY in the desired *fetchNews()* function.

More configuration settings can be provided as the app progresses.


news-aggregator.bot

All channels in which news aggregator app is configured are:

channel-1

channel-2

channel-3

...

...

After configuring, the ***/news channels*** slash command will provide the list of the channels in which the app is configured.

4. Daily News Delivery

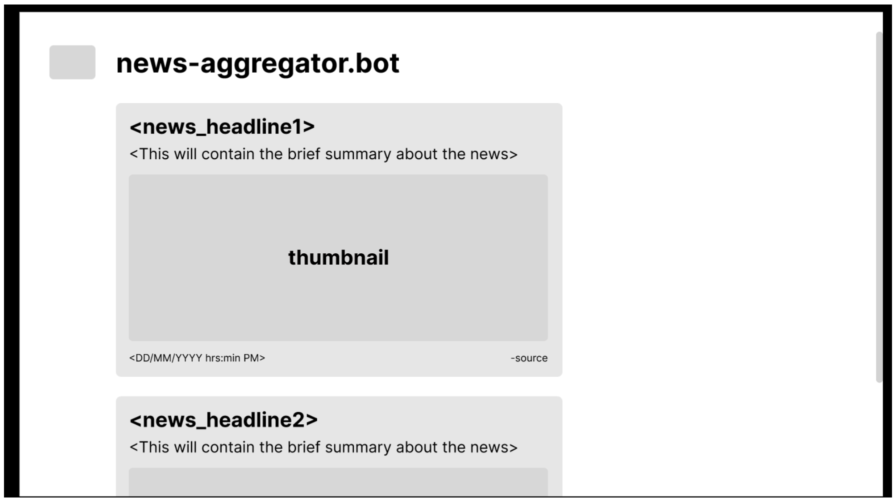
Periodic News Updates: Based on the user's configuration, the app will deliver daily, weekly, or on any other periodic interval news updates to the specified channels. This is demonstrated below using scheduling.

News Summaries and Links: Each news update includes a summary and a link to the full content, allowing users to explore the news in detail. This will be achieved by building various blocks through the *rocket.chat ui-kit.* This is demonstrated below.

5. User Interaction

Slash Commands for Control: Users will interact with the app through slash commands, enabling them to subscribe, unsubscribe, and manage their news preferences.

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.


news-aggregator.bot

<news_headline1>

<This will contain the brief summary about the news>

<DD/MM/YYYY hrs:min PM> -source

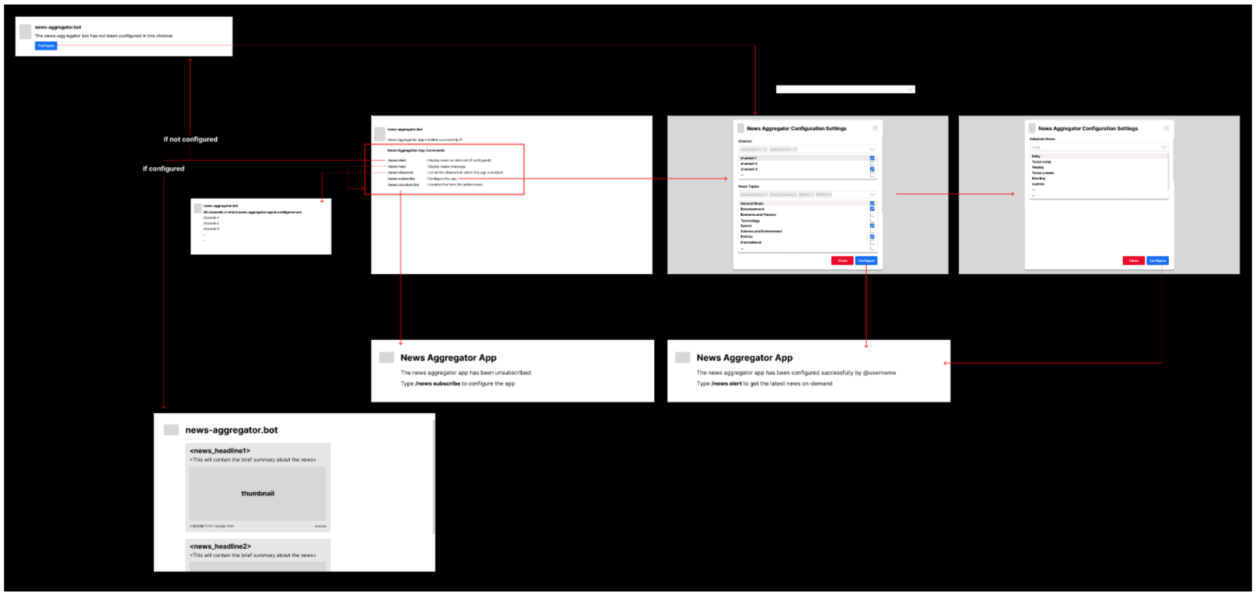
<news_headline2>

<This will contain the brief summary about the news>

Basic outline of how the news will be displayed

6. **Design Integration**

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



news-aggregator.bot
The news-aggregator bot has not been configured in this channel
Configure
if not configured
if configured
News Aggregator Configuration Settings
Close
Configure
thumbnail

