Aalto University
School of Science
Master's Programme in Security and Cloud Computing

Nafis Kamal

API Documentation Generator

Master's Thesis
Espoo, August 28, 2022

Supervisors:	Fabian Fagerholm, Aalto University
	Raphaël Troncy, EURECOM
Advisor:	N/A

Aalto University School of Science

Aalto University
School of Science
Master's Programme in Security and Cloud Computing

ABSTRACT OF MASTER'S THESIS

Author: Nafis Kamal

Title:
API Documentation Generator

Date:	August 28, 2022	**Pages:**	56
Major:	Security and Cloud Computing	**Code:**	SCI3113
Supervisors:	Fabian Fagerholm, Aalto University Raphaël Troncy, EURECOM		
Advisor:	N/A		

The importance of Application Programming Interfaces (APIs) in contemporary software development processes is growing. It can be challenging for developers to rapidly comprehend how to utilize a new API; therefore, good documentation is required.

For efficient documentation support, we must understand how developers utilize widely available tools today. We provide the results of an exploratory study that examined the pros and cons of observing programmers as they used a basic application programming interface to find solutions. By utilizing an existing API documentation, you can save time and money by not having to reinvent the wheel when integrating with third-party enterprise systems and devices.

This thesis describes and evaluates a unique technique to meeting API documentation requirements. I present a list of standards for the documentation of a selection of API tools based on my analysis of the existing literature and standard industry practice. I compare and contrast the documentation processes of Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST with my own prototype implementation, which includes sample code for interacting with the API. I did a randomized study to establish the optimal method for determining the significance of API documentation requirements and to identify a strategy for simplifying documentation, with a focus on fulfilling the needs of user developers.

Using Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST, I found reoccurring difficulties that may be minimized with the suggested documentation.

Keywords: API, Documentation, Tools, Postman, Redocly, SwaggerHub, JavaDoc, AutoREST

Language: English

EURECOM

Master's Programme in Digital Security

ABSTRACT OF MASTER'S THESIS

Author:	Nafis Kamal		
Title: Générateur de documentation API			
Date:	August 28, 2022	**Pages:**	56
Major:	Digital Security	**Code:**	SCI3113
Supervisors:	Fabian Fagerholm, Aalto University Raphaël Troncy, EURECOM		
Advisor:	N/A		

L'importance des interfaces de programmation d'applications (API) dans les processus de développement de logiciels contemporains ne cesse de croître. Il peut être difficile pour les développeurs de comprendre rapidement comment utiliser une nouvelle API ; par conséquent, une bonne documentation est requise.

Pour une prise en charge efficace de la documentation, nous devons comprendre comment les développeurs utilisent aujourd'hui des outils largement disponibles. Nous fournissons les résultats d'une étude exploratoire qui a examiné les avantages et les inconvénients d'observer les programmeurs lorsqu'ils utilisaient une interface de programmation d'application de base pour trouver des solutions. En utilisant une documentation d'API existante, vous pouvez gagner du temps et de l'argent en n'ayant pas à réinventer la roue lors de l'intégration avec des systèmes et appareils d'entreprise tiers.

Cette thèse décrit et évalue une technique unique pour répondre aux exigences de documentation de l'API. Je présente une liste de normes pour la documentation d'une sélection d'outils API basée sur mon analyse de la littérature existante et des pratiques standard de l'industrie. Je compare et oppose les processus de documentation de Postman, Redocly, SwaggerHub, JavaDoc et AutoREST avec ma propre implémentation de prototype, qui inclut un exemple de code pour interagir avec l'API. J'ai fait une étude randomisée pour établir la méthode optimale pour déterminer l'importance des exigences de documentation de l'API et pour identifier une stratégie de simplification de la documentation, en mettant l'accent sur la satisfaction des besoins des développeurs utilisateurs.

En utilisant Postman, Redocly, SwaggerHub, JavaDoc et AutoREST, j'ai trouvé des difficultés récurrentes qui peuvent être minimisées avec la documentation suggérée.

Keywords:	API, Documentation, Tools, Postman, Redocly, SwaggerHub, JavaDoc, AutoREST
Language:	Anglais

Acknowledgements

I want to thank my main supervisor Fabian Fagerholm from Aalto University and my supervisor from EURECOM Raphaël Troncy for their good guidance throughout the whole thesis.

I would also like to thank European Union who funds the Erasmus Mundus scholarship for giving me the opportunity to pursue my master degree.

Espoo, 28.8.2022

Nafis Kamal

With the support of the Erasmus+ Programme of the European Union

Contents

Abbreviations and Acronyms

API	Application Programming Interfaces
ASCII	American Standard Code for Information Interchange
CLI	command-line interface
CSG	Constructive solid geometry
CVS	Concurrent Versions System
DOC	Documentation, Document
ESG	Enterprise Strategy Group
GTK	GIMP Toolkit
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated development environment
JSON	JavaScript Object Notation
MSDN	MicroSoft Developer Network
PCI	Peripheral Component Interconnect
REST	Representational state transfer
SaaS	Software as a service
SDK	Software development kit
SOAP	Simple Object Access Protocol
SVN	Subversion
UI	User Interface
UIB	User Interface Block
URL	Uniform Resource Locator
VS	Visual Studio
XML	Extensible Markup Language
YAML	Ain’t Markup Language

Chapter 1

1 Introduction

The process of telling a computer what to do in a way that allows the program to be re-executed at a later time is known as programming. Programming is normally accomplished via the utilization of one or more textual programming languages. The process of programming might be challenging at times. There are a lot of different factors at play here, and it is possible that some of these challenges are inherent to the activity. However, additional challenges can be attributed to certain aspects of the programming language and tools that are utilized by programmers, and it is likely that these challenges could be considerably mitigated if the appropriate measures were taken. The use of application programming interfaces (APIs) is one significant source of programming complexity on which we could have a significant impact.

