Introduction

Joplin's desktop app has a template feature i.e. users can make their templates and reuse the templates. For example - SWOT Analysis, Time Table, Meeting Notes, Journal, etc. and what not. But Joplin also has an awesome plugin architecture.

So it would be better if we repackage the template feature as a plugin due to many reasons like

- Better maintainability
- Features packaged as plugins would be more robust
- The feature would've better scope of improvement.

I basically propose to remove the existing feature and repackage it as a plugin while improving the template functionality.

Project goals

There are two main objectives of this project.

- Create a new plugin with improved template functionality
- Remove the template functionality from the main application

Implementation

Creating a plugin for template functionality

Plugin design

The main improvements I plan to bring in the template functionality are as follows.

- Removal of Refresh Templates - Currently there is a refresh templates option in the templates menu that reloads the templates from the template directory. Removing this option and loading the templates everytime user tried to select a template would improve the usability of the feature.

- Default Templates - It is highly likely that most users have one or two templates that they use the most. And they want to create a new note using that template they'll currently have to create a new note and then select a template that they want to insert. So, to make it more useful I'd like to introduce default templates feature. This feature was also once requested in the Joplin Forum, however my proposal of this feature is not exactly similar to the request of this user. The details of this feature can be found below in this

section itself.

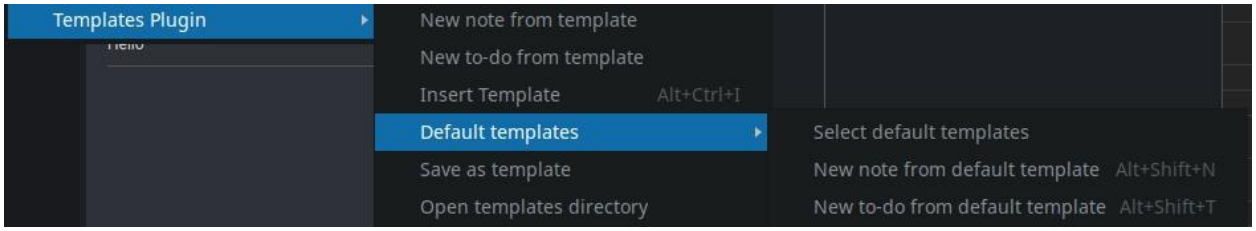
- Pre Installed templates - When I tried to see the templates functionality in Evernote, I found that they provide some templates by default such as Meeting Note, Essay Outline, Meal Planner, Daily Reflection etc. I personally think that it would be great if we do it too in this plugin.

- Save as template - Currently, if a user wants to create a new template they have to open the templates directory and create a new markdown file and maybe then use a third party editor to write the template. The point is that there is no convenient way to do so. So, I'd like to add a save as template option both in the menu and as a toolbar icon.

- Toolbar icons - We can include two icons Insert Template and Save as template in the editor toolbar which I believe would be the best fit.

NOTE - The features are not limited to the above list. I'll add more features which may be requested by users afterwards or after discussing with thementors.

Basically, the plugin in the menu would look something like this.



Here, *new note from template*, *new to-do from template*, *insert template* would work in the same way as they currently do, the only difference being that there will be some pre-installed templates in the dropdown too.

The *open templates directory* will work in a similar way as it does now. It open a file manager with the templates directory.

The *save as template* option would save the current markdown content of the note after taking the name of the template as input from the user.

In the *default template* functionality the user will be able to set two default templates one for notes and one for to-dos. And as it can be seen in the picture there are two options: new *note from default template* and *new to-do from default template.* As the name suggests they would create a new note and to-do from their respective defaults. These options also have keyboard shortcuts which would make this feature a lot more usable.

Also, we can add two toolbar buttons that would look something like

The last two icons denote “Insert Template” and “Save as template”. In my opinion these two options would make the most sense in the editor toolbar.

Plugin Implementation

As the functionality of the template plugin will be similar to the current implementation of the plugin. Most of the implementation of the plugin could be similar to TemplateUtils.js and selectTemplate.ts. Also, the options in the menu could be added using the Menu Joplin Plugin API.

Let’s discuss the implementation of the plugin with respect to individual features.

