

# GSoC 2023 Proposal



## Share data between Talawa and other application suites

Ravi Dev Pandey

**GitHub:** [literalEval](#)

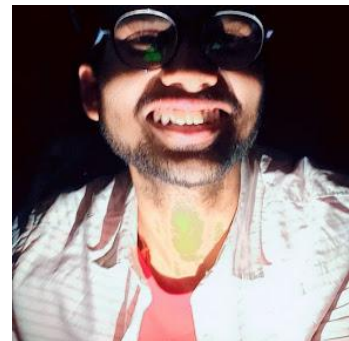
**LinkedIn:** <https://www.linkedin.com/in/ravidev-pandey/>

**Email:** [pandeyravidev@gmail.com](mailto:pandeyravidev@gmail.com)

**Phone:** (+91) 84000 42939

**College:** Indian Institute of Technology (BHU), Varanasi

**Postal Address:** Vill - Harkha Pyas, Pipra Brahman (Barigaon), Maharajganj, Uttar Pradesh, India.  
274301



## Contents

|  |           |
|--|-----------|
| <b>Contents</b>  | <b>1</b>  |
| <b>Overview</b>  | <b>3</b>  |
| <b>Proposed Deliverables</b>                           | <b>4</b>  |
| <b>What exists?</b>                                    | <b>5</b>  |
| <b>Case Study: WhatsApp</b>                            | <b>5</b>  |
| <b>Plan of Action</b>                                  | <b>6</b>  |
| Posts, Public Posts, Content Sharing, and In-App Sites | 6         |
| Metadata   | 7         |
| In-App Video Player                                    | 8         |
| <b>Implementation</b>                                  | <b>9</b>  |
| Public Posts   | 9         |
| Schematic Diagram of the flow                          | 11        |
| Dynamic Linking and Custom URL support                 | 12        |
| Metadata   | 13        |
| TalawaMetadataService                                  | 13        |
| TalawaURLExtractor                                     | 14        |
| TalawaMetadataWidget                                   | 14        |
| Overall Integration                                    | 15        |
| In-App Video Player                                    | 16        |
| TalawaPIPView  | 16        |
| Support for General Video Sites                        | 18        |
| <b>Milestones</b>                                      | <b>18</b> |
| I. Community Bonding Period (May 4 - 28)               | 20        |
| II. Week 1 and 2 (May 29 - June 11)                    | 20        |
| III. Week 3 and 4 (June 12 - June 25)                  | 20        |
| IV. Week 5 and 6 (June 26 - July 9)                    | 21        |
| V. Mid-Term Evaluation (July 10 - July 14)             | 22        |
| VI. Week 7 and 8 (July 17 - July 30)                   | 22        |
| VII. Week 9 and 10 (July 31 - August 14)               | 22        |
| VIII. Week 11 and 12 (August 15 - August 27)           | 22        |
| IX. Final Evaluation (August 28 - September 4)         | 23        |
| X. Week 13 and 14 (September 4 - September 17)         | 23        |
| XI. Week 15 and 16 (September 18 - October 1)          | 23        |



|  |           |
|--|-----------|
| XII. Week 17 and 18 (October 2 - October 15)   | 24        |
| XIII. Week 19 and 20 (October 16 - October 29) | 24        |
| XIV. Week 21 (October 30 - November 5)         | 25        |
| <b>Why Me?</b>                                 | <b>25</b> |
| <b>My Contributions</b>                        | <b>26</b> |
| The Palisadoes Foundation                      | 26        |
| CCExtractor                                    | 27        |
| AppFlowy                                       | 27        |
| Faustus  | 27        |
| <b>Any Other Organization?</b>                 | <b>27</b> |
| <b>About Me</b>                                | <b>27</b> |
| <b>Communication</b>                           | <b>28</b> |
| <b>Post GSoC</b>                               | <b>29</b> |
| <b>THANK YOU</b>                               | <b>29</b> |

## Overview

The project idea asks to implement some ways through which users can share and receive content to/from other apps and view information about external links in the app itself, just like popular messaging apps like WhatsApp and Telegram. This includes

1. **Allowing users to share posts and other content from Talawa to other apps and vice-versa.**
2. The **shared posts** should be **deep linking enabled**.
3. Allow Admins to **make a post public** and **share a public URL** that everyone can open.
4. Open **external links** in Talawa itself with **WebView**.
5. Showing **metadata** about the **link in-app**. Title of the webpage, image (if any), and other metadata of interest.
6. Allowing users to **view externally hosted videos in-app**.
7. This whole system should be **controllable from the admin panel itself**.






## Proposed Deliverables

- Users will be able to **share content and chats** from Talawa to other apps and vice-versa.
- Users will be able to **share posts** and they will be **dynamic-linking enabled**. Moreover, the **admin** will have an option for **making a post public**, in which case anyone can view the post irrespective of whether they have joined the community or not.
- Any **external** link will be **opened in a WebView** in the Talawa app itself, instead of an external web browser.
- Users will be able to see **general information** about the first **URL in any post, chat, message, etc.** This can be further discussed to support all the links present in a text, not only the first one.
- **Allow users to open and view YouTube videos in the Talawa app itself** either in full screen or in a floating pip (picture in picture) window. This can be further discussed to extend other video-playing sites like **Dailymotion, Reddit**, etc. **Admin** will be able to **toggle this feature**.
- **Complete coverage and documentation** for everything added and changed during the implementation.
- Testing the UI on both Android and iOS to ensure it looks the same everywhere.

### Other Ideas that I will implement are -


- Optionally, the **admins** can choose to make the **public posts browser openable**.
- **Internal links** (that of Talawa) will **directly open** the content **in-app**, without opening the URL in the browser.
- Optionally give admins and users the option to allow for caching, in which case the app or server database will cache the metadata fetched so that it doesn't need to fetch it every time the app loads/reloads, based on what admin chooses.
- **Export Chat, Post, etc as a text file.**
- List all of the **media in a single place** to make **finding and sharing content easier**.



