Anjula Karunarathne
anjulakarunarathne@gmail.com

Custom Keyboard Shortcuts

GSoC 2020 Project Proposal

Introduction

Keyboard shortcuts are combinations of keys that provide an alternative way to do something that you'd typically do with a mouse. These shortcuts can expedite common operations by reducing complex input sequences to a few keystrokes.

Joplin desktop application supports keyboard shortcuts to a certain degree. However, it doesn't allow users to configure keyboard shortcuts according to their preferences. Custom keyboard shortcuts would be a great addition for all Joplin users including power users and those who are migrating to Joplin from other note-taking applications. People also do not have the same keyboard layout; not everyone has F6 necessarily. It would solve conflicts between different preferences of workflows, and increase discoverability for all keyboard shortcuts.

I wish to implement a shortcut editor in the Options panel which allows the user to configure keyboard shortcuts.

Project Goal

Implement support for custom keyboard shortcuts in the Joplin desktop application. Users would be able to view keyboard shortcuts and edit them from the keyboard shortcuts editor in the Options panel.

Support for keyboard shortcut schemes will be provided with several default schemes available out of the box. These can be shortcut schemes from alternative note-taking applications. Users will also be able to import other shortcut schemes and export their current shortcut scheme as well.

Implementation

Joplin uses keyboard shortcut functionality provided by Electron. Electron applications define key combinations as Accelerators, which are strings that usually contain multiple modifiers and a single key code, all combined by the + character.

Currently, there are two types of keyboard shortcuts being used in Joplin.

1. Global shortcuts

   Using the `globalShortcut` module, the application can detect keyboard events even when the application does not have keyboard focus. A global shortcut can be defined with an accelerator and a function that will be triggered when the key sequence in the provided accelerator has been pressed.

   Joplin uses a global shortcut to toggle window visibility. Refer following snippet:

   ```
   globalShortcut.register('CommandOrControl+Alt+J', () => {
              this.toggleWindowVisilibity();
   });
   ```

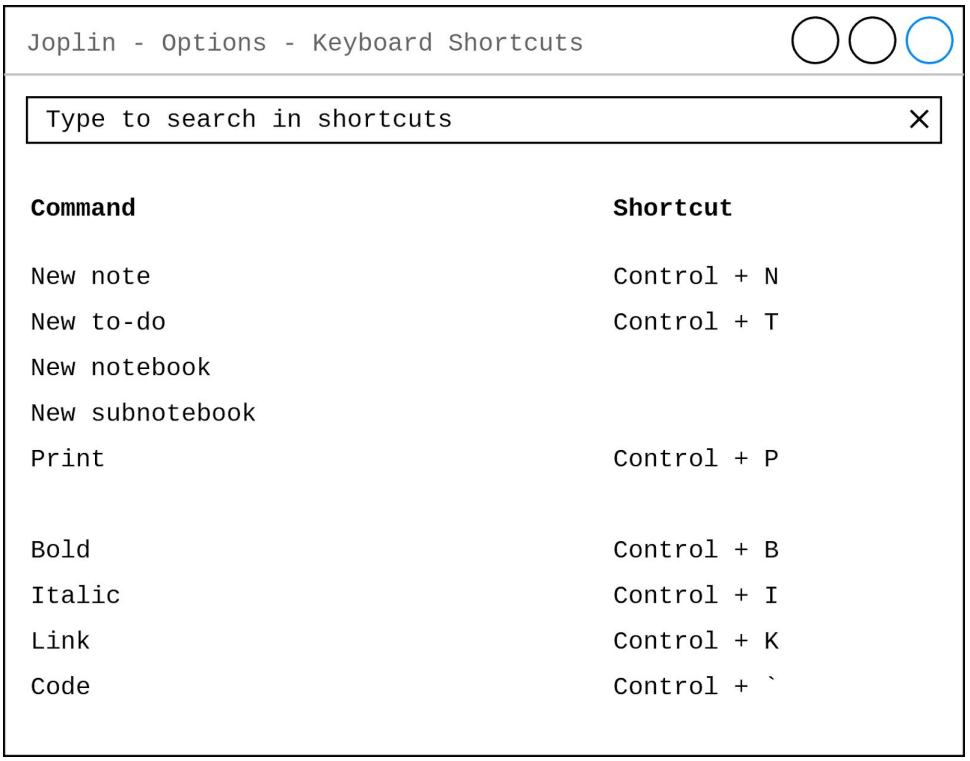
2. Local shortcuts

   Whenever creating a menu item using the `Menu` module, an accelerator property can be specified. It will be triggered only when the app is focused on.