News Aggregator App
The news aggregator app has been unsubscribed
Type **/news subscribe** to configure the app

News Aggregator App
The news aggregator app has been configured successfully by @username
Type **/news alert** to get the latest news on-demand

news-aggregator.bot
<news_headline1>
<This will contain the brief summary about the news>
<news_headline2>
<This will contain the brief summary about the news>

The basic flow of the news aggregator app through figma design can be accessed here - News Aggregator Figma design

Fetching news from various sources

1. **General News**
   - **BBC News -** Offers educational content for sharing with proper attribution and a link back.
   - **NPR (National Public Radio) -** Allows sharing of content for non-commercial use with proper attribution.
2. **Business and Finance**
   - **CNBC -** Provides articles and videos for personal or educational use with credit and a link back.
   - **The Economist -** Content is strictly for personal, non-commercial use only.
3. **Technology**
   - **TechCrunch -** Offers articles and blog posts that can be shared with attribution and a link back.
4. **Entertainment and Celebrity News**
   - **Entertainment Weekly -** Allows sharing of content for non-commercial use with proper credit and a link back.

5. **Sports**
    - **ESPN -** Content sharing is allowed for non-commercial purposes with attribution and a link back.
6. **Science and Environment**
    - **National Geographic -** Select content can be shared for educational purposes with proper credit and a link back.
7. **Politics**
    - **Politico -** Allows sharing of content with proper attribution for non-commercial purposes.
8. **Health**
    - **WebMD -** Permits sharing of content for personal, non-commercial use with proper attribution and a link back.
9. **International**
    - **Al Jazeera -** Some content can be shared with credit and a link back, following their specific sharing guidelines.
10. **Investigative Journalism**
    - **ProPublica -** Encourages the sharing and republishing of investigative content with proper credit and a link back.

The following sources would be used to fetch news from the internet, meanwhile I will also search for more promising sources in the community bonding period.
This document contains the sources including APIs, RSS feeds, and scraping - News Sources

Managing multiple news sources

To manage multiple news sources, a NewsSource parent class will be created, with child classes for each source extending NewsSource. These child classes will implement methods like *fetchNews(), saveNews(), and getNews()* specific to each source. Some methods, like *fetchNews()*, will be overridden for unique logic, while others, like *saveNews()*, will remain unchanged as they share the same storage logic. Additional methods can be implemented in child classes as needed.

```
export class NewsSource {
    config;
    news: NewsItem[] = [];
    constructor(config, ...other_params) {
        this.config = config;
    }

    async fetchNews(app: DemoNewsApp, filter = {}) {
        // logic to fetch news from the source
    }

    async saveNews(persistence: IPersistence) {
        // logic to save news to persistence storage
    }

    async getNews(persistence: IPersistence, filter) {
        // logic to get news from persistence storage
        // return the news according to filter
    }
}
```
*Parent Class^*

```
export class TechCrunchNewsSource extends NewsSource {
    news: NewsItem[] = [];
    fetchUrl = `https://techcrunch.com/wp-json/wp/v2/posts`;

    constructor(config) {
        super(config);
    }

    public async fetchNews(http: IHttp, filter?: {}): Promise<void> {
        // logic to fetch news from the techcrunch API
    }

    static async determineCategory(newsItem: NewsItem) {

    }
}
```
Inheritance using parent class demonstrated for `TechCrunchNewsSource` source.

Aggregating News from multiple sources

Fetching news from APIs

Fetching news from APIs involves using news source API endpoints to access and retrieve news articles. This process includes identifying API endpoints, understanding API documentation, sending requests with parameters like news categories, and parsing responses to extract article titles and URLs. APIs are crucial for efficient data retrieval, automating news updates, and customizing news content in news aggregation applications.