## What exists?

The Chat, Posts, Events, etc systems, where we can integrate this feature, already exist and though they have some bugs for now, their skeleton works fine. Their corresponding APIs are in Talawa API and perfectly integrated with the app. UI for these services is also present and is well-tested. However, we will need new widgets that allow showing metadata and PIP views, along with some new screens for new routes.

So the existing API and UI can be enhanced to our needs, and some custom widgets can be created.



## Case Study: WhatsApp

Before building upon our idea, we will look at this feature's implementation in some industry-standard apps.

WhatsApp shows a lot of information about the links in-app, which includes:


1. Showing metadata of the **first URL** found in the message. Also, it fetches the metadata in **real time**, which means we can see the metadata changing as we type a URL in the chat input field.  
The metadata includes -
  1. The corresponding image associated with the URL.
  2. Title of the webpage.
  3. A text showing the main domain of the URL
2. The layout of the above fields also varies:
  1. For YouTube and Instagram links, the image is shown in large size and the overall layout followed is a column.
  2. For other links, It shows metadata in a relatively compact format using a row.
3. YouTube links are special ones. WhatsApp shows a **play icon** over the image fetched from the metadata and clicking over it opens a **popup window** that plays the **YouTube video in-app**.



The in-app YouTube video player has two modes:

1. **Full-Screen Mode:** The video player occupies all of the screen just like YouTube does when we go to full-screen mode. Works like a full-fledged video player.
2. **Floating Window Mode:** Where we see a pop-up floating over our normal chat screen. The user can move the pop-up either to the top or bottom of the screen by dragging it. It shows all the control buttons as well as a **toggle full-screen** button which activates the **Full-Screen Mode**.

The user gets a cross button over the fetched metadata while typing so that they can choose to hide that if they like. In this case, the metadata is not shown in either party's app.




## Plan of Action

Since there are project ideas for **`Improved Member Management`** and **`General Functionality`**, I will work closely with the people who will implement these features, because that is where this project needs to be integrated.

## Posts, Public Posts, Content Sharing, and In-App Sites

1. We can use some plugins for the sharing part as it will require us to write some platform-specific code. Alternatively, if the plugin is quite old and not maintained actively, we can study the implementation of the said plugins and implement them on our own. The custom implementation has the advantage of being heavily testable, as well as can be kept updated always. One such plugin is [receive\\_sharing\\_intent](#). This allows receiving content sent from other apps to our own. Since this plugin is not actively maintained, I will prefer to implement it on my own. Though my decision will give a lot of priority to the mentor(s)' advice.
2. We can use [share\\_plus](#) to facilitate sharing of **content from Talawa to other apps**. This plugin is actively maintained and can be used directly.
3. Deep Linking is already implemented but has some issues. The plugin we are using to handle incoming links is no longer maintained, and we need to use an official site



too. We can migrate to **Firestore Dynamic Linking** to handle our deep links, as it is always free for whatever traffic and scale. Moreover, if we would like to have a custom domain, we can use any of our sites to achieve this. I can work with both.

4. For the public post thing, I think of using the top layer of our API server (on which GraphQL runs). The URL for a public post will be a direct link to the API so that even a normal GET request fetches the post. Though I will like to make this optional so that the community owner can choose to not have public posts at all if they don't want to.
5. I will use the [webview\\_flutter](#) plugin to facilitate the opening of external links in Talawa itself.
6. I will create a new screen in the Chat system that will show all of the media in a single place (just like WhatsApp, Telegram, etc. do). This will make finding and sharing content easier.

## Metadata

1. I will start by discussing with the mentees selected for these projects and understand how they will be implementing these features. Once having understood their plan of action, I will start by discussing the APIs for Chats, Posts, and all the other places where we need this feature, to make them support the optional metadata field. If any of these projects are not selected for the GSoC period, then I will do the necessary discussion for their implementation to make things work, so that we can build our idea on top of that.  
This metadata field will be optional and configurable via Admin Panel.
2. Next, I will implement `TalawaMetadataService`, which will be responsible for fetching and parsing the metadata of a given URL. It will fetch the HTML associated with the URL and parse it to extract valuable information like the **title of the page**, an **image associated with it** (if any), etc.
3. After this, I will create `TalawaMetadataWidget` which will work as a parent container to a child widget. It will show the child widget as it is if no **metadata** is provided and will show a **metadata widget** around the child (this alignment can be configured) if metadata is not null and is in a valid format. It will also be configurable to not show



metadata information at all if the user chooses to (by having a cross button maybe).

Once the UI part is done and is workable, I will move on to implement text parsing. A function (or if required, then a service) will be implemented to extract the first URL in a text (and can be configured to extract all of them, not only the first one). Then I will attach **TalawaMetadataService** to the text field and the response of **TalawaMetadataService** to **TalawaMetadataWidget** so that metadata is fetched in real-time and shown to the user, just like WhatsApp. This integration of these services and widgets will be **heavily tested**.


4. When all of this becomes stable, I will move on to wrap widgets of interest, like **ChatMessageBubble**, and **Post**, with **TalawaMetadataWidget**, and adjust UI scaling, etc according to the new UI. This will also be an excellent place to discuss the size of the metadata shown so that it doesn't clutter up the whole area.
5. Once all of this starts working in the app, I will modify Talawa API to support an additional metadata field. I will make the necessary changes to store the metadata in the database along with other fields of the post, chat, etc. Additionally, I will discuss with the mentors and use **ChachedNetworkImage** in place of normal **NetworkImage** to save some bandwidth and time in fetching the metadata image from the URL every time.

## In-App Video Player

There is a limitation to what we can extract from the URL's corresponding HTML page. Even if a link refers to a video, we can't always simply get the direct link to the related video by parsing the webpage. And this seems to be the case with messaging apps like WhatsApp, Telegram, etc. If a GDrive link corresponds to a video, these apps can show a thumbnail of the video (which they get from metadata), but can't play the video in-app. But, nearly all of the major video streaming sites have their corresponding APIs, which respond with the video we need, given the video id.