Application Programming Interfaces (APIs) allow software developers to routinely make use of the data and services offered by a variety of different programs. This practice is commonplace. Many tasks need engineers to "weave together" functionality that is already provided by APIs rather than creating functionality from scratch. This is because APIs already exist [1]. The significance of application programming interfaces (APIs) has grown substantially over the past several years due to the fact that corporations and other types of organizations increasingly rely on the internet to publish their data and provide their services. Every software developer is faced with the common challenge of learning the capabilities of an API, the elements it supplies, and how to mix these components together to bring about the necessary functionality. This is a process that must be completed.

APIs are often published together with API references, tutorials, sample projects, and other materials that are aimed to make the work of learning easier. However, getting started with a new API can be difficult at times, and a lack of suitable learning materials in the form of API documentation has been cited as a primary issue that contributes to this difficulty. The results of an empirical study that analyzed how an overview of API documentation is presented to developers are contributed by this research. My work is motivated by the premise that issues with API documentation may in part reflect usability issues, and in particular by the idea that the content and structure of documentation may not always fit the expectations and work habits of developers. Therefore, for API documentation to serve as a effective aid in learning an API, we need to realize which general methods software developers use when solving programming tasks, which information they require, and which information resources they turn to. In addition, we need to know which information resources they turn to [2].

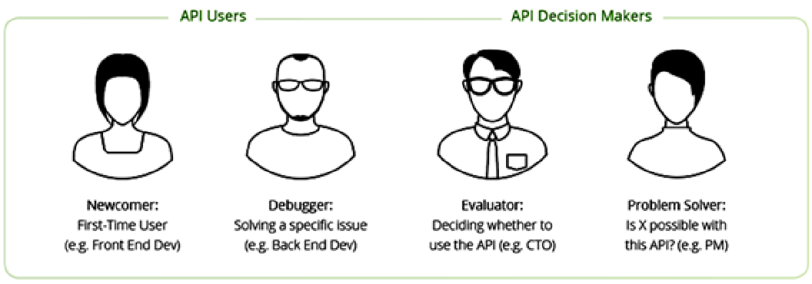


Figure 1.1 : API users to API documents decision makers (source: Zhong Zhendong [2])

In order to investigate these concerns, I carried out research utilizing the observational approach. We presented our developers with a series of programming challenges using an application programming interface (API) that they were not familiar with. This was more like a casual chat. I talked about the ideas with my current and former developer colleagues who live in Asia and Europe and still work with APIs. I asked them how they choose new APIs by reading the documentation and how they deal with difficult problems to learn more about the best practices. Several people who went to the chat gave me their thoughts on it. After that, i studied the techniques that they used to complete the jobs, the sections of the API documentation that they utilized, and the design characteristics of the API documentation that caused issues [3]. On the basis of our research, we offer a number of design recommendations that, if followed, will result in API documentation that is more efficient.

1.1 Motivation

Application Programming Interfaces (API) have been a topic of discussion in both the scientific and business communities for many decades. Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST are the projects that are credited with initiating the development of modern online APIs. The principles were formed by the contribution of selected API tools, and ever since then, an abundance of web APIs have been produced over the course of the last few decades. APIs and web APIs are used interchangeably in today's world. In point of fact, the past ten years have demonstrated that, first, the quantity of APIs and, second, the use of APIs have both experienced meteoric rises. In spite of the fact that these data numbers can only be seen for publicly accessible APIs, industry experts believe that the number of privately held APIs is significantly higher than that of publicly available APIs.

Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST are able to reach new markets, support their business plan, and promote the creation of new inventive solutions because they have made their APIs accessible to the external and internal advantages of customers. Companies have to pay close focus on the quality of their API documentation if they want to maintain a healthy profit margin from the API economy. The ability to traverse the digital age successfully requires API documentation as a core competency. The API Management constitutes a portion of the API

Documentation. However, this knowledge of how to build APIs that are more usable has very little impact on the numerous APIs that are currently in widespread use and have been released. In addition, designers have to take into account essential factors other than usability, such as performance and future extension, which might result in the design of APIs that are more difficult to use for perfectly valid reasons. The creation of wrapper APIs, modifications to the integrated development environment, and revisions to the API documentation are some of the several methods that can be utilized in order to enhance the usefulness of current APIs that have been built in existing programming languages. Previous research indicated that a large number of Java developers heavily rely on documentation based on Redocly, SwaggerHub, JavaDoc, and AutoREST. As a result of these findings, I have been investigating the various ways in which API documentation can be utilized to enhance the functionality of already existing APIs.

1.2 Problem Statement

Developers are increasingly reliant on API documentation. One cause for this is the growing complexity of the programs themselves, both in terms of breadth and the number of moving parts. As an example, word processors have progressed from being basic text editors to sophisticated document production environments with dozens of options. When the mouse is hovered over a button or menu, it will now have a gradient of colors, a drop shadow, and either glow or pulse. Building these features from scratch would require hundreds or thousands of lines of code, but they are generally already contained in standard widgets offered by an API. As a result, developers are becoming more reliant on APIs, as it is becoming more challenging to build their own buttons, for example, while still giving the look and polish that customers have grown to expect.

APIs are not merely a useful tool for facilitating reuse in programming; they are often essential. Some features are only accessible via APIs, and cannot be reproduced without breaching the API's security or encapsulation. Most modern operating systems, for instance, prohibit direct programming access to the graphics hardware, mandating instead that developers use one of several APIs. The code that developers produce reveals their heavy reliance on APIs. A Microsoft framework developer made anecdotal comments on how API calls have replaced more fundamental programming language structures like if statements, for loops, and variable declarations.

This shift toward using APIs has altered not only the coding process but also the understanding, debugging, and maintenance of code, which in turn impacts not only the programmer's efficiency but also the quality of the finished software. Finding defects that arise from APIs' hidden requirements for example, that a method be called only from a specific thread or that a specific return value cannot be cached can be challenging since such requirements are not obvious from looking at the code that invokes the APIs. As a result of the large amounts of functionality they encapsulate, APIs can restrict the flexibility with which programs can be modified, sometimes leaving developers with the option of restricting a feature to the capabilities provided by an API or foregoing the API altogether and re-implementing the features from scratch.

