Documentation tooling (Improving Documenter.jl for Julia Language)

Contributor: Shravan Goswami
Mentor: Morten Piibeleht

Abstract

Documenter.jl is a documentation generator for Julia language projects that produces HTML, PDF, and other formats from Markdown files. The aim of this project is to improve the functionality of Documenter.jl by adding new features, fixing bugs, and enhancing existing documentation. The proposed improvements include:

- Implement new feature to convert Jupyter notebooks to Markdown format
- Enhancing the handling of code snippets in the generated documentation
- Adding new features like feedback system and keyboard shortcuts
- Improving the UI/UX of the generated documentation
- Implementing ElasticSearch Backend to improve search functionality in documentations

Technical Details

Generating documentation from Jupyter notebooks

Jupyter notebooks are a popular way of creating and sharing interactive documents, but Documenter.jl currently does not support generating documentation from Jupyter notebooks. I propose to implement this feature by adding support for parsing and converting Jupyter notebooks to Markdown format, which can then be used by Documenter.jl to generate documentation.

Integrating Jupyter notebooks into Documenter will also provide a more interactive and user-friendly documentation experience. Users will be able to run code snippets directly within the documentation and see the results in real-time, which can greatly improve the understanding and usability of the documented code. Additionally, Jupyter notebooks also allow the inclusion of images, videos, and interactive visualizations, which can further enhance the documentation's effectiveness. This can result in a more engaging and accessible documentation experience for users, ultimately improving the usability and adoption of the documented code.

The technical steps involved in this feature include:

1. Converting Jupyter notebooks to markdown format using nbconvert.
2. Parsing the markdown files using a markdown parser library.
3. Generating HTML pages using Documenter.jl static site generator.
4. Adding custom styling to the HTML pages to ensure consistency with the Julia ecosystem.

The project requires knowledge of Jupyter notebooks, markdown parsing, web development, and Documenter.jl. The feature will provide users with a convenient and flexible way to document Julia projects using the popular Jupyter notebook format.

Enhanced handling of code snippets

The presentation and formatting of code snippets in the generated documentation could be improved to make them more readable and accessible.

- Features such as line numbering and code collapsing could be added to enhance the readability and usability of the documentation. Overall, enhancing the handling of code snippets in Documenter.jl would improve the usability and accessibility of the generated documentation, making it easier for users to understand and work with the code.

Feedback system for Documentation

Adding a feedback system to Documenter can be a valuable addition as it allows users to share their experiences, report bugs, suggest improvements, and provide general feedback on the documentation. This can be done by adding a simple feedback form or button at the end of each page, where users can leave their comments.

The feedback system can also include an email notification feature that alerts the documentation team when a new feedback message is received. This can help in addressing user issues in a timely manner and improving the overall quality of the documentation.

Moreover, the feedback received can be used to prioritize the improvements and updates to the documentation, based on the most common issues or suggestions reported by users. This can help in ensuring that the documentation is meeting the needs of the users and providing them with a better experience.

- I propose to implement a feedback system directly in Documentation.

Adding all possible Keyboard shortcuts

Keyboard shortcuts can greatly improve the usability and user experience of any software application. In the context of Documenter.jl, adding keyboard shortcuts can make it easier and faster for users to perform common tasks, such as navigating between pages, searching for content, and editing documentation.

One potential approach to implementing keyboard shortcuts in Documenter.jl is to use a JavaScript library like Mousetrap.js. This library makes it easy to define custom keyboard shortcuts that trigger specific actions, such as showing or hiding the table of contents, or toggling between editing and preview modes.

To implement keyboard shortcuts in Documenter.jl, we could start by defining a set of default shortcuts that cover common tasks. These could include shortcuts for navigating between pages (e.g., using the arrow keys to move forward and backward), searching for content (e.g., using Ctrl+F to open a search box), and toggling between editing and preview modes (e.g., using Ctrl+Shift+E).

We could also make it easy for users to customize these shortcuts by providing a configuration file where they can specify their preferred shortcuts. This file could be loaded at runtime to override the default shortcuts.

Overall, adding keyboard shortcuts to Documenter.jl can improve the productivity and efficiency of users, and make it a more user-friendly and enjoyable documentation tool.

Improving search functionality

The goal of this project is to improve the search functionality of Documenter.jl by implementing an ElasticSearch-based search backend. The current search functionality of Documenter.jl uses lunr.js, which can result in slow loading times due to the large size of the search index.

1. To set up the ElasticSearch instance, the project will require knowledge of ElasticSearch and server administration skills.
2. Once the ElasticSearch instance is set up, the next step will be to index the documentation. This will involve writing a script that extracts relevant information from the documentation and sends it to the ElasticSearch instance to be indexed. This will require knowledge of Julia and ElasticSearch.
3. The next step will be to implement the search API. This will involve writing a custom search API that communicates with the ElasticSearch instance and returns search results. This will require knowledge of web development and Documenter.jl.
4. The search API will need to be designed to handle different types of search queries and to return relevant search results in a timely manner.

5. Once the search API has been implemented, the final step will be to integrate the search functionality into Documenter.jl. This will involve modifying the existing search functionality to use the new search API, as well as adding a search UI to the generated HTML pages. The project will require knowledge of web development and Documenter.jl.

The feature will provide users with a faster and more efficient way to search through the Julia documentation, making it easier for users to find the information they need. The project will help to improve the overall user experience of Documenter.jl by making it more responsive and user-friendly.