6. Talawa already uses the **video\_player** plugin, which supports network videos. So I will start by integrating the API of major video hosting sites like [YouTube](#), [DailyMotion](#), etc into **TalawaMetadataService**. Given a video id (which we will extract from the URL provided), this API will return the actual URL of the video, which we can use with the **video\_player** plugin.

Alternatively, some plugins are available for YouTube videos like



[youtube\\_player\\_flutter](#), which support **captions, live streams, changing playback speed**, etc. out of the box. I will use this plugin for YouTube videos and metadata fetching + `video_player` for other video sites.

7. Once I am satisfied with the work of **TalawaMetadataService**, I will proceed to implement it in the **app UI**. For this, I will create a widget called `TalawaPIPView`. **TalawaPIPView** will work by scaffolding two widgets in itself, namely the **backgroundPage** and the **floatingWindow**. It will use a **Stack** to emulate the floating behavior, in which the **backgroundPage** will always be shown and the **floatingWindow** will come into existence and float over the background if needed.
8. After that, I will wrap all the screens that will use in-app features with **TalawaPIPView**, with our custom video player as the **foregroundWidget**. This way we can play the videos in-app, move the floating window here and there, minimize, maximize, and close it.



## Implementation

Implementing this project will require some changes and additions to the existing codebase in **Talawa**, **Talawa API**, and **Talawa Admin**.

### Public Posts

When asking the API server for a post, we can **skip the auth for once** and check if the said post is **public or not**. If it is, we return the post for any user, otherwise, check for auth and return the post only for an authenticated user. Moreover, we can **extend this idea** and **allow public posts to be seen in a browser** itself. This will be optional and will **depend on the Admin**.

For this, I plan to use the **NodeJS layer of the API** itself, as the browsers will only understand **REST requests** and not GraphQL ones. Then, a public post's link can be generated as two types. First, as a normal deep link enabled Talawa link, or second, as a normal browser openable URL. Since no site exists for the users to use Talawa in a browser, we will need to build this thing from scratch. Maybe in the future, when we will be having a website along with the app, the feature could be further enhanced.

I will add an option of making a post public in Talawa Admin, so that an Admin can choose to make and share the URL of a public post.

The **pseudocode** can be as follows -

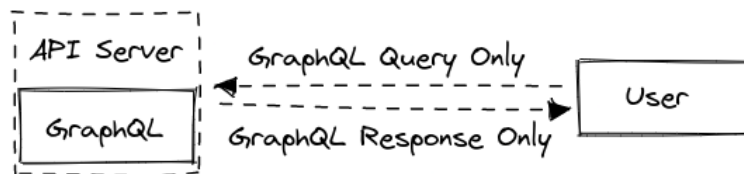
```
// Expose the route /post/:id onto the main backend API
app.get("/post/:id", (req, res) => {
  const postId = req.params.id;

  // Here we are trying to get the post from the Post model from
  // the id provided in the parameter. We have included the
  // public: true field in the arguments itself as if now the post
  // is private, the same would not match the filter query provided
  // and no object will be returned

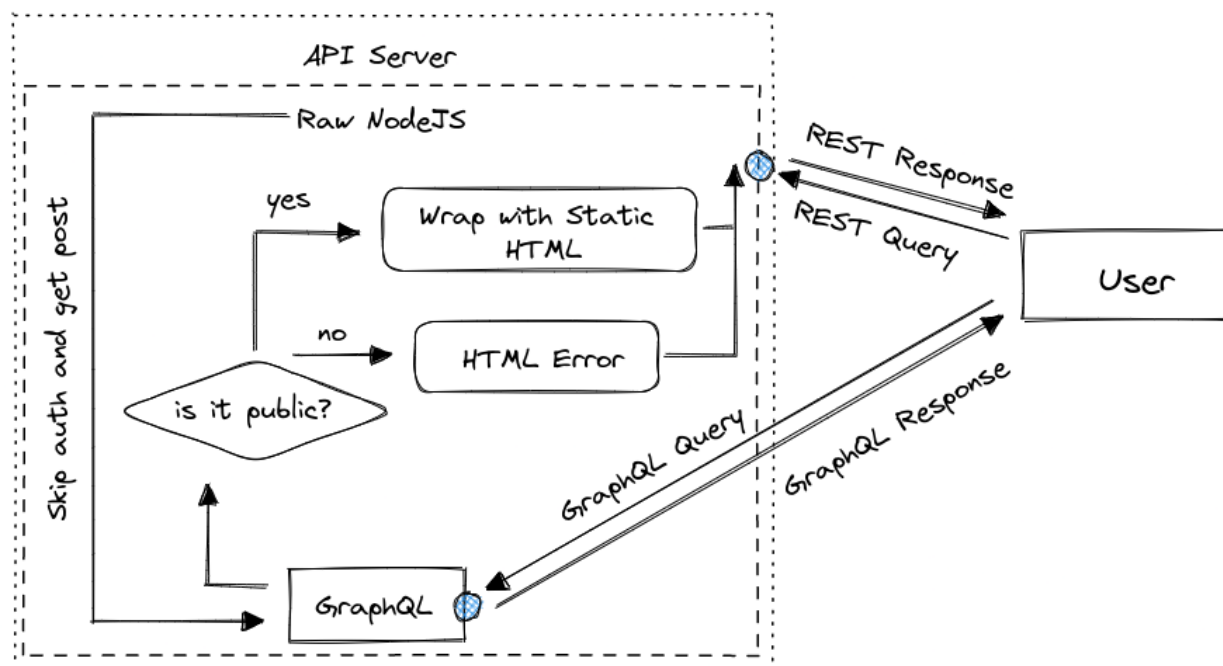
  const postObject = Post.findOne({
    _id: postId,
    public: true,
  }).lean();

  // Render a static HTML page by injecting the postObject details
  // into the same
  res.render("/path/to/the/view", postObject);
});
```

## Schematic Diagram of the flow



Old Architecture



New Proposed Architecture

## Dynamic Linking and Custom URL support

Implementing **Firestore Dynamic Linking** is pretty standard stuff and can be done easily. Setting up a custom URL for this will take some configuration (adding **assetlinks.json** etc to the site). Though the implementation already exists and can be referred to directly while implementing for the new domain. There are a lot of online guides available for this ([example](#)) and all of this is pretty general and doesn't need any elaboration.