A demo implementation for fetching news with APIs is shown below:

```
export class TechCrunchNewsSource extends NewsSource {
    news: NewsItem[] = [];
    fetchUrl = `https://techcrunch.com/wp-json/wp/v2/posts`;

    constructor(config) {
        super(config);
    }

    public async fetchNews(
        app: DemoNewsApp,
        context: SlashCommandContext,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persistence: IPersistence,
        filter?: {}
    ): Promise<void> {
        try {
            const response = await http.get(this.fetchUrl);
            this.news = response?.data;
            console.log('News: ', news);

        } catch (err) {
            app.getLogger().error(`Error while fetching news`);
            console.log(err);
        }
    }

    static async determineCategory(newsItem: NewsItem) {

    }

}
```

The above implementation uses the api-endpoint of TechCrunch for demonstration purposes. The following new item fetch would then be mapped to the desired NewsItem format.

Fetching news from RSS Feeds

Fetching news from RSS feeds involves using the RSS feeds provided by news sources to access and retrieve news articles. This process includes identifying RSS feeds, understanding their structure, fetching them, manually parsing the XML data to extract article titles, descriptions, and URLs, and displaying the news articles to users. This approach is beneficial for Rocket.Chat apps, as it avoids the limitations of using external packages.

A demo implementation of the code to fetch rss-news is given below:

```
interface RssItem {
    id: string,
    title: string,
    description: string,
    link: string,
    publishDate: string,
    image: string;
}

function parseRssItems(xml: string): RssItem[] {
    const items: RssItem[] = [];
    const itemRegex = /<item>([\s\S]*?)<\/item>/g;
    let match: RegExpExecArray | null;
    let id = 1;

    while ((match = itemRegex.exec(xml)) !== null) {
        const item = match[1];
        const titleMatch = item.match(/<title><!\[CDATA\[(.*?)\]\]><\/title>/);
        const descriptionMatch =
item.match(/<description><!\[CDATA\[(.*?)\]\]><\/description>/);
        const linkMatch = item.match(/<link>(.*?)<\/link>/);
        const publishDateMatch = item.match(/<pubDate>(.*?)<\/pubDate>/);
        const imageMatch = item.match(/<media:thumbnail[^>]*url="(.*?)"/);

        if (titleMatch && linkMatch && descriptionMatch && publishDateMatch &&
imageMatch) {
            items.push({
                id: id.toString(),
                title: titleMatch[1],
                description: descriptionMatch[1],
                link: linkMatch[1],
                publishDate: publishDateMatch[1],
                image: imageMatch[1],
            });
            id++;
        }
    }
    return items;
};

async function fetchRssFeed(url: string): Promise<RssItem[]> {
    try {
        const response = await new Promise<string>((resolve, reject) => {
            https.get(url, (res) => {
                let data = '';

                res.on('data', (chunk) => {
                    data+=chunk;
                });

                res.on('end', () => {
                    resolve(data);
                });

                res.on('error', (err) => {
                    reject(err);
                });
            });
        });

        const items = parseRssItems(response);
        return items;
    } catch (error) {
        console.error('Error fetching RSS feed:', error);
        throw error;
    }
}

export class BBCNewsSource extends NewsSource {
    news: NewsItem[] = [];
    fetchUrl = `https://feeds.bbci.co.uk/news/science_and_environment/rss.xml`;

    public async fetchNews(
        app: DemoNewsApp,
        context: SlashCommandContext,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persistence: IPersistence,
        filter?: {}
    ): Promise<void> {
        (async () => {
        try {
            this.news = await fetchRssFeed(this.fetchUrl);
            console.log(items);
        } catch (error) {
            console.error('Error processing RSS feed:', error);
        }
        })();
    }
}
```

The above implementation uses the rss-feed of [BBC] for demonstration purposes.
The RssItem would then be mapped to the desired NewsItem format.

Fetching news through scraping (provided html content is pre-rendered)

Fetching news through scraping involves accessing and extracting news articles from websites without API endpoints or RSS feeds by parsing HTML content. This process includes identifying target websites, understanding their HTML structure, fetching web page content, manually parsing HTML to extract article information, and displaying the news articles to users. Scraping offers flexibility in collecting news from any source, requiring knowledge of HTML and website structure.

Categorizing News

Categorizing news can be approached through both AI-based and non-AI methods. Here's how each method can be implemented:

Non-AI Method: Based on Source

Step 1: Identify the sources from which to fetch news.
Step 2: Assign each source to a predefined category. For example, if the news is being fetched from TechCrunch, it might be categorized under "Technology".
Step 3: When fetching news articles, using the source information to categorize the articles.
Step 4: Display the categorized news articles to the user.