"I am aware that some of the features I am proposing may already be available or that someone else may be working on them. In such cases, I am willing to adapt and work on other tasks that are needed to improve the overall user experience of Documenter.jl. I am also open to collaborating with others and coordinating efforts to ensure that all improvements are implemented efficiently and effectively."

Timeline

The following is a tentative list of goals I plan to achieve in the mentioned timeframe. There could be situations in which the project may get delayed or lead to an early completion. I plan to handle these cases by having buffer week in the timeline and stretch goals.

Community Bonding Period [May 4 to May 28]

May 4 to May 10

Get accustomed to the coding style and reading documentation in detail.

May 11 to May 15

Set up the development environment and familiarize myself with the Documenter.jl codebase.

May 16 to May 20

Setting up a personal blog.

May 21 to May 28

Discussing the plan with mentors.

Coding Period 1 [May 29 to July 10]

May 29 to June 11

Adding a new feature to convert Jupyter notebooks to Markdown format.

June 12 to June 22

Improving UI/UX of documentation generated by Documenter.jl

June 23 to July 10

Implement line numbering of code snippets and code collapsing

July 11 to July 14

I will do some research on feedback systems to allow users to provide suggestions and report issues directly from the documentation page and submitting midterm evaluations.

Coding Period 2 [July 14 to August 28]

July 14 to July 23

Implement the feedback system in Documenter

July 23 to July 30 [Buffer Week]

If there are any unexpected issues or delays during the project, I plan to utilize this buffer week as a contingency plan. This will allow me to catch up on any missed work or address any unforeseen challenges that may arise. It will provide me with additional time to ensure that I meet the project goals and deliverables.

July 30 to August 10

Adding all possible keyboard shortcuts for common actions such as navigating between pages, searching, or switching between dark or light themes, find something in a page, etc.

August 11 to August 21

Implementing ElasticSearch based Search Backend for Documenter.

August 21 to August 28 [Final Week]

- Finalizing the project and submitting the final evaluation.
- Addressing any pending issues or additional features as stretch goals.

Stretch Goals and Additional Features [September 4 to November 6]

In case any of the above tasks take longer than expected, I plan to work on them during the additional time allotted for stretch goals and additional features.

Deliverables :

The deliverables for this project include:

- Support for generating documentation from Jupyter notebooks.
- UI/UX improvement
- Feedback system for documentation
- All possible Keyboard Shortcuts
- ElasticSearch based search backend
- Detailed documentation and code examples for the added features.
- Contributions to the Julia & Documenter.jl community, including bug fixes and code reviews.

About Me

Contributor: Shravan Goswami

GitHub Handle: shravanngoswamii

LinkedIn: shravangoswami

Email: shravanngoswamii@gmail.com

Country: India

Time Zone: GMT + 5:30 hours

University: Uka Tarsadia University, India

Slack: @Shravan Goswami

Personal Background

I am a first year undergraduate student majoring in Computer Science and Engineering at Uka Tarsadia University in India. I am the Founder of Coder's Club in my college. My primary development environment is Windows 11 Home, and I use VSCode as my text editor of choice. I have a strong foundation in several programming languages, including HTML, CSS, JS, C/C++, and Python. In addition, I am currently learning Julia and Qt development tools, and have been contributing to Documenter.jl for the past few weeks. Outside of my academic pursuits, I enjoy participating in coding contests and exploring new technologies.

Why Me?

As a computer science student, I have experience with several programming languages, including Python and C/C++, and have a strong foundation in competitive programming. My interest in open-source software led me to explore Julia and Documenter.jl in-depth over the past few weeks, where I have gained valuable insights into their codebase. I am a fast learner and easily adapt to new things.

While I am new to contributing to open-source organizations, I am passionate about understanding and improving code written by others. I believe that being able to communicate effectively through code is an essential quality of a great programmer. My skills in programming, attention to detail, and dedication to improving my understanding of complex code make me an ideal candidate for this project. I am excited to work with the Documenter.jl team and contribute to the open-source community.

Logistics

I have scheduled a vacation in May, and my second-semester final exams will take place from June 12 to June 22. However, I have no serious commitments before or after my exams, which allows me to devote more than 30 hours per week to this project. I am also willing to dedicate time during the exam days, as I have already prepared for them. Therefore, the proposed timeline aligns well with my availability and I am confident in my ability to fulfill the project requirements.

Open Source Contributions

I am an active contributor to Documenter.jl, a popular documentation generator for Julia programming language. So far, I have submitted a pull request (#2085) to re-enable auto-switching between light and dark themes. Additionally, I am currently working on addressing two issues, namely, auto theme switching based on default browser settings (#1745) and on-hover footnote preview (#2080). These contributions have helped me to gain a deeper understanding of the Documenter.jl codebase and Julia programming language.

Acknowledgment

- Morten Piibeleht (Slack: @mortenpi)

Thanks for guiding me throughout and clearing my silly doubts.

References

- Julia Documentation
- Julia Slack
- Jupyter Documentation
  * https://jupyter-notebook.readthedocs.io/en/stable/
  * https://docs.jupyter.org/en/latest/

"I am excited about the potential of this project and I am committed to working hard to bring these enhancements to the Documenter.jl community. I look forward to the opportunity to contribute and make a positive impact."

THANK YOU!!