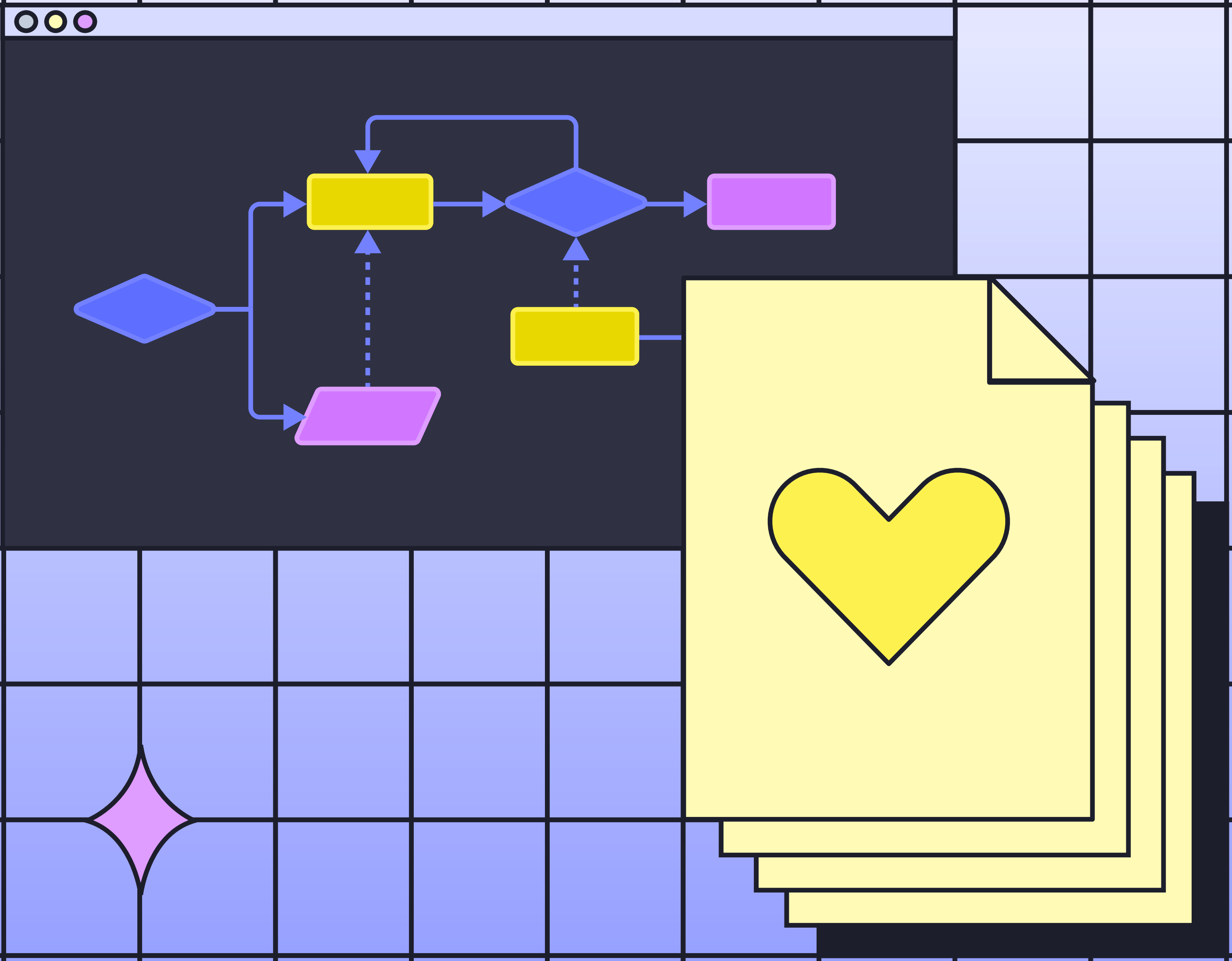
swim

10 Code documentation templates



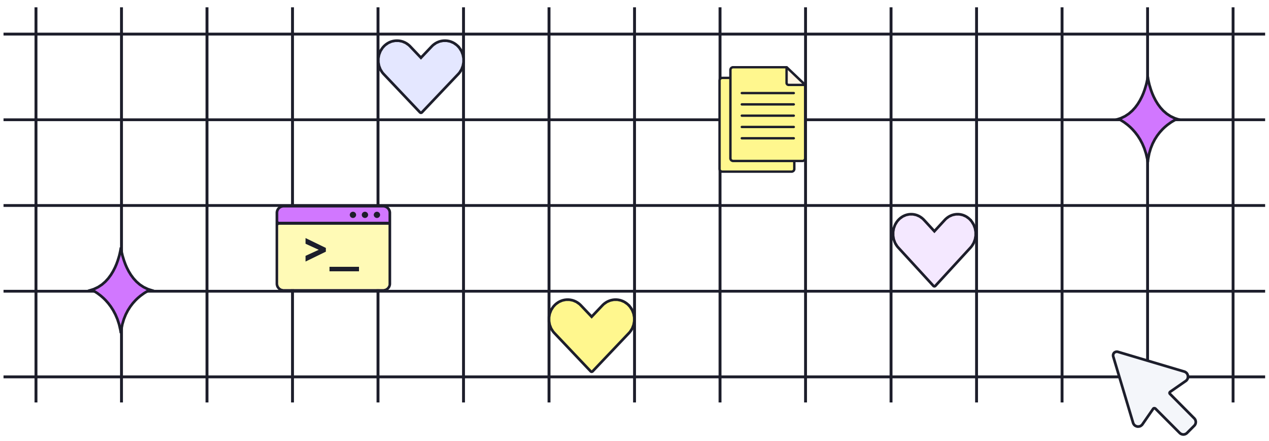
Developers, and development teams deserve and need good documentation. When such documentation exists – it makes the onboarding process easier and faster, it helps teams be more agile and transfer knowledge when needed, and it helps you in those cases where you need someone to understand something in the moment – for example, when you have a bug affecting production and you must find a solution.

While high-quality and up-to-date documentation is an essential component of software development, developers today lack efficient code documentation and relevant tools.

To help you get there faster, Swimm created a new bank of templates to inspire you with suggestions about what exactly to write to make you better documentarians. This way, you can overcome "writer's block" and gain the confidence to know that creating effective documentation does take just a few minutes.

The following templates include a mix of instructions for those documenting, in addition to the elements the documents actually require. You can use each template as is, on any documentation tool you're using, or you can also sign up to Swimm, create a new document, and select a template from Swimm's internal template library.

Click to access all templates on GitHub



Template 1:
A flow in the code

When should you use it?

A "flow in the code" documentation template is valuable for documenting complex algorithms, critical business processes, and integration points. Complex code can be hard to understand, even by the person who wrote it. Creating documentation that explains complex code saves you and your team a lot of time when revisiting the code in the future.

Introduction

This doc describes the {{SUBJECT}} flow in our system. We will follow its implementation across the various locations so you can understand how the different parts create the full picture.

Following the flow

Include the code snippet showing the first step where the flow begins - e.g., the entry point, or an API call. From there, keep {{adding snippets}} from the next steps of the flow.

Things to note

Actually, selecting the relevant code is most of the work. Now all you have to do is explain how the different parts interact, and what's important to know and is not clear from reading the code.

Describe who uses this flow and when? Context is everything, and if documentation doesn't spell out the relevant parties, it may not be clear.

Template 2:

Architecture decision records

When should you use it?

Architecture Decision Records **(ADRs)** serve as a valuable tool in the software development process, offering a structured approach to documenting decisions made during the project's evolution. You should consider using ADRs in the event of change management, historical record maintenance, and decision making transparency.