For demonstration, I will use the TechCrunch *API* example. Its response has a “*categories” attribute* for the category field for a particular response object. So we can get the name of the category by adding that *random_number* in the fetchUrl.
Example JSON object of TechCrunch:

```
{
    "id": 123,
    ...
    ...
    ...
    "categories": [577055593], // categoryId
    ...
}
```

```
export class TechCrunchNewsSource extends NewsSource {
    fetchUrl = `https://techcrunch.com/wp-json/wp/v2/posts`;

    constructor(config) {
        super(config);
    }

    public async fetchNews(
        app: DemoNewsApp,
        filter?: {}
        ...other_params,
    ): Promise<void> {
        // logic to fetch news and store in NewsItem format
    }

    static async detemineCategory(newsItem: NewsItem, http: IHttp) {
        return Promise.all(newsItem.categories.map(async categoryId => {
            const response = await
http.get(`https://techcrunch.com/wp-json/wp/v2/categories/${categoryId}`);
            return response?.name; // will return the category in string format.
        }));
    }
}
```

AI-Based Method

Training custom model:

Natural Language Processing (NLP):

Step 1: Use an NLP library (e.g., PyTorch, NLTK) to analyze the text of the news articles.

Step 2: Apply text classification algorithms (e.g., Naive Bayes, Support Vector Machines) to categorize the articles based on their content.

Step 3: Train the model with a dataset of news articles already categorized.

Step 4: Use the trained model to categorize new articles.

I have some prior experience developing machine learning models. So this can be done (if required).
My prior work related to machine learning and deep learning can be found here - **Sentiment Analysis**, **Image Classification**

Using Large Language Models (LLMs)
To demonstrate this, I have used OpenAI's API to create a custom prompt and generate the response through the endpoint, on the basis of that prompt.

```
export class TechCrunchNewsSource extends NewsSource {
    static news: NewsItem[] = [];
    fetchUrl = `https://techcrunch.com/wp-json/wp/v2/posts`;

    constructor(config) {
        super(config);
    }

    public async fetchNews(
        app: DemoNewsApp,
        filter?: {}
        ...other_params,
    ): Promise<void> {
        // logic to fetch news and store in NewsItem format
    }

    static async determineCategory(context: SlashCommandContext, read: IRead, http:
IHttp) {

        const room = context.getRoom();
        const appUser = (await read.getUserReader().getAppUser()) as IUser;


        const getOpenAIApiChatCompletion = () =>
`https://api.openai.com/v1/chat/completions`;
        const openAIApiKey = `${process.env.OPENAI_API_KEY}`;
        const getOpenAIPayload = (newsContent) => {
            const newsCategories = `General, Business and Finance, Technology,
Entertainment, Sports, Politics, Health, International, Investigative Journalism`;
            const prompt = `This news -> ${newsContent} falls under which category
out of these -> ${newsCategories}, answer in one word`
            const data = {
                model: "gpt-3.5-turbo",
                messages: [
                    {
                        role: "system",
                        content: prompt,
                    },
                ]
            }
            const headers = {
                'Content-Type': "application/json",
                "Authorization": `Bearer ${openAIApiKey}`,
            }

            return { headers, data };
        }

        let allNewsBlocks: Array<Array<Block>> = [];

        try {
            var message = `News fetched`;

            for (let i=0 ; i<this.news.slice(0, 3).length ; i++) {
                    const openAIResponse = await http.post(
                        getOpenAIApiChatCompletion(),
                        getOpenAIPayload(this.news[i].description),
                    );

                    const newsBlock = await buildNewsBlock(this.news[i]);
                    allNewsBlocks.push(newsBlock);
                }
                await sendMessage(modify, room, appUser, message, allNewsBlocks);
        } catch (err) {
            console.error('Error generating OpenAI response ', err);
        }
    }
}
```

The `determineCategory` method will call desired LLM and will classify if the category from source is not present.

Storing News

The news fetched from the source will be mapped from the source-specific format to a standard format and stored in the persistence storage.
The news model which will be stored is described below using a class based approach.

```
export class NewsItem {
    id: string;
    title: string;
    description: string;
    link: string;
    image: string;
    source: string;
    author?: string;
    publishedAt?: string;

    constructor(
        id: string, title: string, description: string, link: string, image: string,
source: string, author: string, publishedAt: string) {
        this.id = id;
        this.title = title;
        this.description = description;
        this.link = link;
        this.image = image;
        this.source = source;
        this.author = author;
        this.publishedAt = publishedAt;
    }
}
```

A specific job will be scheduled to save the news after fetching from all the sources, The *NewsFetchService* is described below.

```
export class NewsFetchService {
    newsSources = [];
    constructor() {
        // initialize the constructor
    }

    load() {
        // load all the news sources from settings
        // load config for each source
    }

    job() {
        // fetch news from all the sources, convert them to NewsItem format and store
in the persistence storage
        // delete old news from the storage
    }
}
```

If the category is not determined through the source while storing the news after fetching at scheduled intervals, AI based methods will be used which is described above.