1.3 Research Question

In the following section, a series of research questions is offered. These research questions are formed from the problem statement that was discussed in the preceding section, and they serve as a compass for this thesis all through the entirety of the research.

- Why do we need to write documentation?
- Why waste time on that instead of coding it?
  - Improves the experience for developers
  - Decreases the amount of time spent on-boarding new users
  - Leads to good product maintenance and quicker updates
  - Agreement on API specs
  - The API documentation can act as the central reference
  - Unblocks development on different sides
  - Allows identifying bugs and issues in the API's architecture
  - Decreases the amount of time spent on understanding how the API works

1.4 Contribution

To summarize, this thesis provides several important contributions to the existing body of study in the field of software engineering, which are as follows:

1. Enhances the experience of API documentation requirements for developers, generated from the existing body of literature and the current state of industry practices, to be utilized by practitioners as a guide.
2. Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST are examples of unique techniques and reusable tools that employ interception to automatically generate and maintain developing API documentation. This results in a reduction in the amount of time that is spent on-boarding new users.
3. This results in good product maintenance and speedier updates to illustrate the practicality of the proposed technique that API developers can follow to improve their experiences with Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST API documentation.

4. The API documentation has the potential to serve as the central reference evidence, demonstrating the typical challenges encountered by API client developers such as Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST. These challenges can be mitigated with the help of usage examples, which will assist in the prioritization of API documentation efforts.

5. For the selected tools, the thesis documented the documentation commands, pros and cons, user experience, limitations.

1.5 Structure of the thesis

The rest of the thesis is organized as follows. Chapter 2 overviews the background of API documentation. Chapter 3 discusses the methods and materials used in this study. Chapter 4 details the results, pros-cons, limitations of API documentation. Chapter 5 details the guidelines and problem study. Finally, Chapter 6 concludes the thesis.

Chapter 2

2 Background

Programming is the process of instructing a computer to perform specific tasks using one or more textual programming languages. Programming is difficult. Some of these difficulties may be inherent to the undertaking. By altering the programming language and tools used by programmers, other problems may be mitigated. APIs are a source of programmability challenges that can be mitigated.

2.1 Concept, Definition and Terms of API Documentation

API stands for Application Programming Interface and, at a high level, defines a framework for reusing code. Instead of beginning from scratch with every program, code reuse enables programmers to build upon the work that other programmers (or themselves) have already completed [4]. Copying and pasting code straight into one's application, often known as the copy and paste style of reuse, and invoking pre-packaged functions that are intended to be reused are the two primary methods of code reuse. In this thesis, these functions are referred to as libraries, frameworks, or APIs. With APIs, programmers are able to reuse code without needing to edit, comprehend, or even view the implementation; instead, they engage solely with the programming interface.

Table 1: Different API-related words with corresponding examples. In this study, I collectively refer to these as "API"

Libraries	Math library, "standard" library in C
Frameworks	.NET Framework, Eclipse Framework
`Software Development Kits`	.NET Development Kit, Java Development Kit
`APIs`	Win32 APIs, Google Map APIs
`Toolkits`	The GIMP Toolkit (GTK), Google Web Toolkit

Source: Robillard et al. [5]

By utilizing APIs, developers may design highly complex applications with minimal effort. This is in part due to the fact that APIs can grant access to a plethora of features at a high level of abstraction. A web-browser widget, for instance, allows you to embed a full web browser with a single line of code, as opposed to the tens or hundreds of thousands of lines of code required to build the same functionality from scratch. But this strength is not without cost [6]. APIs frequently place restrictions on what developers may do and how they must utilize the APIs in order to produce a successful software. However, it seems that certain APIs are far more difficult to use than others, even when the two APIs provide the same functionality at the same abstraction level. This gives us optimism that we can simplify the process of programming by figuring out how to make APIs more approachable.

2.1.1 API Documentation

The documentation of APIs has been the subject of numerous research, all of which offer best practices that may be applied to API documentation as well. Inadequate documentation and other learning aids were identified to be a major barrier for developers while learning new APIs. When it comes to API documentation, authors stressed the need of providing clear examples, covering all bases, accommodating a wide variety of use cases, staying on top of organization, and incorporating pertinent design aspects [5]. The relevance of providing examples in API documentation was explored. In order for a user to really use an API, Endrikat suggested converting the API example scenarios into executable test cases [7].

To determine what features are essential in high-quality code examples, Nasehi et al. conducted a case study based on conversations in JavaDoc. They suggested using wiki-like collaboration tools with online API documentation and recommending that API developers provide examples in the documentation. According to research by Maalej and Martin [8] users of API documentation can interact with API creators via social media. For the sake of user convenience, Inzunza et al.[9] suggested including crowd sourced frequently asked questions in API documentation. Subramanian et al. demonstrated an automated method to connect SwaggerHub's code samples with the API documentation of several Java and JavaScript libraries.

Software as a service (SaaS) based solutions for API documentation systems have a number of benefits, as outlined by Watson et al.[10]. These include being affordable while still being powerful, being platform independent and highly accessible, having higher document quality, reusing content, having access to automated tools, and having a well-organized, scalable documentation process. Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST are just few of the technologies available that aid in the automatic development of API documentation for local APIs. Despite their success in describing local library APIs, these tools have limited utility when it comes to documenting APIs as a whole due to their inability to handle HTTP-specific information out of the box.

2.1.2 Classification of API User

It is widely understood that APIs must be suitable for the developers who will be using them. An API that is useful to one group of developers may not be to another. This begs the question, though, of how API designers may best categorize and arrange programmers into distinct communities that correlate appropriately to the various API needs [11].

One method involves sorting programmers according to their level of expertise and the programming languages and tools they are familiar with. A creator of an API could take this approach by creating two distinct sets of APIs, one for beginners and the other for seasoned pros. Two different sets of application programming interfaces (APIs), say, one for Visual Basic developers and the other for C++ developers [12].

According to a related system, "professional" programmers are often software engineers whose major work function is to code and who frequently have formal programming education. End-user programmers, on the other hand, are those whose primary job is not programming but rather, say, physics or administrative support [13].