   Joplin uses this functionality to create keyboard shortcuts for menu items. Refer the following snippet:

   ```
   {
       label: _('Toggle sidebar'),
       screens: ['Main'],
       accelerator: shim.isMac() ? 'Option+Command+S' : 'F10',
       click: () => {
           this.dispatch({
               type: 'WINDOW_COMMAND',
               name: 'toggleSidebar',
           });
       }
   }
   ```

The first step would be storing the accelerators in a central location and displaying them in a separate panel within Options. I'd suggest a screen as in the mockup below:



All Joplin applications share the same library located in `ReactNativeClient/lib`. Within the library is the `Setting` model, which contains every configurable setting of the Joplin platform. Each setting has the property `appTypes` which determines if they belong to the CLI, mobile and/or desktop application. There are also the properties section, `label`, `value`, `type` and etc. which determines the respective properties of that particular setting. These settings are rendered separately by the desktop, CLI and mobile application.

A keyboard shortcut could be specified as an individual setting that belongs to the desktop application. It has the `value` of an accelerator. An additional `type` for the keyboard shortcut settings can be implemented to render the setting as shown in the above mockup. For displaying the settings on a separate screen, the `section` property can be used. Refer the following snippet for an example setting definition:

```
newNoteShortcut: {
    value: 'CommandOrControl+N',
    type: Setting.TYPE_SHORTCUT,
    section: 'shortcuts',
    appTypes: ['desktop'],
    label: () => _('New note'),
}
```

We can then update existing menu items to use the new accelerators acquired from the `Setting` model. A setting value can be accessed through `Setting.value()` which then can be used in `accelerator` property of the menu items.

The next step would be making the settings editable. A custom field can be provided to the user for each keyboard shortcut setting in which when highlighted, captures the pressed key combination. The combination should then be validated to contain the appropriate key combinations as specified in the Accelerator documentation. It may contain multiple modifiers and just one key. If the keystrokes are valid and it is not used in any other existing shortcut, we can generate an accelerator string by combining them with the `+` character. When combining the accelerator, modifier(s) should be first and then the key should be at last. The resulting string can then replace the value of the particular keyboard shortcut setting.

An important thing to consider is that macOS and Windows/Linux platforms have some variances when it comes to keyboard layouts. They don't necessarily have the same modifiers. For example, macOS doesn't have a `Control` key, but a `Command` key. Therefore validating a key combination would require research and thorough testing on different platforms.

Refer below for a simple implementation. It contains the logic for capturing the pressed key combination and validating it. Please note that it's a prototype and only works on PC.

See it working on: https://codesandbox.io/s/anjulalk-custom-keyboard-shortcuts-wlc7i

```
class ShortcutCapturer extends Component {
 state = { preview: [], accelerator: `` }
 modifiers = [`Control`, `Alt`, `Shift`];
 keys = `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.split('');

 handleShortcut = (e) => {
   e.preventDefault();

   switch (e.key) {
     case 'Enter':
       console.info(`Validating shortcut..`);
       this.validateShortcut();
       break;
     case 'Escape':
       console.warn(`Shortcut discarded!`);
       this.discardPreview();
       break;
     default:
       const preview = [];
       if (e.ctrlKey) preview.push(`Control`);
       if (e.altKey) preview.push(`Alt`);
       if (e.shiftKey) preview.push(`Shift`);
       if(this.keys.includes(e.key.toUpperCase()))
           preview.push(e.key.toUpperCase());
       this.setState({ preview });
   }
 }

 validateShortcut = () => {
   const { preview } = this.state;
   if (
     preview.length
     && this.modifiers.includes(preview[0])
     && this.keys.includes(preview[preview.length - 1])
   ) {
     console.info(`Saving shortcut ${this.state.preview.join('+')}`);
     this.setState({ accelerator: this.state.preview.join('+') })
```

```
    } else {
      console.error(`Invalid shortcut
          ${this.state.preview.join('+')}`);
      this.discardPreview();
    }
  }

  discardPreview = () => {
    this.setState({ preview: [] });
  }

  render() {
    const { preview, accelerator } = this.state;
    return (
      <div className="App">
        <header className="App-header">
          <input
            type="text"
            readOnly
            placeholder={accelerator || "Capture shortcut.."}
            value={preview.join('+')}
            onKeyDownCapture={this.handleShortcut}
            onBlur={() => this.discardPreview()}
          />
        </header>
      </div>
    );
  }
}
```

The state contains two variables, `preview` and `accelerator`. The former is an array that stores the pressed keys. Using an array for storing key combinations allows easy validation. It's used for previewing the pressed keys to give real-time feedback to the user. For generating the `accelerator` string, simply combine the content of the `preview` array with + character.

The input field captures key combinations whenever it's focused. If the field is blurred or the user presses `Escape`, it discards the captured key combination. When capturing the combination, modifiers are pushed to the `preview` array first and then regular keys. This way, we can generate the `accelerator` string very easily.