Insert template

Once the user selects the template it will be very easy to insert the template using the insertText command. The main implementation detail to be discussed is the implementation of the prompt that will be used to take input from the users. I noticed that there is a Dialog plugin API which can be used to do this. But we also have a PromptDialog component that can be used to achieve this. I would personally prefer using this prompt because it has an autocomplete feature and better UI.

We can use this prompt from the plugin by creating a command for it in the Joplin codebase something like this.

```
import { CommandRuntime, CommandDeclaration, CommandContext } from
'@joplin/lib/services/CommandService';
import { _ } from '@joplin/lib/locale';

export const declaration: CommandDeclaration = {
   name: 'showPrompt',
};

interface PromptConfig {
   label: string;
   inputType: 'dropdown' | 'datetime' | 'tags' | 'text';
   value: any;
   autocomplete: any[];
}
```

```
export const runtime = (comp: any): CommandRuntime => {
   return {
       execute: async (_context: CommandContext, config: PromptConfig) => {
           return new Promise((resolve, reject) => {
               comp.setState({
                   promptOptions: {
                       label: _(config.label),
                       inputType: config.inputType,
                       value: config.value,
                       autocomplete: config.autocomplete,
                       onClose: async (answer: any) => {
                           if (answer) {
                               resolve(answer);
                           }

                           reject();
                           comp.setState({ promptOptions: null });
                       },
                   },
               });
           })
       },
   };
};
```

NOTE - This is just an example command which I used as a proof-of-concept to test the functionality.

New note / to-do from a template

These options can be implemented similarly as the above “Insert Template” option, the only difference being in this case we can invoke “newNote” and “newTodo” commands respectively.

Save as template

To implement this functionality we can first use the selectedNote plugin API to get the content of the current note and then use the dialog box to get the input of the templates name in the similar way as we do in the above options and then we can save the content of the note in the templates directory.

Open template directory

Currently we use bridge.ts for opening directories. To be able to do it using a plugin we can create a command in the Joplin desktop app something like this.

```
import { CommandRuntime, CommandDeclaration, CommandContext } from
'@joplin/lib/services/CommandService';
import { _ } from '@joplin/lib/locale';
import bridge from '../../../services/bridge';

export const declaration: CommandDeclaration = {
   name: 'openExternalItem',
   label: () => _('Open external item')
};

export const runtime = (): CommandRuntime => {
   return {
       execute: async (_context: CommandContext, path: string) => {
           await bridge().openItem(path);
       }
   }
}
```

Then we can simply invoke this command in the plugin to open a directory.

Default templates

We can use the similar approach as stated above while selecting / inserting the templates. The main point to be discussed is the implementation of storing the default templates for the users. I think the best way to do that would be to use the Settings plugin API to register a setting and then storing the default values in the registered setting.

Toolbar buttons

Toolbar buttons can be easily implemented using the Toolbar Buttons plugin API by connecting the buttons to the commands we'll register for the respective options.

Testing

Currently the plugin framework doesn't include the testing setup but I'll set up Jest in the plugin repository and use Github Actions as a CI to test every Pull Request and push to the plugin repository. I'll try to unit test most of the code mocking some necessary functions as it is a plugin.

Documentation

I'll document the usage of the plugin in the README of the plugin repository. Also, I'll try to use comments while coding wherever necessary to make the code easy to understand.

Removing the current template functionality

Main Changes

While removing the functionality, it is important to leave no unused code behind. So, I've already figured out the main places that need to be modified.

- We store the location of the templates directory. So, we can remove that setting from the Settings.ts file.
- We can remove the templates feature from the MenuBar.tsx.
- We can remove the TemplateUtils.js.
- We can remove the selectTemplate.ts.
- We can remove / refactor the code dependent on the above functions / files removed such as the newNote command i.e. dependent on the TemplateUtils.

Documentation Updates

After the plugin is built and the feature is removed from the main application successfully, we can update the documentation on the home page of Joplin's website here accordingly.

Timeline

Conflicts

I have no conflicts in the official GSoC coding period. I'd be available full time by the minimum of 40 hours per week in the summers.

Communication

I would be available according to the IST timezone but am flexible according to the timezone of my mentor to ensure a clear and regular communication. I plan to communicate my progress daily to my mentor on whatever platform my mentor is comfortable. I also plan to write weekly blog posts about the progress in my GSoC project.