Consuming News on demand

The consumption of news on demand would be done with slash command. A simple demonstration is shown below which fetches news stored in persistence storage and displaying on demand.

```
export class GetNewsCommand implements ISlashCommand {
    public command = 'get-news';
    public i18nParamsExample = "";
    public i18nDescription = "This is the command to get news on demand";
    public providesPreview = false;

    public async executor(context: SlashCommandContext, read: IRead, modify: IModify,
http: IHttp, persis: IPersistence): Promise<void> {
        const [category] = context.getArguments();
        const room = context.getRoom();
        const sender = context.getSender();

        // using just a single source for a demo.
        const newsSourceInstance = new TechCrunch(config, news);
        const filter = {categories: [category]};
        const newsItems = await newsSourceInstance.getNews(persis, filter);
        const newsDigestBuilder = await
modify.getCreator().startMessage().setRoom(room).setSender(sender).setParseUrls(true)
;

        const blocks = createDigestBlock(newsItems);
        // set the required blocks to display news
        messageBuilder.setBlocks(blocks);

        await modify.getCreator().finish(newsDigestBuilder);

    }
}
```

Subscribe News

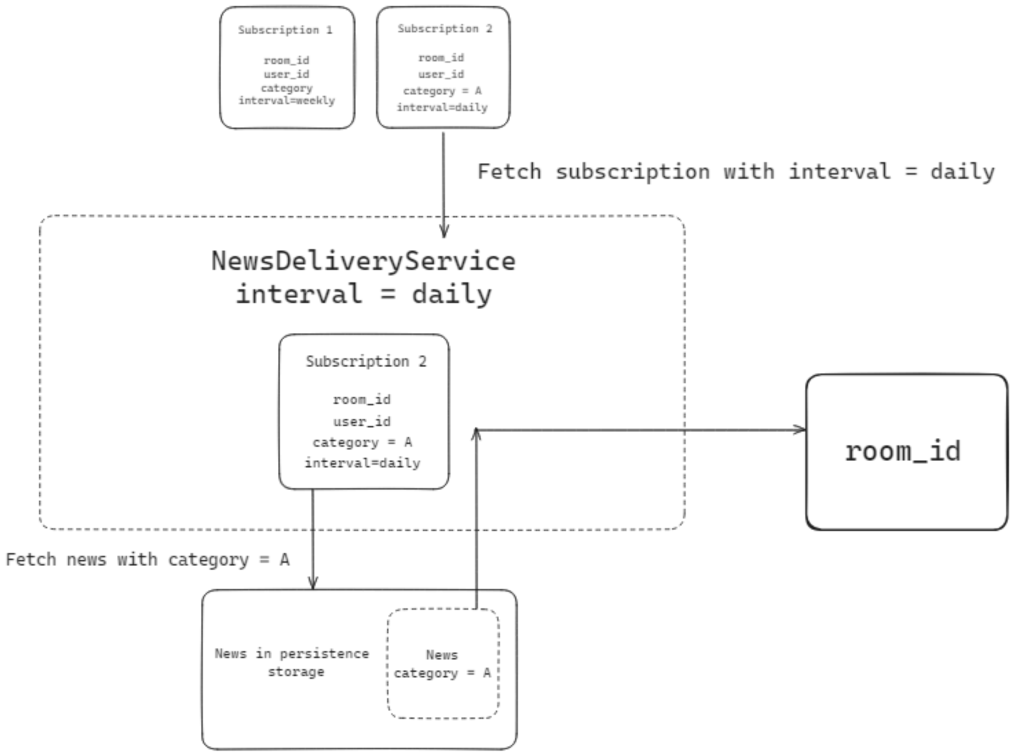
1. User will execute a slash command (***/news subscribe***) to create a subscription. In the slash command, a subscription object will be created and stored in persistence storage.

2. Then we will schedule jobs to process the subscription. For example, when a *daily* job will be scheduled, it will fetch all the subscription objects with *'daily'* interval value. Now it will fetch all the news that have the category same as in the subscription object and send it to the respective *room_id* of the subscription object.

When the user will subscribe through the command, then a *subscription* item would be created in this format:

```
export interface ISubscription {
    userId: string,
    interval: string,
    user: string,
    room: string,
}
```

Then a `NewsDeliveryService` would run to process subscriptions periodically.