Though the public URL of a Post will break the Dynamic Link as the app won't be configured to recognize each of the communities' servers. For this, we can:

1. Give a banner on the webpage to open Talawa.
2. Add Deep Link boilerplate to our Node Server to make the custom sites deep link enabled. The communities will need to modify the boilerplate along with other setups to make it work.



Standard Talawa deep link  
enabled URL

Custom Browser openable URL

**Moreover**, I will modify the mechanism of opening external links and **have a check** that looks at whether the **URL belongs to Talawa or is an external URL**. This can be done using **RegEx pattern** matching, and once we get a Talawa URL, we can **route the user directly to the respective page** instead of opening the URL in the browser separately.

## Metadata

We will fetch the metadata on the client (app) side itself, and this fetching will have no connection with the server whatsoever. The reason for this is that when we implement End-to-End Encryption in our project, there will be no way for the server to know the contents of a message. Sure we could save some users' bandwidth by moving this part on the server, but compromising the security is not possible and so client-side fetching is the only option.

### TalawaMetadataService

I will use **HTTP get request** to fetch the HTML page of the URL and parse it with HTML Parser to get fields of interest.

```
final response = await http.get(
  Uri.parse('https://github.com/literalEval'),
);

final document = parse(response.body);
final metaElements = document.getElementsByTagName('meta');

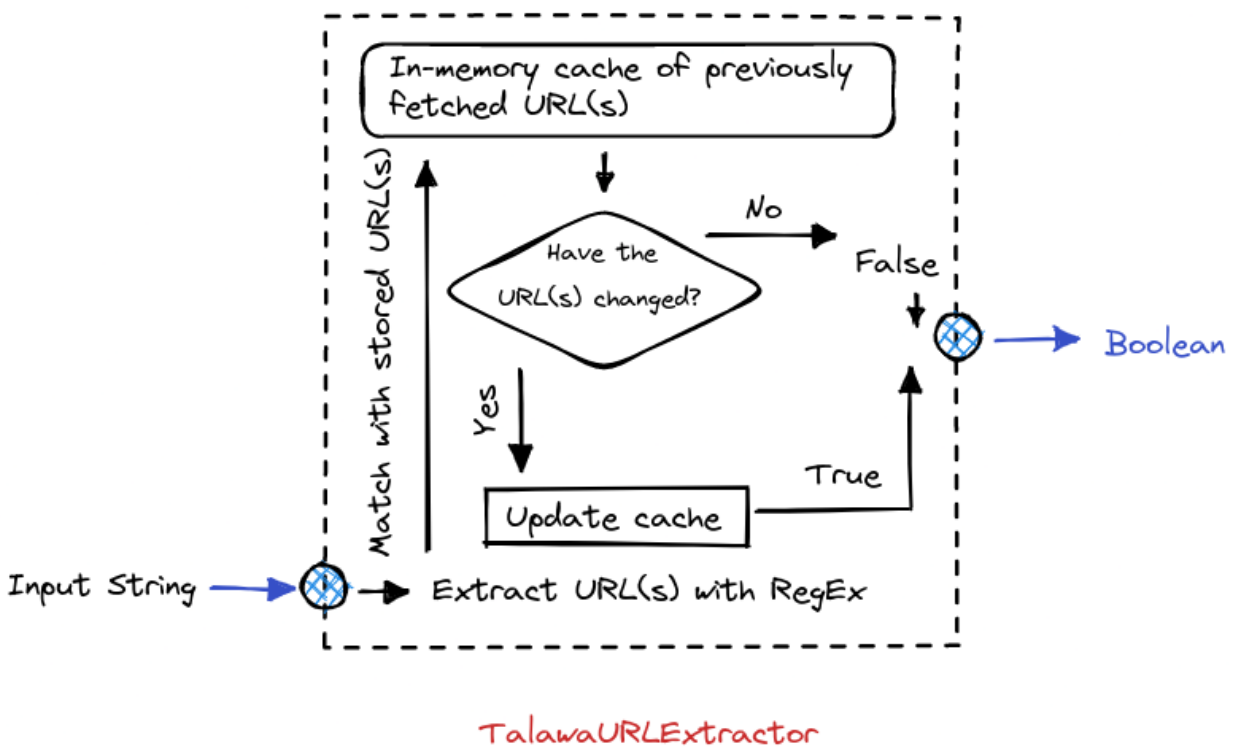
String? title, image, description;

for (final element in metaElements) {
  if (element.attributes['property'] == 'og:title') {
    title = element.attributes['content'];
  }
  if (element.attributes['property'] == 'og:image') {
    image = element.attributes['content'];
  }
  if (element.attributes['property'] == 'og:description') {
    description = element.attributes['content'];
  }
}
```

## TalawaURLExtractor

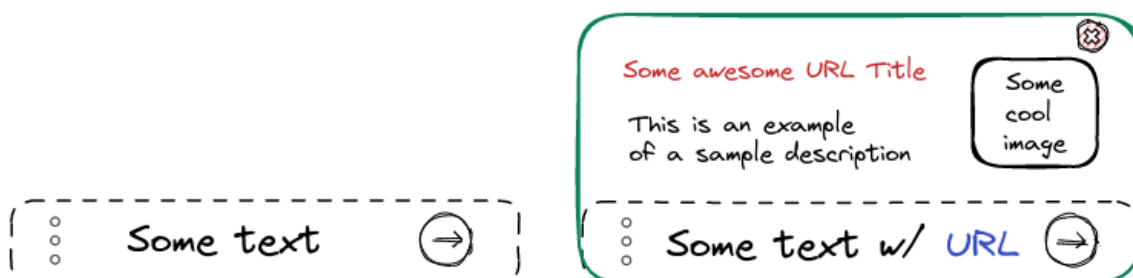
This will be implemented using **RegEx**. Given a String of text, it will return all of the URLs it can find in the text. I plan to make this a **'stateful' class** so that any instance **'remembers'** the URLs and texts it **has parsed recently**. This will be used to ensure **some cooldown and caching** to not waste **users' bandwidth** on unnecessary metadata fetching. Moreover, this can be a general class that also **extracts any ASCII emoji**.