Finally, "personas" for programmers are another strategy. Individuals who represent typical consumers are called "personas," and they are frequently utilized in design [14]. simpler and clearer to work with while designing. Microsoft employs three distinct programming personas, each of which was based on observations of hundreds of people using Visual Studio. All popular approaches to programming are attempted to be represented in these personas. Every programmer has the ability to fall into any of the personas, which are typically determined by how they approach various programming tasks. This is because, while the personas do correlate generally with different skill levels and job types, they do not connect exactly. These characters are meant to represent a variety of work styles rather than actual skills or expertise. This helped us target our experiments and select the right participants. Our findings can be extrapolated to other developers of a similar personality type because we based them on a large sample size.

2.1.3 Quality Attributes

There are many different qualities that are desirable for APIs, though previous research had not attempted to enumerate them all. We create a set of quality attributes here, forming a hierarchy of attributes. Figure 2.1 includes a summary of these attributes and the stakeholders most affected by each.

The two most fundamental characteristics of an API are its accessibility and its strength. The "documentation usability" of an API refers to its attributes that affect its use while developing and debugging code, whereas the "power" of an API describes the boundaries of the code that may be written [15].

In terms of documentation usability should focus on how well an API matches users' mental models, how simple it is to use, how consistent it is, how quick it is to learn, and how productive programmers are while using it.

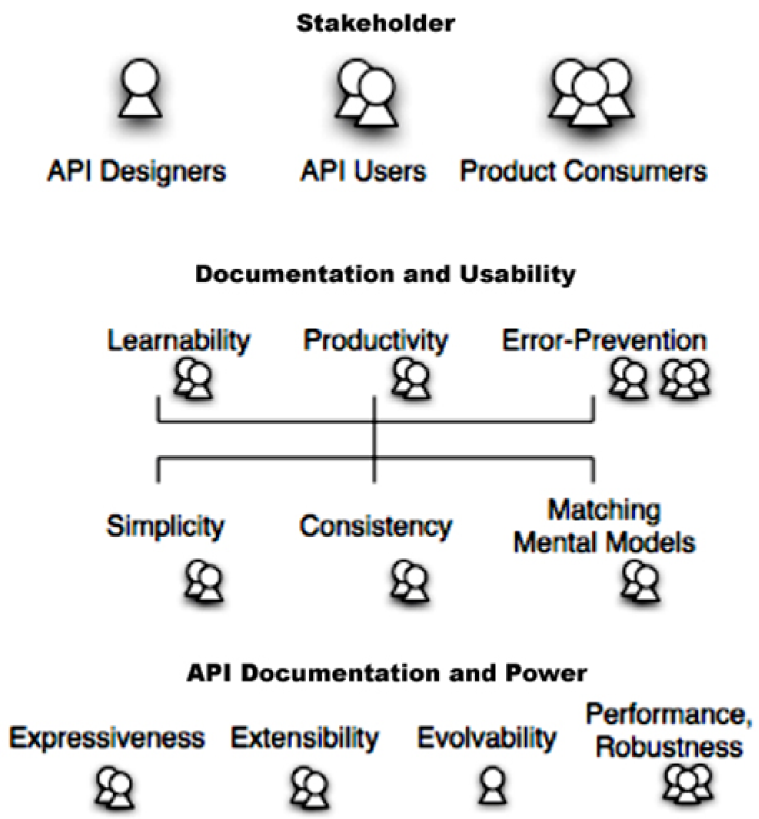


Figure 2.1 : Quality attributes of API documentation and the stakeholders most affected by each quality [16]

The expressiveness (what kinds of programs an API can generate), the extensibility (how users can extend the API to generate convenient user-specific components), the evolvability the performance (in terms of speed, memory, and other resource consumption), and the robustness and bug-freeness of the API implementation all contribute to an API's fomentation and power. Consumers of the resulting products benefit from error prevention, while API users benefit from usability. API designers are affected by the evolvability, but API users and product consumers are more directly impacted by the power [17].

Anecdotal research suggests that API usability may also play a role in driving user uptake. Some businesses may switch to a different API or develop less complex functionality in-house if learning the required API would take too long. Sometimes, it is necessary to give up one or more of these qualities in order to fully realize another. In other situations, however, modifying the APIs may actually boost all of the metrics or at least have no negative effects on any of them.

2.2 Techniques in API Documentation

Previous study frequently employed exploratory methods such as diary studies, interviews, and questionnaires. Despite their usefulness and contribution to the production of key findings, the fact that they dominate API documentation research creates a number of challenges. To begin, the ability of developers to self-report and reflect on their work habits, approach to problem solving, and shortcomings in the documentation they use is critical to the accuracy of the answers they provide during interviews and in response to a questionnaire. This talent could be found in a variety of forms.

What developers claim to accomplish may differ from what they actually do, therefore distinguishing between the two is critical when conducting interviews or filling out surveys. According to Alvaro et al. [18], people have a poor memory for ordinary occurrences and tend to recall only emotionally meaningful past experiences. It has also been observed that the activities in which developers claim to be involved are not always the same as those in which they are actually involved. Alvaro et al. [18], for example, found that 40 percent of software developers in their survey spent 40 percent of their time reading documentation. However, data acquired throughout the course of the observation period revealed that just 3 percent of all incidents reported were really documentation-related. We conclude that more study is required that goes beyond self-reporting and instead actively observes software engineers as they interact with an API [18].

The developer portal's home page introduces the API used in our research. This screenshot illustrates the structure and layout of the gateway at the time the test was run [19]. Concepts, API Reference, and Recipes are just a few of the information kinds accessible through tabs in the major menu [19].