Flowchart of how the news subscription would be done.

```
export class NewsDeliveryService {
    // this will be a scheduled job that will run at a specific time to deliver news
to users
    // send news
    context;
    constructor(context, ...other_params) {}
    public async scheduleDailyNews(modify: IModify) {
        const dailyTask = {
            id: 'dailyNewsTask',
            interval: '1 day',
            data: {
                context
            },
            skipImmediate: false,

        }

        await modify.getScheduler().scheduleRecurring(dailyTask);
    }

    public async scheduleWeeklyNews(modify: IModify) {
        const weeklyTask = {
            id: 'weeklyNewsTask',
            interval: '1 week',
            data: {
                context
            },
            skipImmediate: false,

        }

        await modify.getScheduler().scheduleRecurring(weeklyTask);
    }
}
```

This will fetch the information such as category, room_id, and then the messageBuilder will send the message to the respective rooms.
To delete a subscription, the subscription object will get deleted from the persistence storage.

Now the **`IProcessors`** would be registered in the **`extendConfiguration`** of the main entry app.

```
public async extendConfiguration(configuration: IConfigurationExtend,
environmentRead: IEnvironmentRead): Promise<void> {
        // other code
        // ...
        configuration.scheduler.registerProcessors([new DailyNewsProcessor()])
    }
```

The working of a sample **`DailyNewsProcessor`** is demonstrated below.

```
export class DailyNewsProcessor implements IProcessor {
    public id = "daily-news";
    async processor(jobContext: IJobContext, read: IRead, modify: IModify, http:
IHttp, persis: IPersistence): Promise<void> {
        // this would get the daily news in respective rooms.
        const context = jobContext;
        const {category, interval, news, ...other_params} = context;

        const roomPersistence = new RoomPersistence(
            persis,
            read.getPersistenceReader()
        );
        const subscribedRooms = await roomPersistence.getAllSubscribedRooms(
            interval,
            category
        );

        // setting blocks
        const blocks = getNewsBlock(news);
        const sendNews = async (subscribedRooms) => {
            try {
                const room = await
read.getRoomReader().getById(subscribedRooms.room.id);
                if (room) {
                    const newsBuilder = modify.getCreator()
                        .startMessage()
                        .setRoom(room)
                        .setBlocks(blocks);
                    await modify.getCreator().finish(newsBuilder);
                }
            } catch (err) {
                console.error(err);
            }
        };
        await Promise.all(subscribedRooms.map(sendNews))
    }
}
```

To schedule a daily news digest in Rocket.Chat using the Scheduler API, these steps would be followed:

Aggregate News: Gather news from various sources. This could be done by a separate service or part of the Rocket.Chat app.

Format the Message: Prepare the news digest into a readable message.

Schedule the Message: Use the Scheduler API to set the message to be sent as scheduled. The documentation to schedule the news will be referred from here - **Scheduler API**

To have scheduled options in the schedule service, configurations such as intervals would be provided when the instance of the job would be created. This option would be enabled only for admins.

The scheduled jobs would be canceled when a specific slash command would be executed, which would behind the scenes remove that particular subscription object from the storage.

To stop the scheduled news -

```
export class NewsStop implements ISlashCommand {
    public command = "stop-news";
    public i18nParamsExample = "";
    public i18nDescription = "";
    public providesPreview = false;

    constructor(private readonly app: DemoNewsApp) {}

    public async executor(
        context: SlashCommandContext,
        read: IRead,
        modify: IModify,
        http: IHttp,
        persis: IPersistence
    ): Promise<void> {
        const room = context.getRoom();
        const sender = context.getSender();
        const appUser = (await read.getUserReader().getAppUser()) as IUser;

        // logic to remove subscription
        try {
            const associations: Array<RocketChatAssociationRecord> = [
                new RocketChatAssociationRecord(
                    RocketChatAssociationModel.MISC,
                    `subscription`
                ),
                new RocketChatAssociationRecord(
                    RocketChatAssociationModel.USER,
                    `user:${appUser?.id}`
                ),
                new RocketChatAssociationRecord(
                    RocketChatAssociationModel.ROOM,
                    `room:${room?.id}`
                ),
            ];
            await persis.removeByAssociations(associations);
        } catch (err) {
            console.warn(err);
        }

    }
}
```
The scheduler API can be used like in - [Apps.Github]()
A demo of the scheduler API - [Demo]()