To save the shortcut, the user has to press `Enter`. The shortcut is validated to contain at least one modifier and exactly one key. If the shortcut is a valid accelerator, it should also be checked against other shortcuts to be sure that it's unique.

There are many improvements to be made to this prototype. It can be properly implemented by testing with various keyboard layouts and platforms. Unit testing the capturer and validator functionality is possible via mocking the keyboard events.

Additionally, users can be allowed to export/import their current shortcut configuration. This can be done in a similar fashion as in the CLI application. In the CLI, it uses a JSON array for storing the keymap. We can use a similar format here to export/import shortcut configuration. Refer following entry snippet for an example:

```
{ "keys": “CommandOrControl+N”, "command": "New note", … }
```

Users can also be provided some default configurations to choose from. These can be extracted from popular alternative note-taking applications that most users would be migrating from.

Timeline

Community Bonding	April 28 - May 17	Get familiar with the codebase Communicate with project mentors Explore the technology stack
	May 01 - May 17	Refine the project ideas with mentors' consultation Finalize the idea and project structure
Phase 1	May 18 - June 20	Display all shortcuts in a separate screen Move keyboard shortcuts to Settings Perform tests and inspect for any arisen issues
Phase 2	June 21 - July 17	Allow editing keyboard shortcuts Capture and validate pressed key combinations Save valid keyboard shortcuts Write tests for key capture functionality
Phase 3	July 18 - Aug 09	Implement support for exporting and importing keyboard shortcut configurations Add default presets from shortcuts used in alternative note-taking applications Write tests for export/import functionality
Finalization	Aug 10 - Aug 27	Submit the finalized project

I can easily devote 40-45 hours per week to the project. However, I will be having my end-semester exams in early-August. Therefore, I may finish my work earlier, but no later than the specified time periods.

About Me

Name	Anjula Lakshan Karunarathne
E-mail address	anjulakarunarathne@gmail.com
GitHub	anjulalk (https://github.com/anjulalk)
LinkedIn	anjulalk (https://linkedin.com/in/anjulalk)

I'm Anjula Karunarathne, a third-year Computer Science undergraduate from the University of Colombo School of Computing, Sri Lanka. I have been actively involved in software projects based on web technologies, with JavaScript being my primary programming language of choice.

I discovered Joplin while searching for a good free and cross-platform note-taking software. I've used Google Keep previously, but it lacked any kind of organization methods which really limited its usability. On top of that, it didn't even have a desktop application. After a while, I switched to Evernote, but the drawbacks of their free plan, the biggest one being no offline access on their mobile application forced me to look for other alternatives. Joplin solved all those issues for me I've been using Joplin for more than 4 months now, and I never looked back again.

I have contributed to Joplin. See my pull requests below:

- (Under review) Desktop: Go To Anything by body (#2686)
- Desktop: Fix undefined showButtonLabels (#2619)
- Desktop: Fix incorrect location format (#2480)
- Desktop: Fix button label wrapping (#2708)
- Made some changes to BUILD.md have better readability (#2542)

I've received great support from the Joplin community. It has actually been an honor to contribute to Joplin and to be part of this awesome community.

I've also contributed to other open-source projects in my free time. One of them is LabelLab (built with React, Node.js, and MongoDB) which is an open-source web-based image labeling and classification program. Below are some of my merged pull requests:

- Revamp login and register interfaces (#332)
- Update eslint configuration (#303)
- Fix missing key prop (#329)

- Cleanup environment variable: remove colon (#351)
- Update README.md (#280)

Below are some of the other software projects I've been actively involved in:

- Athwela (Angular, Node.js, MongoDB)

  Athwela is an online fundraising platform built as a group project to help raise funds for social welfare causes. It allows users to host fundraising campaigns. Donors can donate funds to which campaign they see fit through the system.

- InEvents (Angular, Node.js, MongoDB)

  InEvents is a modern event registration management platform. It allows users to host events, create dynamic forms for applicants, manage registrations, select applicants and etc. It has a web client and native Android and iOS clients.

- express-starter (Node.js)

  Bootstrap an Express application quickly and integrate dependency management and version controlling with ease.

- GoldenArrow (CodeIgniter, MySQL)

  Sports club management system for GoldenArrow Football Club built with CodeIgniter framework. It allows club administrators to manage resources available for the club. Through the system, new tournaments, matches and practice sessions can be scheduled. Members can also pay membership fees right through the system.

- Webpad (PHP, MySQL)

  A simple online note-taking application built with PHP and Bootstrap.

I think the experience I got with developing these projects will help me in this project as well. I also haven't submitted proposals to other organizations; only to Joplin. I believe this project would be a great opportunity for me to make Joplin better while learning new awesome things from the Joplin community.