Status

There are many possible statuses. A decision may be "proposed" if the project stakeholders haven't agreed with it yet, or "accepted" once it is agreed. If stakeholders change or reverse a decision, it may be marked as "deprecated" or "superseded", with a reference to its replacement

Context

This section describes the forces at play concerning the decisions. Who, what, when, where, why, and how regarding the project should be answered in this section. The language in this section is value-neutral. It is simply describing facts.

Consequences

This section describes our response to these forces. It is stated in full sentences, with active voice. For instance, it may begin with “We will...”

Decision

This section describes the resulting context, following the decision’s application. All consequences and outcomes should be listed here, not just the "positive" ones. Just one decision may have positive, negative, and neutral consequences, all of which may affect the team and project in the future, if not immediately.

Template 3:
Component or service overview

When should you use it?

A "Component or Service Overview" document is a valuable resource when you need to understand the key aspects of a component. It should be used:

- To gain a clear understanding of the component's main features and how it can be accessed.
- To familiarize yourself with the directory structure of the component or service.
- To comprehend the key design decisions that have shaped this component.
- To become acquainted with essential terms and concepts relevant to the component.

Main Features

This section outlines the key features that define {{COMPONENT NAME}} . These features encompass its interface, directory structure, design decisions, and important terms within a glossary.

Interface
To begin, ask yourself “How can this component be accessed?”. Understanding its interface is essential for effectively integrating it into your project.

Directory Structure
Take a close look at the directory structure of {{COMPONENT NAME}} . Mention the main folders within the component. By doing so, you'll gain valuable insights into its layout and architecture.

Design Decisions
Explain the key design decisions that influenced the development of {{COMPONENT NAME}} . This section will provide a deeper understanding of the thought processes and choices that have shaped its functionality and structure.

Glossary
Present a glossary of essential terms and concepts relevant to {{COMPONENT NAME}} . This glossary serves as a reference to ensure that all team members and stakeholders share a common language and a comprehensive understanding of the component. Trying getting started with “Here are some important terms to know:”

Template 4:

Dev environment setup

When should you use it?

A "Dev Environment Setup" guide is essential when you need to configure your local development environment. Use this document to ensure a smooth setup and efficient development process.

Installations:

For example:

- Begin by installing Node.js, with version 14.x as the recommended choice.
- Secure an Integrated Development Environment (IDE), such as Visual Studio Code (VS Code).
- Make sure Git is installed on your machine. If it's not, install it.

Getting the Sources:

- Clone the repository locally with the following command:
- git clone https://github.com/my_company/company_repo.git

Build:

Within the repository directory, follow these steps:

1. Execute `yarn install` to install the project's dependencies.
2. Build the project by running `yarn build`.

Troubleshooting:

If you encounter an error like 'Cannot execute command (...) - "need executable 'ar' to convert dir to deb," you might need to install the 'binutils' package.
Use this command to ensure it's available:

```
sudo apt-get install binutils
```

Run the Tests:

- To run all tests, use the command: `yarn test.`
- For specific subsets of tests, utilize commands like `yarn test:server` or `yarn test:utils.`

Running the Application:

- macOS and Linux users should execute `./scripts/run.sh.`
- Windows users can run `.\scripts\run.bat.`
- For web development, use `yarn web.`

Useful Scripts:

- Serve your code with a development web server: `yarn dev.`
- To package the application for production and generate installers, use: `yarn pack.`
- Explore the `package.json` file for a full list of supported yarn scripts.

Debugging:

For debugging purposes, open developer tools by pressing *Command+Option+I* (Mac) or *Control+Shift+I* (Windows, Linux) to access the Console panel. You can also utilize breakpoints for efficient debugging.

Congratulations! Your development environment is now configured and ready for action. 🎉

Template 5:
Engineering design

When should you use it?