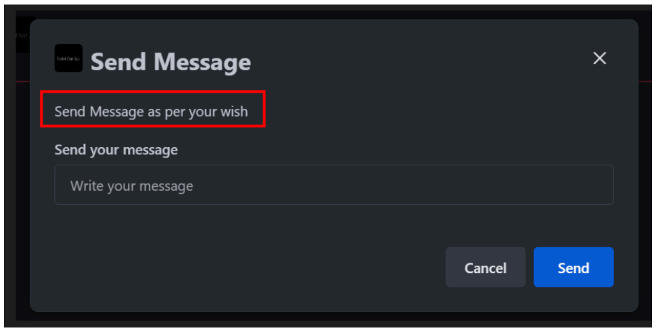
Displaying News

Displaying of the relevant news to the user on the channel will be done through [Rocket.Chat Ui-Kit]()

Some basic blocks built are shown below:
```
export function getSectionBlock(
    labelText: string,
    accessory?: any,
) {
    const block: SectionBlock = {
        type: "section",
        text: {
            type: "plain_text",
            text: labelText,
            emoji: true,
        },
        accessory: accessory,
    };
    return block;
}

export function getImageBlock(
    newsTitle: string,
    newsImage: string,
) {
    const block: ImageBlock = {
        type: "image",
        imageUrl: newsImage,
        altText: newsTitle,
    }
    return block;
}
```
This code will render a section block and an image block respectively.

```
export function getMultiStaticSelectElement(
    placeholderText: string,
    options: Array<Option>,
    appId: string,
    blockId: string,
    actionId: string,
    initialValue?: Option["value"][],
) {
    const block: MultiStaticSelectElement = {
        type: "multi_static_select",
        placeholder: {
            type: "plain_text",
            text: placeholderText,
        },
        options,
        appId,
        blockId,
        actionId,
        initialValue,
    };
    return block;
}
```
This code will render a MultiStaticSelectElement.

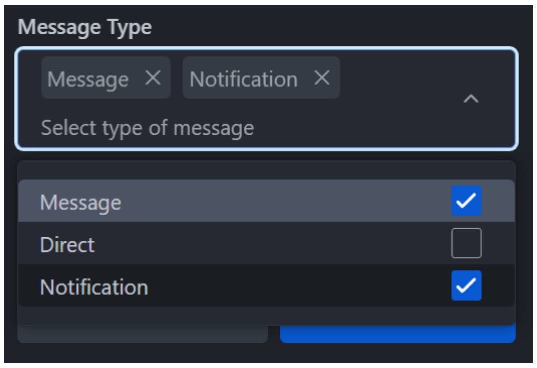
```
export function getButton(
    labelText: string,
    blockId: string,
    actionId: string,
    appId: string,
    value?: string,
    style?: ButtonStyle.PRIMARY | ButtonStyle.DANGER,
    url?: string,
) {
    const button: ButtonElement = {
        type: "button",
        text: {
            type: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


Send Message

Send Message as per your wish

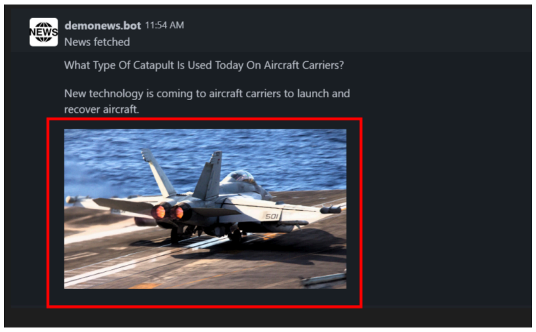
Send your message

Write your message


Message Type

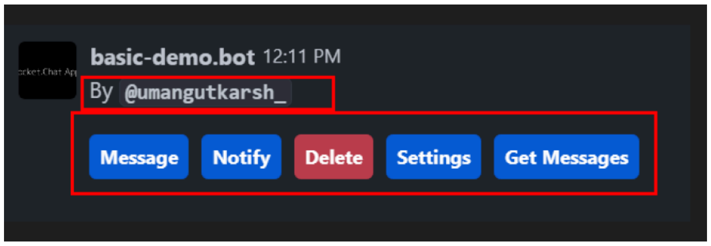
Section Block

MultiStaticSelectElement Block


demonews.bot 11:54 AM
News fetched

What Type Of Catapult Is Used Today On Aircraft Carriers?

New technology is coming to aircraft carriers to launch and recover aircraft.


basic-demo.bot 12:11 PM

By @umangutkarsh_

Image Block

Buttons and Markdown block

Rocket.Chat Demo App

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

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

Sending message, Notifying user, direct message to the user

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

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

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

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

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

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

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

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

App Data Persistence

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

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

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

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

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

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

        return result;
    }
```
Get all data^