1. Parse the text and extract the URLs. Optimally extract and swap ASCII emojis
2. Send a 'no-change' signal if none of the URLs have changed, or the URL is actively being edited.
3. Send a 'change' signal only if a URL has changed and its editing is finished.



## TalawaMetadataWidget

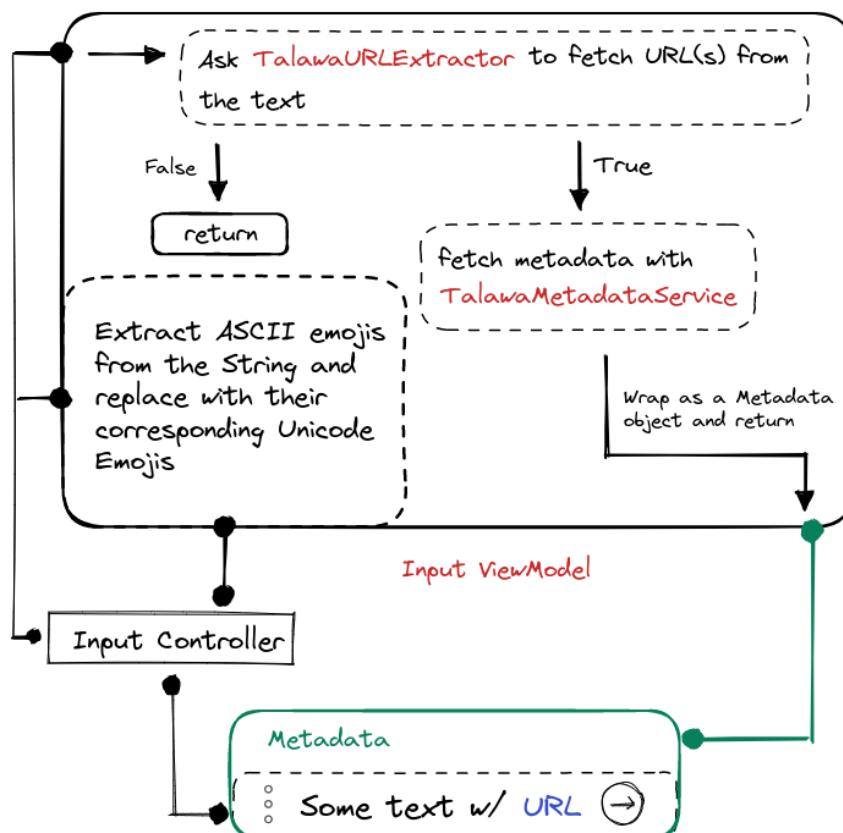
A **Column** can be used with **AnimatedBuilder** and a widget as its children. The **AnimatedBuilder** will be shown when it gets some metadata and will not be shown otherwise, whereas the child widget will be shown always. Having the **AnimatedBuilder** will help us achieve some beautiful animations.



*TalawaMetadataWidget*

## Overall Integration

**TalawaURLExtractor**, **TalawaMetadataService**, and **TalawaMetadataWidget** combined should work like this





## In-App Video Player

This part will be inclined more toward UI changes and designs.

### TalawaPIPView

A stack can be used to simulate a floating effect. The floating window can be wrapped with **GestureDetector** and **AnimatedBuilder**. The GestureDetector will allow for detecting the taps and swipes (which the user will do to move the floating window), whereas the AnimatedBuilder will ensure that all the movements of the floating window are smooth and animated. The floating window can be made sticky by wrapping this whole system inside LayoutBuilder. The LayoutBuilder will provide the exact size our widget is getting and we can use that to calculate the position of the edges of the usable screen. This information can be used to calculate the nearest corner, given any position.

```
return LayoutBuilder(
  // Dimension calculations
  // ...
  // ...
  builder: (ctx, child) {
    return Stack(
      children: [
        widget.backgroundPage,
        if (_showFloat)
          AnimatedBuilder(
            animation: _dragAnimationController,
            child: widget.floatingWidget,
            builder: (context, child) {
              return Positioned(
                left: _floatOffset.dx,
                top: _floatOffset.dy,
                child: GestureDetector(
                  onPanStart: _showFloat ? _onPanStart : null,
                  onPanUpdate: _showFloat ? _onPanUpdate : null,
                  onPanCancel: _showFloat ? _onPanCancel : null,
                  onPanEnd: _showFloat ? _onPanEnd : null,
                  onTap: widget.onTapFloat,
                  child: child,
                ),
              );
            },
          ),
      ],
    );
  },
);
```

```
    },
  );

```

The **Pointer Events** of the GestureDetector will be used to listen for movements and scaling of floatingWidget. **onPanStart**, **onPanUpdate**, **onPanCancel**, and **onPanEnd** will be used to mark start, update, cancel, and end events respectively. **\_floatOffset** will store the current position of the floatingWindow on the screen and will be changed on Pointer Events.

Eg: **\_onPanUpdate** can be implemented like so:

```
void _onPanUpdate(DragUpdateDetails details) {
  if (!_isDragging) return;
  setState(() {
    _dragOffset = _dragOffset.translate(
      details.delta.dx,
      details.delta.dy,
    );
  });
}

```

Then, when the Pointer Event ends, we can move the floating window to the nearest corner.

```
void _onPanCancel() {
  if (!_isDragging) return;
  setState(() {
    _dragAnimationController.value = 0;
    _dragOffset = Offset.zero;
    _isDragging = false;
  });
}

void _onPanEnd(DragEndDetails details) {
  if (!_isDragging) return;

  final nearestCorner = _calculateNearestCorner(
    offset: _dragOffset,
    offsets: _offsets,
  );
  setState(() {
    _corner = nearestCorner;
    _isDragging = false;
  });
  _dragAnimationController.forward().whenCompleteOrCancel(() {

```

```

    _dragAnimationController.value = 0;
    _dragOffset = Offset.zero;
  });
}

```

There also exist some plugins like [pip\\_view](#) (whose implementation I studied thoroughly before devising my implementation), which allow us to create PIP View widgets. But I think we should create our own custom PIP View, as discussed above, as it is only a 1-2 file implementation. Also, creating our own will allow us to further customize it to our will (say in the future we decide to allow free movement of the floating window instead of sticking it to a corner), and also test widgets that depend on it, properly.