Expected deadlines

Community Bonding (May 17 - Jun 6)

- Discuss my project with my mentors and the community to see if there are any more improvements that can be done in the feature.
- Try to complete all my previous pull requests to Joplin, so that during the GSoC coding period, I can concentrate on my project.
- Start a blog to document the coding period.
- Interact with the maintainers to get my doubts / technical design decisions cleared if there are any left.

Coding Phase 1 (Jun 7 - Jul 16)

Weeks 1-3

- Initialize the plugin repository.
- Make changes in the Joplin main app to add insert template functionality.
- Add insert template functionality in the plugin.
- Setup jest and unit testing in the plugin.
- Setup CI/CD using Github actions for unit tests and lint checks to ensure quality of the code is good.
- Make changes in the Joplin main add to allow opening template directory.
- Add Open Template directory functionality.
- Write unit tests for the plugin code written.
- Make weekly blog posts on the progress.

Weeks 4-5

- Add new note / to-do from template functionality.
- Add save as template functionality.
- Add toolbar buttons.
- Write unit tests for the plugin code written.
- Make weekly blog posts on the progress.

Coding Phase 2 (Jul 17 - Aug 16)

Weeks 6-7

- Add the default template functionality.
- Add some pre-installed templates.
- Write unit tests for the plugin code written.
- Document the usage of the plugin completely in the readme file.
- Make weekly blog posts on the progress.

Weeks 8-10

- Wrap up the plugin so that it can be used completely.
- Discuss any improvements / features that can be done further in the plugin.
- Remove the template functionality from the main Joplin desktop app safely so that it doesn't break anything.
- Write relevant unit tests.
- Update documentation on the Joplin home page accordingly so that the community can know that the Template functionality is now replaced by a plugin.
- Make weekly blog posts on the progress.
- Make a final GSoC project report.

Post GSoC

I found the codebase and the idea of Joplin to be pretty awesome. Therefore I plan to keep contributing to Joplin even after the GSoC ends. I personally think there are a lot of things I can learn if I contribute to Joplin. Some of the things I find pretty amazing is the plugin design and structure, plugin framework, organized plugin and data APIs.

About me

Basic Info

Name - Nishant Mittal
Github Username - nishantwrp
Field of Study - Computer Science
Resume - Resume
Email - mittalnishant14@gmail.com
Phone - +91 9417907862
Country - India

Introduction

I am Nishant Mittal, passionate programmer, third year undergrad at IIT BHU. I have experience in building REST APIs, JAMstack / full-stack web applications, test-driven development. I love exploring new tech and have worked with various technologies and languages. I love contributing to the open source community whether it be contributing to open source software or building something that can be used by other developers. You can view my personal projects that I've worked on here.

Contributions to Joplin

Pull requests

- Desktop: Fixes #4788: Set plain/text clipboard while copying (Merged)
- Desktop: Partially fixes #4602: Remove timestamp while copying images (Merged)
- API: Fixes #4575: Set the value of filename while creating a resource (Closed) [It was decided to remove the filename field]
- API: Fixes #4655: Don't compress image when resource is added through api (Merged)

Issues / Bugs Reported

- Jolpin doesn't create it's own copy of images while pasting (#4767)
- Can't copy a single image to online editors like Google Docs (#4764)

GSoC 2020 with Oppia

Last year, I successfully completed a GSoC project with Oppia Foundation. Thus, I am experienced with the collaborative environment in open source projects. Even after GSoC, I was involved with Oppia Foundation for about 7 months. My contributions to oppia foundation can be found here.

Like Joplin, Oppia also encourages it's contributors to unit test the code. During the GSoC period I followed TDD and unit tested 100% of the code written by me both in TypeScript and Python. Also, a part of my project was documenting the working of webpack and typescript checks in Oppia. The details of the documentation and GSoC project and be found in my final GSoC blog post.

Contributions to other open source projects

I am an open source enthusiast, and apart from Joplin and Oppia I've contributed to many other open source projects like Forem, Creative Commons, Zulip etc. that involved various technologies like Ruby on Rails, Node.Js, PHP, Python etc. The complete list of contributions can be found in my Github profile contribution overview chart.

Multiple Proposals

Joplin is the only organization I'll be submitting a GSoC proposal for.