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

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

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

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

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

Remove a data^

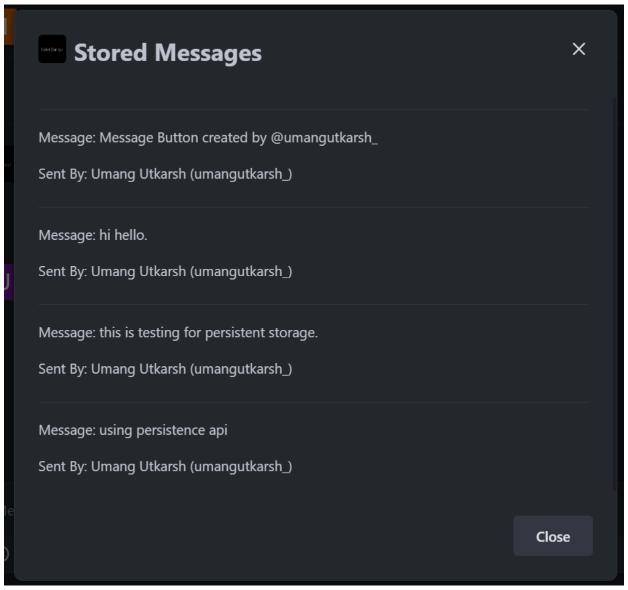
All of these functionality could be accessed through a class name `MessagePersistence`

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

OR

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

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


Stored Messages

Message: Message Button created by @umangutkarsh_

Sent By: Umang Utkarsh (umangutkarsh_)

Message: hi hello.

Sent By: Umang Utkarsh (umangutkarsh_)

Message: this is testing for persistent storage.

Sent By: Umang Utkarsh (umangutkarsh_)

Message: using persistence api

Sent By: Umang Utkarsh (umangutkarsh_)

Building UI Blocks

This is demonstrated above in the Ui-Kit section

Workflow Timeline

Application Review Period (April 2 - May 1)

During this period, my focus will be on:

1. Gain insights into Apps-Engine and RC developer documentation.
2. Investigate Apps-Typescript definition and TypeScript for app development.
3. Explore reliable news sources and categorization mechanisms.
4. Engage with the Rocket.Chat community, contributing to discussions and projects.

Community Bonding Period (May 1 - May 26)

During this period, my focus will be on:

1. Establish communication with my mentor for strategy discussions.
2. Deepen engagement with the Rocket.Chat community.
3. Conduct in-depth research on the Rocket.Chat core codebase.
4. Investigate categorization mechanisms, including LLMs and ML model testing.
5. Engage in discussions to refine app features and user experience.

Week 1 (May 27 - June 2)

- Set up the basic structure of the News Aggregation App.
- Make sources configurable in settings for sources such as APIs and RSS.
- Fetch news from various sources and make them configurable.

Week 2 (June 3 - June 9)

- Use Object Oriented Programming's abstraction for app structure.
- Use persistence API for data management.
- Develop basic user interface with Ui-Kit.

Week 3 (June 10 - June 16)

- Train and test a custom model for categorization *(if required)*.
- Explore open-source LLMs for categorization.
- Work on custom model deployment and API access.

Week 4 (June 17 - June 23)

- Research efficient deployment methods.
- Improve user interface and user experience.

- Build complex blocks for app interaction.

Week 5 (June 24 - June 30 - Midterm Evaluations)

- Buffer week for mid-term evaluations.
- Work on feedback and incomplete tasks (if any).

Week 6 (July 1 - July 7)

- Work on job scheduling using Rocket.Chat scheduler API.
- Implement basic news scheduling.
- Enhance news scheduling efficiency.

Week 7 (July 8 - July 14)

- Develop 'on-demand' news fetch feature.
- Further improve user interface and user experience.
- Work on documentation and summarize work done.

Week 8 (July 15 - July 21)

- Develop extra features if time permits.

Week 9 (July 22 - July 28)

- Buffer week for feedback and final adjustments.
- Complete any remaining work.

Final product and Evaluation submission period would be for the final week.

The actual duration can be adapted depending on the actual availability as per the duration for a small project, i.e. 8 - 12 weeks.

Future Development

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

1. **Provide a news search option:** Providing a keyword search option to filter out desired news.
2. **Analytics and Insights:** Provide admins with analytics and insights on user engagement, popular categories, and sources, helping to refine the app's features and content.

3. **Content Moderation:** Incorporate a basic content moderation system to filter out inappropriate content, ensuring a safe and respectful environment for users.
4. **Social Sharing:** Allow users to share news articles on social media platforms, promoting the app and its content.

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, configuring and managing 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

These projects are relevant 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, which uses a completely different approach which is writing typescript in a constrained format.

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, small projects require around 90 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.