A quarto book for technical documentations

A demonstration how quarto books can be used for documentation and collaboration

Damian Oswald

August 26, 2024

Abstract

This document presents a comprehensive demonstration of the versatile capabilities of Quarto books through a series of practical examples. By leveraging Quarto's robust markdown support, we illustrate how to seamlessly integrate emojis, construct informative tables, and create complex diagrams using Mermaid syntax.

Table of contents

Chapter 1

Introduction

Welcome to my very first Quarto wiki!

1.1 What is this wiki for?

This wiki has been created exclusively for the purpose of thoroughly testing and exploring the various functionalities and features that Quarto's wiki platform offers.

1.2 What can I do in a Quarto Wiki?

A Quarto Wiki is a powerful tool that you can use to enhance your project's documentation and collaboration. Here are some of the key features and actions you can perform in a Quarto Wiki:

1. **Create and Organize Pages:** You can create multiple pages to document different aspects of your project, such as installation guides, API documentation, tutorials, and FAQs. Arrange pages in a hierarchical structure with nested pages, or use a table of contents to provide easy navigation.

2. **Write and Format Content:** Quarto Wikis support Markdown, allowing you to format text with headers, lists, links, images, code blocks, and more. For more advanced formatting, you can also use HTML.

3. **Collaborate with Others:** Multiple collaborators can edit wiki pages to contribute to the documentation. Track changes made to the wiki pages, view revision history, and revert to previous versions if necessary.

4. **Embed Media and Code:** Embed images and videos to enhance the documentation visually. Include code snippets with syntax highlighting for various programming languages.

5. **Link to Other Resources:** Internal Links: Link to other pages within the wiki for better navigation. External Links: Link to external resources such as websites, other repositories, or documentation.

6. **Search and Navigation:** Use the search functionality to find specific content within the wiki quickly. Customize the sidebar and footer to provide links to important pages and

resources.

7. **Access Control:** Public and Private Wikis: Depending on the repository settings, the wiki can be public for anyone to view or private, accessible only to repository collaborators. Permissions: Control who can edit the wiki pages by managing repository permissions.

8. **Git Integration:** Clone and Push: Clone the wiki repository to your local machine, make changes locally, and push updates back to GitHub. This allows for more advanced editing using local tools and version control.

9. **Project Management:** Documentation for Projects: Use the wiki to document the project's development process, including roadmaps, milestones, and task lists.

1.3 Examples of Usage

- **Project Documentation:** Comprehensive guides and references for using and contributing to the project.
- **API Documentation:** Detailed information on API endpoints, parameters, and examples.
- **Tutorials and How-Tos:** Step-by-step instructions for common tasks and workflows.
- **Developer Guides:** Documentation for developers to understand the codebase and contribute effectively.
- **User Manuals:** Instructions for end-users on how to install, configure, and use the software.

By leveraging these features, a GitHub Wiki can significantly enhance the quality and accessibility of your project's documentation, making it easier for contributors and users to understand and engage with your project.

Chapter 2

Some demonstration of quarto books

2.1 First, a little markdown guide

Hello and welcome! Markdown is a *lightweight markup language* with plain-text formatting syntax. It can be converted into HTML and other formats. Here's a quick demonstration of common markdown features.

You can make text **bold** by wrapping it with two asterisks or underscores. Italics are just as easy! Wrap text with one asterisk or underscore: *Italic Text*.

2.1.1 Lists

Creating lists is straightforward. There are unordered lists…

- Unordered list item 1
- Unordered list item 2
  - Subitem 2.1
  - Subitem 2.2

…and then there are ordered lists:

1. A first item
2. A second item
3. And a last item

2.1.2 Headers

Headers from H1 to H6 are essential for structure. They're made with #:

```
# H1 Header
## H2 Header
### H3 Header
```

2.1.3 Links and Images

Adding a link is as simple as wrapping text in brackets followed by the URL in parentheses. To add an image, it's very similar but starts with an exclamation:

Figure 2.1: This is an image of some agricultural activity in coorporate design.

2.1.4 Quotes and Code

Quotes are also a default part of the markdown syntax.

> This is a blockquote. Use it to highlight important sections.

And so is code. For inline code, use single backticks: `Inline code here` For longer code, use triple backticks:

```
def hello_world():
    print("Hello, world!")
```

2.1.5 Emojis

To insert emojis, simply type `:heart:`. Use whatever name the emoji has and it will be rendered correspondingly Here, I want to write something else. So that I am !

2.2 Tables

Here's an example of a markdown table using pipe syntax, representing a list of programming languages and their respective release years:

Table 2.1: We can add a table caption. And a reference.

Programming Language	Release Year	Creator
Python	1991	Guido van Rossum
JavaScript	1995	Brendan Eich
Java	1995	James Gosling
C++	1985	Bjarne Stroustrup
Ruby	1995	Yukihiro Matsumoto
Swift	2014	Apple Inc.
Go	2009	Robert Griesemer et al

Feel free to use or modify Table 2.1 as needed!

2.3 Mermaid diagrams

This diagram visualizes the fundamental structure of the product catalog without the junction tables, i.e. containing many-to-many relationships.

