Johanna - Project Portfolio

PROJECT: Notably

Overview

Notably is a **note-taking desktop app**, optimized for those who **prefer to work with a Command Line Interface (CLI)** while still having the benefits of a Graphical User Interface (GUI). It aims to help the tech-savvy NUS students to get their notes taken down **faster than using traditional GUI apps**.

Summary of contributions

- **Major enhancement**: **Suggestion Engine**
  - What it does: `SuggestionEngine` actively give users an updated list of suggestions of notes as the users type their input. Furthermore, it displays a response text to enable users to understand the meaning behind their inputted command.
  - Justification: `SuggestionEngine` played a pivotal role in providing a seamless user experience of managing the user's notes. This feature is curated for our target users (NUS tech-savvy students) who often have a large number of notes and thus may find difficulty traversing all of their notes to get to a particular note. Therefore, `SuggestionEngine` improves the user experience significantly by giving this convenience.
  - Highlights: It gives response text and suggestions (if any) for all available valid commands in Notably. The implementation was quite challenging as different commands require different implementation.
- **Major enhancement**: **Search feature**
  - What it does: The Search feature searches the occurrences of a keyword in all of the notes, not just the currently opened one. It can also search for partial or incomplete words of the note's content and count the number of times the keywords appear in the note. It then displays the list of suggestions of notes to the users with the most relevant search result being on top of the list.
  - Justification: Students often remember a certain keyword from their note but can't precisely remember where it is located. The Search feature thus gives the convenience to the users to find the relevant note as suggestions are sorted based on the highest number of occurrences. If the number of occurrences is the same, the suggestions listing will based on their respective positions in the hierarchical notes arrangement.
  - Highlights: The feature is complete as it traverses through all of the notes and can even search for incomplete words, hence giving the relevant suggestions to the user even before the user has finished typing.
- **Code contributed**: [Functional code]

- **Other contributions**:
  - Project management:
    - Wrote most parts of the README: #51
    - Maintained the issue tracker
  - Enhancements to existing features:
    - Wrote additional tests for existing features to increase coverage significantly (Pull requests #428, #470)
  - Documentation:
    - Updated the Search and Auto-Suggestions feature in UG: #461
  - Community:
    - PRs reviewed (with non-trivial review comments): #106, #127, #279, #291, #432
    - Reported bugs and suggestions for other teams in the class: #1, #2, #3, #4, #5, #6, #7, #8, #9, #10, #11, #12, #13, #14, #15, #16

Contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Find a note based on certain keywords: `search`

If you need to look for a note that contain a specific keyword, use the `search` command and Notably when show you the result sorted by the number of matches in the note.
Format: `search [-s] KEYWORD`

NOTE

- `search` looks through **all** the notes that you have
- Partial matches work as well!
- Matches are case insensitive, meaning it will find the word no matter if it is in uppercase or lowercase or even mixed-case

Example: Searching for the keyword "Computer science"

1. Let’s look through all the notes that we have saved, for the keyword "Computer science"

search -s Computer science

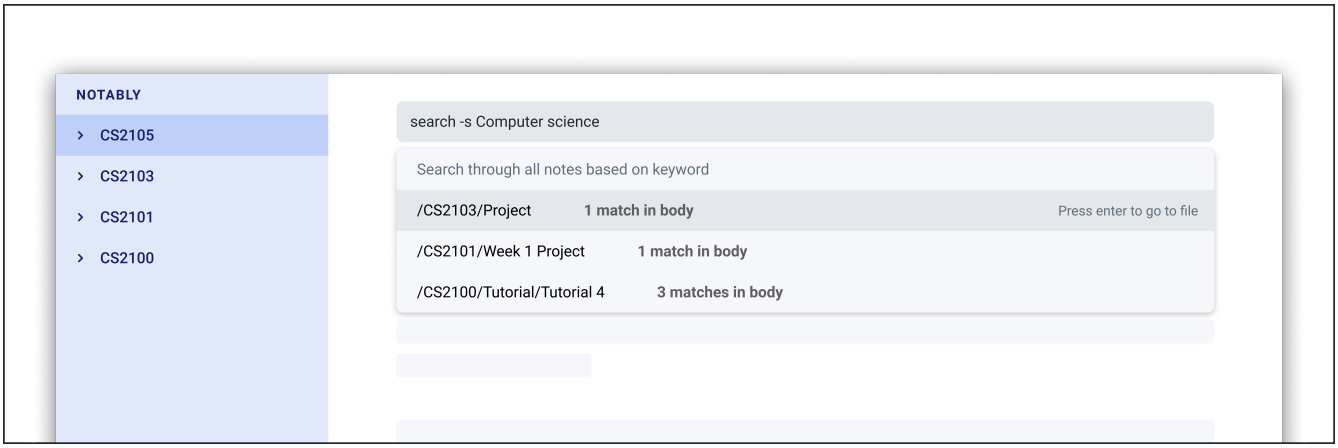

Figure 1. Demo for the `search` command

Auto suggestions

As you type, Notably will provide you suggestions. You can then press kbd:[Enter] to select the first suggestion.