Table 2: Content categories of the API documentation utilized throughout the examination

Category	Description
Welcome page	Developers' starting point; details developer registration and sandbox access procedures and provides links to further reading.
`Concepts`	An overview of the Application Programming Interface (API), including its authentication method, basic request structure and parameter handling, list of supported carriers and services, license details, pricing, and supplementary resources (integration guide, webhooks)
`Integrations`	Allows Ruby and other language users to gain access to specified API integrations. Add a reference to the GitHub repository where the integrations are kept.
`Samples`	Describes the most common scenarios, such as making a new shipment, editing an existing shipment, and requesting a quote for a shipment.
`Recipes`	Includes illustrative use cases for and explanations of carrier-specific services, such as weekend delivery, pickup, and delivery to designated drop boxes.
`API reference`	Includes a list of resources, descriptions of their parameters and payloads, and instructions for accessing, editing, and deleting those resources.

Source: Tao et al. [20]

2.3 Existing Documentation and Tools

Methods for efficiently mining large source code repositories for useful insights have been the subject of recent studies. Instead of relying on a central CVS repository or a language-specific search engine, Jadeite uses code snippets found in Google's standard online search results. There were two main factors that led us to choose this tactic. First, when compared to other code search engines and code repositories, Google's index has the most variety of examples. Second, to aim to be representative of typical usage: many code search engines are heavily influenced by a small number of really large open-source applications, whose use of a particular API is not always indicative of how a regular programmer may utilize it. Unlike compilable.java files, code snippets taken from the web are sometimes incoherent, difficult to understand, and even wrong. This complicates the task of developing a system to harvest data from the entire World Wide Web [21]. We still think this is the best approach for large, popular APIs like the Java standard APIs. However, a different approach may prove more efficient when using a different, perhaps proprietary API.

Cummaudo et al. [22] is one example of an API search engine that draws its data from snippets of websites. One of the main differences between Jadeite and these systems is that it uses a hierarchical browsing interface, similar to Javadoc, rather than a search interface [22]. It is possible to utilize both search and browsing interfaces together, as they complement one other well. Nonetheless, we decided to put our efforts into developing the best browsing-based interface possible for this project, in part because this allowed us to do our analysis in advance, allowing us to analyze more data while avoiding latency issues during use. In addition, browsing is a useful adjunct to search interfaces since it helps users zero in on the exact query they need to do a search for. Automatically, jungloids figure out how to go from one set of sorts to another. Redocly and other takes a different tack by showing the most common path to building a target type from a set of possible input types.

Recent work in repository mining has recommended the most popular parts of an API based on method popularity data. Redocly font sizes are similarly motivated, but presentation style (font sizes) and context affect how they are displayed (lists of classes in standard API documentation).

2.4 Program Tools and Documentation

The research confirmed the usefulness of code examples. It looks at the context of an IDE and finds and suggests code examples to use. It does not go looking for examples on the web or allow for direct queries, but it does solve the issue of learning how to use APIs to accomplish a goal by examining existing code snippets. It is unable to help programmers in situations where there is no starting point because its example search is implicit and based on the statements already present in the code. My tools make it easier for programmers to learn the basics of using a framework like Postman [23].

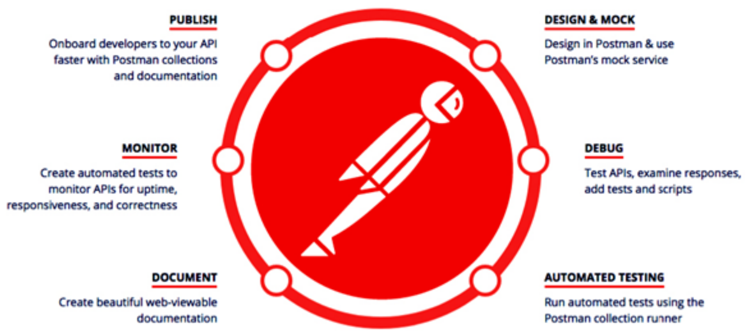

PUBLISH
Onboard developers to your API faster with Postman collections and documentation
DESIGN & MOCK
Design in Postman & use Postman's mock service
MONITOR
Create automated tests to monitor APIs for uptime, responsiveness, and correctness
DEBUG
Test APIs, examine responses, add tests and scripts
DOCUMENT
Create beautiful web-viewable documentation
AUTOMATED TESTING
Run automated tests using the Postman collection runner

Figure 2.2 : Program Tools and Documentation Procedures of Postman API [24]

Tools that automatically determine which parts of code are relevant to other parts of code by analyzing the aggregate use patterns of programmers were another inspiration for our tools. Tools built into integrated development environments (IDEs) like Team Tracks make it easier for developers to access the private APIs used by large projects. When a new team member looks at the paint() method, for example, invalidate() will be suggested if it is frequently looked at after paint() by other team members (). In contrast to the tools presented in this thesis, which are better suited to learning public APIs or open source projects large enough to have Internet discussion sites, this tool is better suited to learning private code, about which there may be no Internet information. In this thesis, we borrow ideas from applications like Team Tracks by using co-occurrence in web results to locate similar classes and procedures.

Many different search engines offer code 1 -specific query options. The problem is that it only searches a small subset of the known-good code repository. They will not help with the vocabulary problem. Developer interactions with APIs are observed by Nafi et al. [25], who use naive terminology because they do not bother searching informal forums and other pages that help developers find the right terminology. Their small size and lack of textual descriptions make it difficult for programmers to locate methods used for a specific purpose, even when they know the name of the method [25].

2.5 Measures API Documentation on the Web

Many programmers now rely on web searches for anything from learning new languages to referencing existing ones. When information is readily available with a quick Google search, the identity of the source becomes less significant to the seeker.

However, things were not always so simple. Developers used to have to rely on paper manuals, large documentation files, or their own time-pressed coworkers to get the information they needed. Despite numerous initiatives aimed at encouraging the development and upkeep of thorough documentation, such materials are frequently either lacking or insufficient even when they do exist. Written records quickly become obsolete. Because of this halt, distrust develops, and the documentation is rarely referred to in actual practice. Many surveys of software development firms have revealed that documentation is not only disregarded, but also varies widely in quality and scope [26].

Companies and API developers may be putting their own documentation online, but programmers are beginning to look elsewhere. A recent survey of more than 3000 MSDN developers confirmed that developers find out about new APIs primarily through web search. Furthermore, developers are more likely to discover a response on a blog than from that other colleague. Resulting from the widespread availability of social media and related infrastructure Growth in popularity of wikis, blogs, and online discussion forums has resulted in an explosion of information and communication [27].