An "Engineering Design" document plays a crucial role when you need to establish a comprehensive technical plan and overview for a feature or system. This document finds utility in various scenarios. For instance, it can include details such as:

References:

Incorporate relevant documents that provide context or additional insights. Additionally, provide a link to the Product Requirements Document (PRD) for a deeper understanding of the feature's requirements and objectives.

Goals:

Define the primary objectives and goals for the feature, outlining the specific Key Performance Indicators (KPIs) that will gauge its success.

High-Level Design:

Visualize the high-level design with tools like flowcharts to elucidate the system's architecture and logic. Furthermore, include references to:

- DB Changes: Provide a detailed account of any modifications or updates to the database, covering schema changes and data model adjustments.
- UI Components: Clarify the user interface components involved in the feature
- Storage: describe where information is stored, such as in the state or local storage.

Third-Party Integrations:

Outline the integrations with external services, APIs, or platforms. Describe the data flow and interactions with third-party entities, as well as:

- Logs: Discuss the mechanisms in place for logging events and errors within the feature.
- Analytics: Specify the analytics tools or methods employed to collect performance data for the feature.

Additional Elements to Consider:

Highlight any other pertinent aspects or elements not covered in the preceding sections that bear relevance to the engineering design, including:

- A plan for introducing tests.
- Strategies for migration where applicable.
- Address security implications and the measures in place.
- Develop a roll-out plan to effectively implement the feature.

Template 6:
Incident report

When should you use it?

An "Incident Report" is invaluable when you need to analyze and document incidents to understand what happened and identify areas for improvement. Use this report when investigating and addressing incidents or problems within your system.

What happened?

Provide a brief and concise description of the incident, summarizing the key details of what occurred during this particular event.

Who was involved?

Identify the individuals or teams engaged in debugging, investigating, and resolving the issue. You may opt to keep the involvement anonymous or specify the engineers or teams from different departments who played a role in incident management.

Who or what was impacted?

Explain the extent of the impact caused by the incident, and if possible, quantify the damage. Specify the tasks, functionalities, or user experiences affected during the incident.

What was the root cause?

Delve into the root cause of the incident, describing the sequence of events leading to the incident. Work backward as far as possible, from the inception of the issue to the immediate events preceding the incident.

How was it reported?

Detail how the issue was discovered, who reported it, and the process of reporting. Include information about the individuals or systems involved in incident reporting.

When did it happen?

Establish a comprehensive timeline of the incident, documenting key timeframes, including the start and end of service degradation, the first alert raised, and significant milestones associated with the incident. Use a format like this:

- Service degradation start: `20YY-MM-DD HH:MM:SS UTC`
- Service degradation ended: `20YY-MM-DD HH:MM:SS UTC`
- First alert was raised: `20YY-MM-DD HH:MM:SS UTC`
- Timeline (time in UTC+2):
    - `20YY-MM-DD HH:MM:SS UTC` - offending code was deployed
    - `20YY-MM-DD HH:MM:SS UTC` - ..

Template 7:
Internal API

When should you use it?

The "Internal API" type of document is a valuable resource when you need to understand and work with specific APIs internally within your system. Utilize this document when you are dealing with and implementing internal APIs for various use cases.

How does it work?

Provide a detailed description of the {{API Name (e.g., sending Analytic Events)}} API and how to correctly utilize it. This API becomes essential in scenarios where {{use cases}} are relevant.

API definition

Present code snippets of the various function definitions that compose the API. This will give the reader an understanding of where and how the API is implemented within the system.

Simple usage

Demonstrate a simple example of how to use this API, providing a straightforward illustration of its practical application.

Advanced usage: {{explain a scenario where this is needed}}

Delve into the root cause of the incident, describing the sequence of events leading to the incident. Work backward as far as possible, from the inception of the issue to the immediate events preceding the incident.

Best practices and additional notes

When working with this API, it is crucial to adhere to specific best practices and avoid common mistakes. This section provides guidance on optimal API usage and offers additional insights for effective implementation.

Template 8:

Product Requirements Document (PRD)

When should you use it?