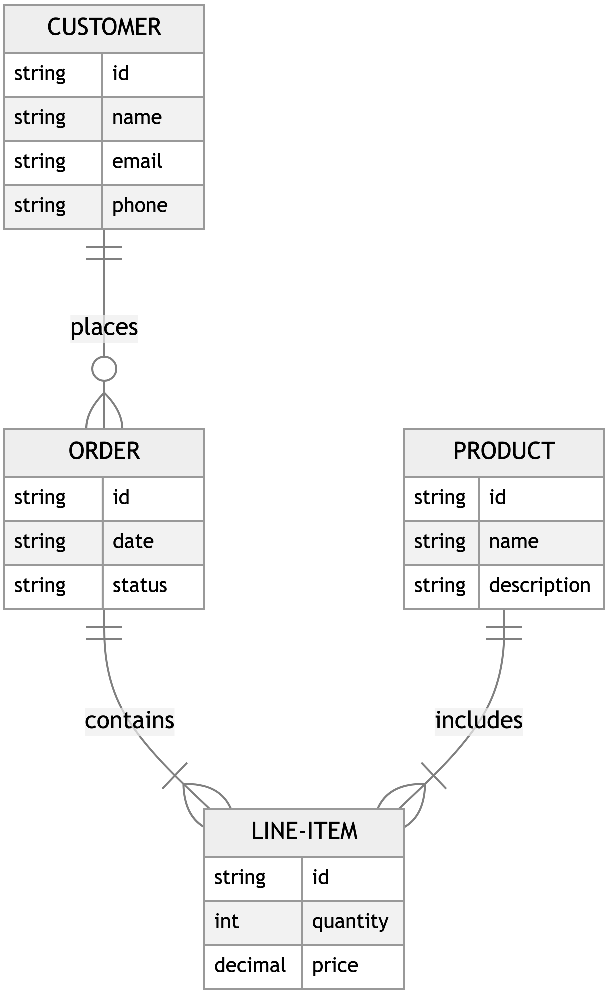

Figure 2.2: Example of a simple entity relationship diagram using Mermaid JS.

Here's a simple example of a sequence diagram using Mermaid JS. This diagram will illustrate a sequence of interactions between two actors, `System A` and `System B`, with a message exchange:

In this sequence diagram: - `SystemA` and `SystemB` are the participants (systems) involved in the sequence of interactions. - `SystemA ->> SystemB: Message` denotes a message sent from `SystemA` to `SystemB`. - `SystemB -->> SystemA: Response` denotes a response message sent from `SystemB` back to `SystemA`.

This example shows a simple sequence where `SystemA` sends two messages (`Message 1` and `Message 2`) to `SystemB`, and `SystemB` responds with `Response 1` and `Response 2` respectively.

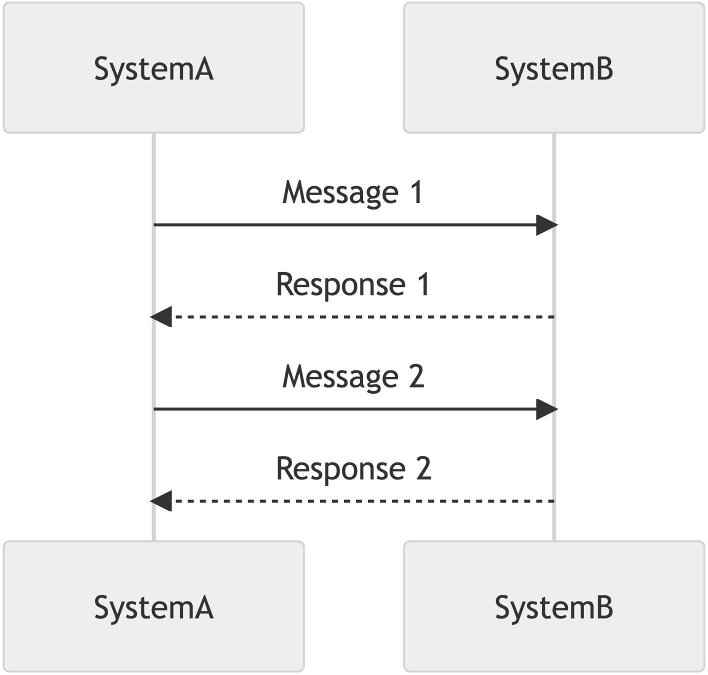

Figure 2.3: Example of a simple sequence diagram using Mermaid JS.

2.4 Code files

Below is a simple demo Python code that demonstrates a basic program to calculate the factorial of a number using both iterative and recursive methods:

Here's an explanation for the code above.

- **Iterative Method (`factorial_iterative`)**:
  - Initializes `result` to 1.
  - Loops from 1 to `n`, multiplying `result` by the loop counter `i` in each iteration.
  - Returns the final `result`.
- **Recursive Method (`factorial_recursive`)**:
  - If `n` is 0, returns 1 (base case).
  - Otherwise, returns `n` multiplied by the factorial of `n-1`.
- **Main Program**:
  - Defines a variable `number` to hold the value for which the factorial is to be calculated.
  - Calls the iterative and recursive factorial functions and prints the results.

You can run this code in any Python environment to see the output for the factorial of 5 using both methods.

Listing 2.1 `factorial.py`

```python
def factorial_iterative(n):
    """Calculate factorial of a number iteratively."""
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

def factorial_recursive(n):
    """Calculate factorial of a number recursively."""
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Input: Number for which factorial is to be calculated
number = 5

# Calculate factorial using iterative method
iterative_result = factorial_iterative(number)
print(f"Factorial of {number} (iterative): {iterative_result}")

# Calculate factorial using recursive method
recursive_result = factorial_recursive(number)
print(f"Factorial of {number} (recursive): {recursive_result}")
```

Chapter 3

A last page

Aaand that's it. This is the last page of this wiki.