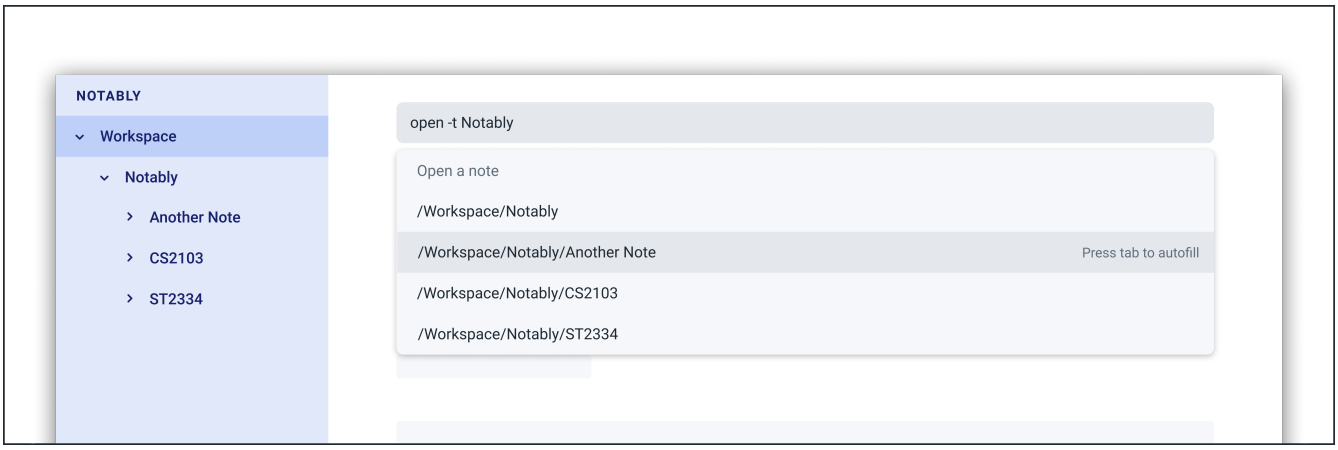

Figure 2. Demo for the suggestions feature

(Coming in v2.0) Suggestion response text when opening or deleting a parent note

Variations of path with `../` (e.g. `open ../`, `open Note/../Note`, etc.) will not generate a comprehensive response text. Currently, typing `open ../` will generate a response text of `Open a note titled "../"` instead of `Open a parent note`.

Contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.

Suggestion Engine

Rationale

`SuggestionEngine` allows the users to traverse their notes conveniently, without having to remember the hierarchical structure of their notes. `SuggestionEngine` gives users the meaning of the command they input and a list of notes suggestions that they want to open/ delete/ search/ edit.

Current implementation

1. `Logic` uses the `SuggestionEngine` class, to handle the user input.
2. According to the command the user inputs, `SuggestionEngine` will create a `XYZSuggestionArgHandler` or `ABCSuggestionHandler` object which implements `SuggestionArgHandler` and `SuggestionHandler` interface respectively. `XYZSuggestionArgHandler` are for commands that require argument parsing, i.e. `open`, `delete`, `search`, `new`, whereas `ABCSuggestionHandler` are for commands that do not require argument parsing, i.e. `edit`, `exit`, `help`.
3. If `SuggestionArgHandler` object is created: the `responseText` in the `Model` will be updated. This case will also result in the creation of `XYZSuggestionGenerator` object (except for `new` command) which implements `SuggestionGenerator` interface. `XYZSuggestionGenerator` is then executed by the `SuggestionEngine`.
4. If `SuggestionHandler` object is created: the `responseText` in the `Model` will be updated.
5. The `Model` could be affected in 2 ways:
    - Update `responseText` of the `Model` (by the `SuggestionHandler`): for instance, `open` command will set the `responseText` in the model as "Open a note".
    - Store a list of `SuggestionItem` in the `Model` (by the `SuggestionGenerator`).
6. The UI will then be able to retrieve the `responseText` and list of `SuggestionItem` from the `Model` to be displayed to the user.

Design considerations

Aspect: Design with respect to the whole architecture

1. `SuggestionEngine` is segregated from `Parser` in order to differentiate the logic when the user has finished typing and pressed kbd:[Enter] (which will be handled by `Parser`) in contrast to when the user presses `tab` to take in the suggestion item.
2. In order to keep the App's data flow unidirectional, `SuggestionEngine` will update the `responseText` (which tells the user the meaning of his command) and the list of `SuggestionItem` into the `Model`. Thus, by not showing the `responseText` and suggestions immediately to the UI, `SuggestionEngine` will not interfere with the `View` functionality.
3. `SuggestionArgHandler`, `SuggestionGenerator`, `SuggestionItem`, and `SuggestionModel` are implemented as interfaces, in an attempt to make the design of the `SuggestionEngine` component resilient to change.

Use case: Search notes using the Auto-suggestion feature

MSS

1. User types in a keyword of a note's content that he wants to open.
2. Notably lists out the relevant search results, with the most relevant being on top of the list (based on the keyword's number of occurrences in the note).
3. User chooses one of the suggested notes.
4. Notably opens the chosen note.

   Use case ends.

Extensions

2a. No suggestion is being generated.

2a1. Notably displays a response text, indicating that the user is trying to search through all of the notes using that particular keyword.

2a2. Since the empty suggestion conveys that the keyword cannot be found, the user enters a new data.

Steps 1a1-1a2 are repeated until the data entered is correct. Use case resumes from Step 3.

Use case: Open/ Delete notes using the Auto-suggestion feature

MSS

1. User types in an incomplete path or title of a note.
2. Notably lists out suggestions of notes.
3. User chooses one of the suggested notes.
4. Notably opens/ deletes the chosen note.

   Use case ends.

Extensions

1a. Path or title contains invalid character(s) (symbols `-` or `` ` ``)

1a1. Notably displays a response text, indicating that the path or title is invalid.

1a2. User enters a new data.

Steps 1a1-1a2 are repeated until the data entered is correct. Use case resumes from Step 2.

1b. Path or title does not exist

1b1. Notably displays a response text, indicating that the user is trying to open/ delete the note with that particular path or title.

1b2. Notably does not generate any suggestions, which means the note cannot be found.

1b3. User enters a new data.

Steps 1b1-1b3 are repeated until the data entered is correct. Use case resumes from Step 2.