## Support for General Video Sites

This will require some research and discussion to decide which sites we would like to support and to what extent. The aim will be to get the direct URL of the video associated with a general URL. This includes checking the meta tags for a video tag or pinging their API with the video id that we get.

## Milestones

Strictly dividing the work on a per week basis is hard for me so I am dividing the work on a fortnightly basis. Since we take documentation and testing very seriously in the Palisadoes Foundation, every fortnight of work will be followed by complete documentation and testing of the work done till then.

| Deliverable              | Work Done   | Timeline                          |
|--------------------------|---|-----------------------------------|
| Community Bonding Period | Bonding with the mentees and the org, in general.                                 | <a href="#">May 4 - May 28</a>    |
| In-app Metadata          | <b>Metadata extractor</b> with caching and cooldown                               | <a href="#">May 29 - June 11</a>  |
| In-app Metadata          | Create a <b>metadata showing widget</b> that supports network images with caching | <a href="#">June 12 - June 25</a> |

| Deliverable                                  | Work Done   | Timeline                          |
|--|---|-----------------------------------|
| In-app Metadata                              | Create <b>URL and ASCII emoji extractor</b> , and integrate all of the metadata services to all the places of need  | <a href="#">June 26 - July 9</a>  |
| Evaluation                                   | Everything related to <b>metadata</b> will <b>work</b>  | <a href="#">July 10 - July 14</a> |
| Deep Linking                                 | Implement <b>Firebase Dynamic Linking</b> (or any other method as the mentor(s) suggest(s). Add logic for the creation and opening of posts links   | <a href="#">July 17 - July 30</a> |
| Public Posts                                 | Add logic to make a <b>post public</b> . Give options in the admin panel itself to make a post public and share a public URL. Modify Talawa API to support browser openable public posts. | <a href="#">July 31 - Aug 14</a>  |
| Public Posts                                 | Make <b>browser openable custom community URLs dynamic link ready</b> . Integrate a custom deep link scheme.  | <a href="#">Aug 15 - Aug 27</a>   |
| Evaluation                                   | The <b>metadata</b> feature and the <b>public posts</b> features will be <b>completely ready</b> and <b>usable</b> .  |                                   |
| Sharing content, chat, post, etc from Talawa | Install plugins that facilitate post, chat, and media sharing and implement logic to create public URLs of public posts.  | <a href="#">Sept 4 - Sep 17</a>   |
| Receiving content from other apps            | Install plugins that facilitate receiving of content from other apps and create routes that will open respective pages.   | <a href="#">Sept 18 - Oct 1</a>   |
| In-app PIP View                              | Create PIP widget and install plugins to facilitate special access to most used video platforms   | <a href="#">Oct 2 - Oct 15</a>    |
| In-app PIP View                              | Finalize PIP integration and add it to all the places that need the feature. Perform a complete E2E testing of the feature.   | <a href="#">Oct 16 - Oct 29</a>   |
| Suggestions                                  | Discuss and implement anything else, as suggested by the mentor(s)  | <a href="#">Oct 30 - Nov 5</a>    |
| Final Evaluation                             | The project will be ready to be evaluated by the mentor(s) and used by an end user.   |                                   |



## I. Community Bonding Period (May 4 - 28)

- I will start by discussing the implementation of **Improved Member Management**, and **General Features** with the selected mentees of these projects. If these projects are not selected for this year, I will discuss which parts need to be changed, with the mentors.
- Will discuss the modification in existing APIs needed to make the metadata system work and discuss with the mentors of **Talawa API** the best ways to do so.
- I will discuss the features in the Ideas List, and after undertaking some suggestions, consider and crosscheck my Plan of Action.
- Discuss the **caching aspect** of this project and discuss in detail whether we need a caching system or not.
- I will also be discussing **Talawa's release on Play Store** after final implementations and checks. This would be optional and will heavily depend on the other work done by the mentees and the approval of mentors and admins.

## II. Week 1 and 2 (May 29 - June 11)

- Implement **TalawaMetadataService** to fetch the HTML page of a URL and parse it to extract its metadata. Create a **Metadata** class that will store **parent domain, image, title**, etc metadata fetched from a URL. Study the different meta tags exposed by general sites and some special APIs like the YouTube data API to get video thumbnails from a given video ID. Discuss the then scenario of which sites we will be focusing on, and study their API and tags specifically.
- Implement in-memory caching and cooldown in **TalawaMetadataService** so that it doesn't fire up on every keystroke and not at all when the same URL is provided. Additionally, discuss and implement local storage caching in the app using **Hive** to store frequently used links and domains by the user. The aim will be to use as little bandwidth in subsequent metadata fetches as possible.

## III. Week 3 and 4 (June 12 - June 25)

- Discuss the progress of API integration for Posts, Chats, etc with the assigned mentees and mentors and do necessary modifications and additions in Talawa API for it to support an additional and optional **metadata** field wherever necessary.
- Implement a new widget called **TalawaMetadataWidget**, which will act as a wrapper around another widget and show the provided metadata. Discuss and extend **TalawaMetadataWidget** to support multiple metadata at once, if needed. Discuss the layout and design of the metadata image, parent domain, title, and content, and how it should vary according to the dimensions of the metadata image fetched. Discuss the usage of **CachedNetworkImage** for frequently used sites instead of normal **NetworkImage** to save some bandwidth and implement the metadata image accordingly.