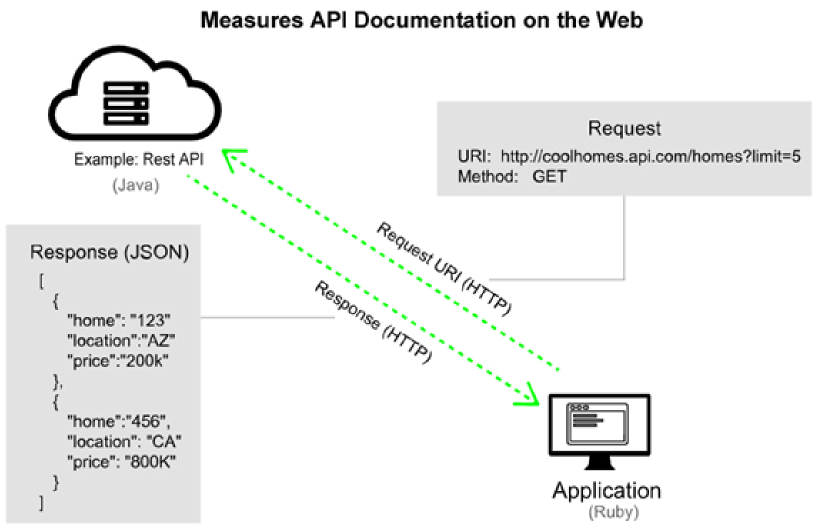


Figure 2.3: Measures of API documentation [28]

In crowd documentation, where users can send to documentation of an open platform via social media, the value of any given contribution is less important than the sum of all the contributions. Simply put, because you can find anything you need on the internet with a quick Google search. Regardless of who or where the information originated, the crowd documentation learning process treats all input equally. For example, a programmer can post a question on stackoverflow.com relating to almost any technical topic, and receive a detailed answer in the average of just 10 minutes. Not only does this solve the problem for the original programmer, but it also allows future programmers to view a curated and voted-on set of solutions to the same problem. No community vetting or formal processes relevant to social documentation are required for developers who contribute in just this fashion.

Chapter 3

3 Methods and Materials

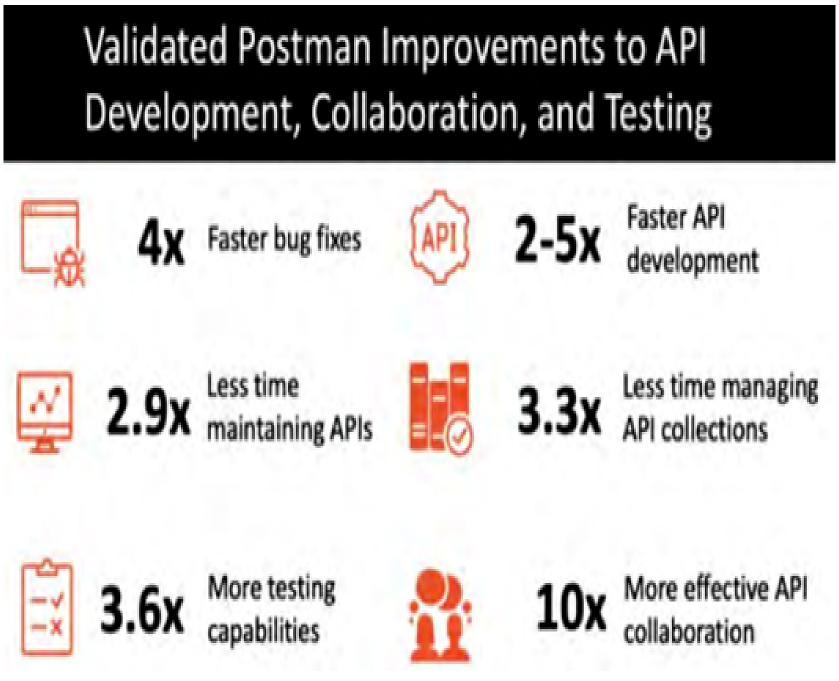
This section will walk you through the process, decisions, and tools involved in designing an application programming interface (API). First, I analyze the context of these decisions in relation to other development decisions, such as the design of the tool and the documentation. The rationale behind selecting Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST to document this particular API.

3.1 API Tools Overview

An application programming interface, or API, is a collection of software functions and procedures that other software applications can access or execute through. When performing API testing, software is utilized to make calls to the API, obtain output, and log the response of the system. Because of the increased pressure that shorter development cycles place on automated testing, API Testing is becoming increasingly important. The following is a discussion of certain API Tools. The list consists of tools that provide open source API documentation.

3.1.1 Postman

Postman is one of the best API automation and documentation tools available today. Postman has evolved from its humble beginnings as a Chrome plugin to become a full-featured API testing solution used by over 5 million developers and 100,000 businesses around the world. It has become the de facto standard for developing business APIs and is worth dollar 2 billion on its own. According to a recent report by ESG (Enterprise Strategy Group), the Postman API platform offers crucial tools for businesses to facilitate the simplification and quickening of API development, collaboration, delivery, and maintenance. Reduced risk, fewer bugs, and happier customers are just two outcomes of a shorter time to value, which in turn creates earlier revenue streams. As a result of using Postman, ESG estimated that a model company could save dollar (3.6) million on API testing and dollar (1.9) million on API development. Using Postman and adopting an API-first approach could further cut down on development time and costs for businesses by 4.7x [29].