The "Product Requirements Document (PRD)" is an essential resource used to outline and communicate the requirements and objectives for a new feature or product. This document plays a pivotal role in defining the specifications and development plans for a product or feature.

Problem Statement

In this section, define the specific user pain point that the feature aims to address, as well as the motivation behind implementing this feature.

Requirements

The user flow when engaging with this feature is elaborated in this segment, offering insights into the feature's functionality. Also address how edge cases will be handled and any unique considerations that should be taken into account during the feature's development.

UX

In this section, be sure to link to the relevant Figma file, allowing stakeholders to access and review the user interface and experience design for the feature.

Rollout Plan

Here, explain the strategy for introducing the feature to customers. Specify whether the release will be gradual or if it will be part of an A/B test, ensuring a clear roadmap for deployment.

Analytics

This section outlines the approach to tracking the feature's performance using the product analytics tool. It defines the events that should be triggered from the app and those that should be sent to Salesforce for further analysis.

KPIs

Identify the success criteria for the feature. It includes specific metrics, outcomes, or data points that will serve as indicators of the feature's success.

Template 9:
Research plan

When should you use it?

The "Research Plan" is an invaluable document that outlines the strategy and steps for conducting research, particularly in cases where extensive knowledge and data gathering is needed. Use this document when preparing for research initiatives.

Background

In this section, provide comprehensive context on the current state of affairs. Cover all the essential information needed to grasp the research goals and objectives fully.

Goal

This segment outlines the specific issues you intend to solve and the overarching goal of our research.

Issues, Considerations, and Constraints

In this part, document any constraints and considerations that may affect your research, such as security limitations or performance requirements. It serves as a reminder of the parameters within which the research must be conducted.

Past Directions and Relevant Knowledge

Delve into previous research efforts and any relevant knowledge that could provide insights or solutions to the problem. This section discusses what has already been tried and the knowledge that can be relied upon.

Prerequisite Reading / Information Finding

To ensure a thorough understanding of the problem at hand, outline the general information you should gather before embarking on the research.

Possible Directions

For each potential research direction, detail your plans, outlining what actions you intend to take and provide estimates of the time required for each step. This structured approach ensures that research efforts are well-documented and organized.

Template 10:
Testing overview

When should you use it?

The "Testing Overview" is a crucial document employed to outline the testing strategy and details for various types of tests. This document finds utility when preparing for testing initiatives and ensuring a clear understanding of the testing framework and processes.

Testing Frameworks

In this section, specify the testing frameworks we employ, such as {{UNITTEST FRAMEWORK (e.g., jest)}} for unit tests and {{E2E FRAMEWORK (e.g., playwright)}} for end-to-end tests. It provides an overview of the tools and technologies used in our testing process.

Running Tests Locally

Here, provide the specific commands for running tests, such as using yarn test. This simplifies the testing process and helps team members quickly execute tests as needed.

Run the Tests

Here, provide the specific commands for running tests, such as using `yarn test`. This simplifies the testing process and helps team members quickly execute tests as needed.

Writing Tests

In the "Assertions" section, add examples of common assertions used in tests, explaining how they work. For instance, we illustrate how certain assertions are used in tests to improve comprehension.

Best Practices

In this part, outline best practices to follow when writing tests. Emphasize the structure of test suites and provide examples, such as how {{path for a test file}} tests the {{feature's name}} feature. Additionally, share specific best practices to enhance the quality and effectiveness of our testing efforts.

Templates

- **Template 1:** A flow in the code
- **Template 2:** Architecture decision records
- **Template 3:** Component or service overview
- **Template 4:** Dev environment setup
- **Template 5:** Engineering design
- **Template 6:** Incident report
- **Template 7:** Internal API
- **Template 8:** Product Requirements Document (PRD)
- **Template 9:** Research plan
- **Template 10:** Testing overview

Sign up for a **community demo today** or

Get a personalized demo

swimm