#### IV. Week 5 and 6 (June 26 - July 9)

- Create **TalawaURLExtractorService** which will extract the first URL (can be discussed to make it extract all URLs if needed), from a given text.
- Research and include all the standard as well as short URLs of commonly used services.
- Test its integration with **TalawaMetadataService** and do necessary modifications.
- Test the integration of **TalawaURLExtractorService**, **TalawaMetadataService**, and **TalawaMetadataWidget** altogether and implement cooldowns wherever needed. Integrate **TalawaMetadataWidget** to all widgets that show 'chat like' text, like chats, posts, etc.
- Integrate these three services in all the text fields (wherever needed, like creating a post, chat input field, etc) and thoroughly test the real-time fetch and display of metadata.
- Additionally, **implement emoji replacer**, i.e., Telegram and Slack prompt to swap **ASCII** emojis like :) to 😊. This can be a nice addition to our service.
- Discuss the state of **Plugin Architecture** with the mentee of the project 'Creating new features and refactoring existing features into Plugins' and implement the metadata field as a plugin in the server. This will mean that if **turned off**, metadata will be returned as **null**. Refactor the metadata feature as a plugin in the app. Now metadata widgets will act conditionally and will not be shown at all if turned off by the Admin.
- Write remaining tests and documentation and perform an e2e testing of the metadata feature.



## V. Mid-Term Evaluation (July 10 - July 14)

- A perfectly working **Metadata fetching service** will be ready and finely integrated with **the API**. The service will be as optimized as possible to use the **least possible bandwidth** and be a very less load to the server and database.
- Deep linking will be enabled again in perfect working conditions, and posts will now be sharable. The URL will open the app and respective post (if the user is logged into the respective organization) if the app is installed, and any custom link otherwise (we can later configure this to open Play Store / App Store if we release our app sometimes).


## VI. Week 7 and 8 (July 17 - July 30)

- Discuss the desired method of handling deep links and implement the same. I will prefer to go with **Firebase Dynamic Links**, but can also do any custom domain if needed.
- Add logic to generate links for posts and enhance existing deep link handler function to recognize links containing post information and open the right page accordingly.
- Discuss and add any custom URL scheme for Talawa.
- Test deep linking on both Android and iOS.

## VII. Week 9 and 10 (July 31 - August 14)

- Add a new **boolean** field to **posts** suggesting whether it is **private or public**. Do necessary changes everywhere in the codebase (including API, admin, and app).
- Prompt an option in the **Admin Panel** on the posts to **toggle their public/private behavior**.
- Add **routing** in the **top level of the API** (on which GraphQL runs). This route will handle any **link which has information about any post**. It will request the GraphQL instance internally to fetch the data of that post, and if it is public, will **respond with static HTML**, otherwise with a **Not found error**.

## VIII. Week 11 and 12 (August 15 - August 27)

- 
- Add a button to the static HTML saying **Open in the app**, which will open the post in Talawa, if installed. Additionally provide information about how to get the **API server** ready for **handling deep links**, so that this redirection happens automatically, **without the need for a button**.
  - Discuss with the mentor(s) and implement any other workaround for making Deep Linking work from the API server.

## IX. Final Evaluation (August 28 - September 4)


- The **metadata** feature will be **completely ready**.
- Users can type in some **text containing URL(s)** and the app will fetch information about the URL and show it **just above the input box**. They can then choose to not send the metadata by removing it (clicking on the cross button).
- A text containing metadata can be sent over the API and will be stored in the database. Subsequently, if a user fetches a post, chat, etc, they will also receive the metadata.
- If a Post or Chat contains some metadata, it will be shown above (or below, depending on our design choice) the main text. Users can view what the link contains from there.
- **Dynamic linking** will be **functional** again and **posts** can now be **shared**.
- **Admins** will now have the option of making a **post public** and sharing its public link which can be seen in both a browser and Talawa.

## X. Week 13 and 14 (September 4 - September 17)

- Install the [share\\_plus](#) plugin to facilitate **sharing of content from Talawa to other apps**.
- Adding sharing option to the content screens like **Chat, Posts, Events**, etc.
- Configure the Posts page to share the public or private link of the post depending on its visibility.
- Use [share\\_plus](#) in conjunction with [social\\_share](#), so that we can provide the user with the option of **sharing directly on Instagram stories etc**.

## XI. Week 15 and 16 (September 18 - October 1)




- 
- Install [receive\\_sharing\\_intent](#) to facilitate sharing content from other apps to Talawa. Though this plugin is a bit old, so I will be open to implementing it on my own if the mentor(s) advise so.
  - Configure the routes in the app so that the app asks the user where they want to share the incoming content in the app (**Chat or Post**) and open the route accordingly.
  - Test these receiving and sharing features on all social media apps in both Android and iOS.

## XII. Week 17 and 18 (October 2 - October 15)


- Start the implementation of in-app external video players.
- Create a new scaffolding widget called **TalawaPIPView**. This widget will be responsible for all the PIP effects we will be achieving. Understand and implement the positions where we expect the PIP window to be present in **floating mode**.
- Discuss and integrate [youtube\\_player\\_flutter](#) and any other platform-dependent video streaming plugin (like Dailymotion and Vimeo) and configure the plugin to our needs. The aim will be to make the most used video sites as easily accessible from the app as possible.
- Research **optimization techniques** are needed to **support two screens** (in-app floating video and background screen which will be Post Screen, Chat Screen, etc.) on low-end devices without issues.
- Thoroughly test animations, positioning, and conditionality of the **floating window**.

## XIII. Week 19 and 20 (October 16 - October 29)

- Wrap all of the screens which will need this feature with **TalawaPIPView** with the screen being the **backgroundPage** and **video\_player / youtube\_player\_flutter / any other platform-specific plugin** as **floatingWindow**.
- Thoroughly test this video-playing feature, subtitles in **youtube\_player\_flutter**, minimization/maximization, etc.
- Perform **complete e2e testing**. Test everything from writing a text containing URL, checking if metadata information shows up in real-time, checking if metadata gets sent to the server properly, fetching the text in another device, checking if metadata shows up properly there, and testing the in-app video player properly.