Validated Postman Improvements to API Development, Collaboration, and Testing

4x Faster bug fixes

2-5x Faster API development

2.9x Less time maintaining APIs

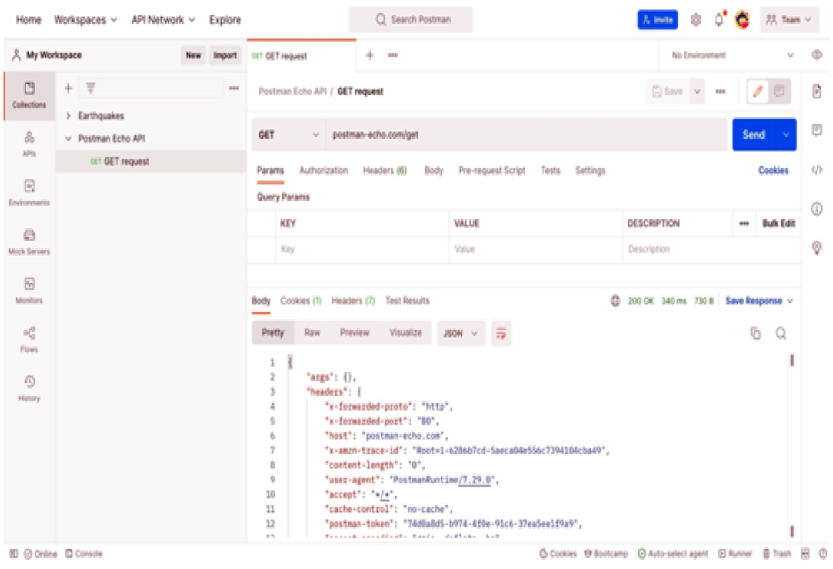
3.3x Less time managing API collections

3.6x More testing capabilities

10x More effective API collaboration

Figure 3.1: Postman Overview [30]

By using Postman, businesses can build new APIs, keep tabs on their current ones, and conduct continuous testing to ensure that their customers never experience any slowdowns or interruptions in service. Postman users, both solo programmers and members of development groups, exhibited high levels of self-assurance.



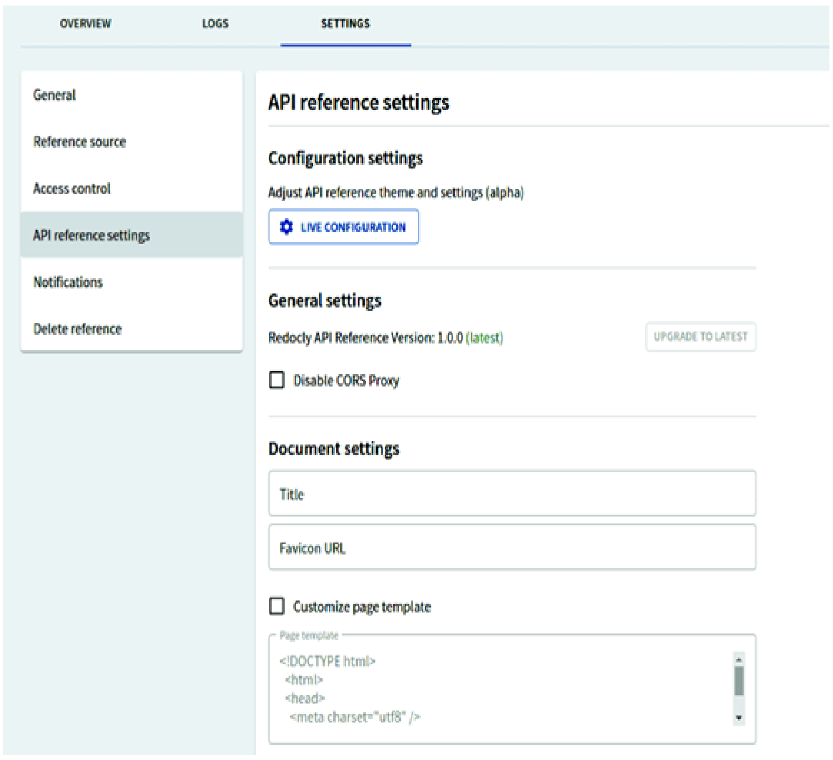
```
{
    "args": {},
    "headers": {
        "x-forwarded-proto": "http",
        "x-forwarded-port": "80",
        "host": "postman-echo.com",
        "x-amzn-trace-id": "Root=1-6286b7cd-5aeca04e556c7394104cba49",
        "content-length": "0",
        "user-agent": "PostmanRuntime/7.29.0",
        "accept": "*/*",
        "cache-control": "no-cache",
        "postman-token": "74d0a8d5-b974-4f0e-91c6-37ea5ee1f9a9",
```

Figure 3.2: Postman program introduction [31]

Growing pains in the API economy mean more work for programmers. Because modern applications can interact with hundreds of APIs, the old methods of creating and testing APIs manually no longer scale. To ensure that APIs are used to the full potential of the business and not as a hindrance, the development, testing, and delivery teams must collaborate. Modern API-powered applications require teamwork and efficient operations to be successfully delivered.

3.1.2 Redocly

API documentation is a crucial component in the success of a business in today's interconnected economy. Redocly assists businesses in optimizing the use of their application programming interfaces (APIs) by maintaining documentation that is clear, up to date, and available to both internal and external clients. Redoc has the capability to automatically generate API documentation by using the OpenAPI specifications. It is a docs tool that is both highly effective and free, and it generates documentation that is tidy, modifiable, and presented in an elegant three-panel layout [32].



API reference settings

Configuration settings

Adjust API reference theme and settings (alpha)

General settings

Redocly API Reference Version: 1.0.0 (latest)

☐ Disable CORS Proxy

Document settings

Title

Favicon URL

☐ Customize page template

Page template

```
<!DOCTYPE html>
 <html>
 <head>
  <meta charset="utf8" />
```

Figure 3.3: Redocly Open setting mode [33]

Following is the working procedure of Redocly.