- 
- Discuss mentors and fellow mentees for any other mini-feature that can be added to this project.

#### XIV. Week 21 (October 30 - November 5)

- Implement any suggestions from fellow mentees and mentors.
  - Check and **finish all of the testing and documentation**. Will discuss with others what more documentation will be needed to be added so that future contributors can understand it well.
  - Thoroughly test that **each added and modified file** strictly has **100% test coverage**.
  - **This project will be ready to be used by an end user.**
- 

### Why Me?

I am an avid learner and can adapt to any new tech as soon as possible. When I saw the “**Create a custom linting system...**” issue, I immediately felt that it was the best way to brush up on my AST skills. I took the challenge and finally implemented **talawa\_lint**, Talawa’s own custom lint rule set. Be it our codebase’s migration to Flutter 3.7.3 or GraphQL Scalar migration, I always took the job and never feared changing the whole codebase for its betterment.

I have great communication skills and have always come to the front to help newcomers in our community, especially during the **talawa\_lint** integration.

I am **proficient in both front-end and back-end and anything in between**. Talawa’s codebase expands to App, Website, API, and Database... and having someone who has control of all of these can be a great resource to the organization. I have proved my efficiency in all of these by contributing to the respective repositories.

Overall, I believe I can be a great resource to Palisadoes Foundation and the open-source community in general.




## My Contributions

I am fairly active in the OpenSource community and contribute to a variety of projects.

### The Palisadoes Foundation

1. Further enhancing talawa\_lint [link](#)
2. Add a reference for our custom lint rules [link](#)
3. Feature Request: Using proper GraphQL scalars for different data types shared between clients and server [link](#)
4. Bump build\_runner from 2.1.10 to 2.3.3 [link](#)
5. Create a documentation comment linter that is compatible with DartDoc. Writing an AST parser from scratch [link](#)
6. Multiple widgets using the same GlobalKey [link](#)
7. Fix documentation in lib/views/main\_screen.dart [link](#)
8. The update method of User class updates wrongly [link](#)
9. Models: Tests for chat\_message.g.dart [link](#)
10. Models: Tests for user\_info.dart [link](#)
11. Views: Create tests for chat\_message\_bubble.dart [link](#)
12. Views: Create tests for event\_info\_body.dart [link](#)
13. Views: Create tests for user\_tasks\_page.dart [link](#)
14. Views: Create tests for create\_event\_page.dart [link](#)
15. Views: Create tests for join\_organisation\_after\_auth.dart [link](#)
16. Views: Create tests for main\_screen.dart [link](#)
17. Unit tests for widget event\_card.dart [link](#)
18. Pinned carousel widget test [link](#)
19. Add test for raised\_round\_edge\_button.dart [link](#)
20. Increasing coverage from 28% to 35% [link](#)
21. Fix overflow on org selection screen [link](#)
22. Overflow error [link](#)
23. Text font size difference on Language Selection screen [link](#)
24. [Bug report] Inconsistencies in Password RegEx [link](#)

Other PRs

- 
1. Bump SDK in workflow and codebase [link](#)
  2. Implement custom lint [link](#)
  3. Exclude test files and non-existent files from the Custom Lint workflow [link](#)

## CCExtractor

1. Update the flutter version and other packages [link](#)
2. Replace duration picker and version update [link](#)
3. Fix Location Permission [link](#)
4. Add the "coverage" folder in gitignore and bump the flutter version in workflow [link](#)
5. Add Mockito to support mock generation and update all packages [link](#)

## AppFlowy

[Bug] AppFlowy Editor Upgrade to 3.7.x [link](#)

## Faustus

Extend keyboard backlight driver support on Asus FX506LH.309




## Any Other Organization?

I have only focused on The Palisadoes Foundation and am trying GSoC only in it this year.



## About Me

I am Ravi Dev Pandey, a pre-final year pursuing a Bachelor of Technology at the Indian Institute of Technology (BHU), Varanasi. I have been into Computer Science since my childhood when I used to download ebooks on C++ and Game Development with my 2G network. I started my programming journey with QBASIC and have been in love with programming ever since. I proceeded in my programming journey by making some basic



games using a drag-and-drop game engine called Game Maker. Then I moved on to Python, C++, JS/TS, Rust, and Dart/Flutter. I am proficient in all of these and can learn new language frameworks quickly.

I am an avid learner and am always into new tech and concepts. Recently I grew interested in Conway's Game of Life and implementing it in C++/SDL in my free time nowadays. I have been into OSS for some time now, so much so that every software on my laptop is open source (from OS to terminal). When not doing Open Source, I do Competitive Programming, some Computer Graphics stuff, a bit of Image Processing, and anything Computer Science.

I love learning new concepts but can not bear repetition. Once I understand how something is done, implementing it over and over again is not enjoyable to me. I am very much into desktop customization and have a nearly custom GNU/Linux distribution installed.

I am the Lead of Systems Programming in my College's Programming Club and do and teach a lot of systems stuff like emulators, kernels, and compilers there.

Apart from tech, I like playing chess, and guitar and watching animated movies, especially the ones by Pixar and Dreamworks.



## Communication

(Timezone: Indian Standard Time (UTC +5:30))

Any mode of communication is totally fine for me. On weekdays I am available full-time between 9 AM IST (3:30 AM UTC) to 12 AM IST (6:30 PM UTC). On weekends I would love to have some exchanges with my fellow mentees and will try to learn from them. Timings can be flexible depending on what the mentors decide and are suitable for all.

There are currently no planned absentees from my side but I will let my mentor(s) know as soon as anything comes up.



## Post GSoC

If anything related to my work remains undocumented, untested, or has bugs, I will fix them on a priority basis and will continue to do until everything becomes clean. I will in general like to engage with newcomers and help them on Slack and GitHub. I will also discuss our app's release on Play Store and will help with anything related to it.

# THANK YOU