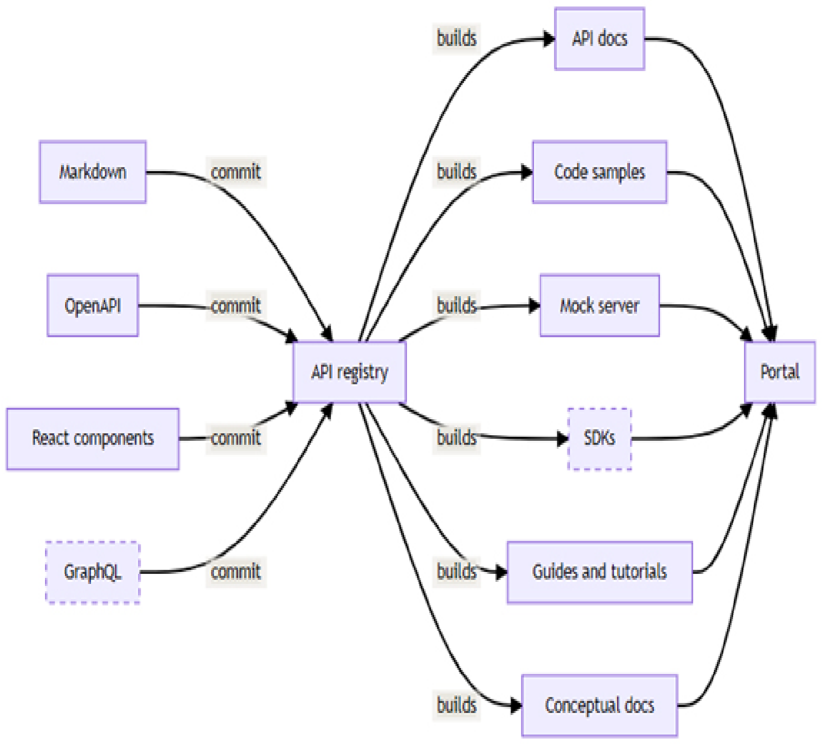


Figure 3.4: Redocly Working Procedure [34]

3.1.3 SwaggerHub

SwaggerHub is a platform for API design and documentation that aims to help teams enforce standards and discipline throughout the whole API development process. SwaggerHub offers free, team, and enterprise tiers. The Enterprise plan is the only one that may be deployed locally, however the other two are strictly SaaS. During the trial time, you will have access to all the same tools available to paid Enterprise plan subscribers, but if you do not upgrade to a paid Team or Enterprise plan, your account will revert to the free plan. Unless otherwise specified, the information presented here is applicable to any and all deployment strategies [35].

Formed in 2011, the Swagger API was initially conceived of by Tony Tam, technical co-founder of the online dictionary site Wordnik. Automation of API documentation and client SDK production became an enormous cause of irritation during Wordnik's product development. Using the adaptability of the REST architectural style and the various features of tools developed for the SOAP protocol, Tam created a straightforward JSON representation of the API. Ayush Gupta initially introduced the idea for the UI, noting that an interactive UI would be useful for end users who wanted to "test out" and build against the API. The first code generator was implemented by Ramesh Pidikiti, and the name Swagger was created by Zeke Sikelianos, a designer and developer. In September 2011, the source code for the Swagger API project was released to the public. A standalone validator, Node.js and Ruby on Rails support, and other features were incorporated into the project shortly after its initial release.

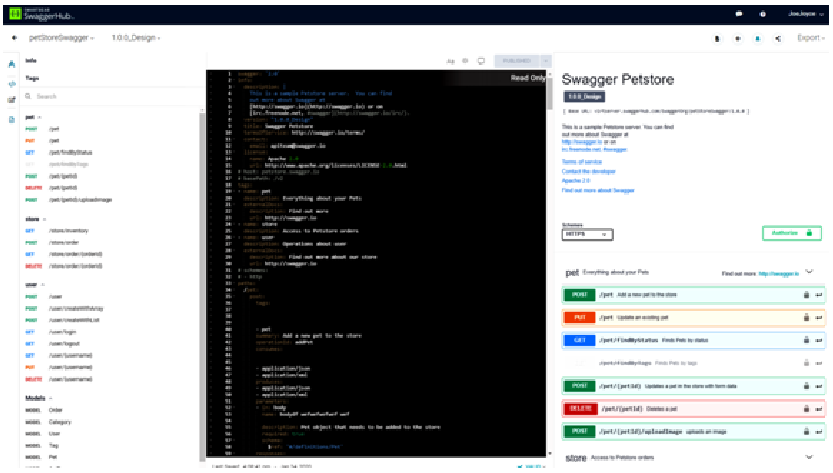


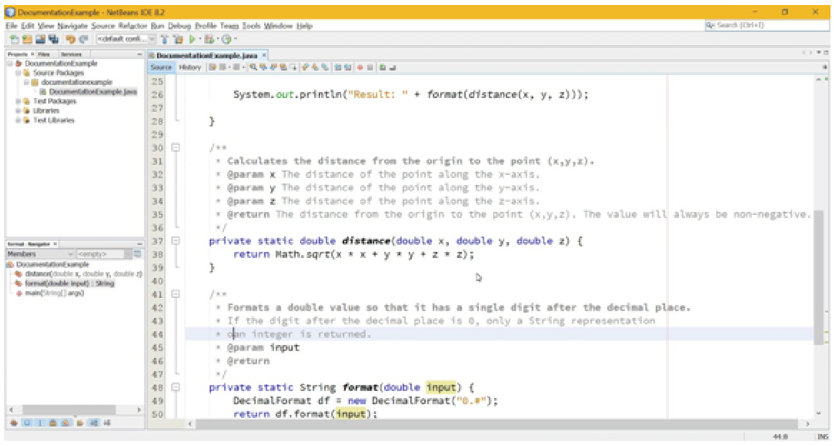
Figure 3.5: Swagger full view mode [36]

Swagger initially gained only minor traction from smaller organizations and individual developers. For RESTful APIs, Swagger offers a straightforward and easily discoverable means of describing their functionality for automated systems. Tony attended a meeting with key players in the API market, including John Demonstrated (ProgrammableWeb), Marsh Gardiner (Apigee, now a Google product), Marco Palladino (Kong), and Kin Lane (API Evangelist), to discuss the possibility of standardizing API descriptions. Even though no actionable plan came out of the conference, Swagger's importance as a game-changing innovation in the application programming interface (API) space was established.

Documenting APIs: The API is published as an OpenAPI document; if you want to interact with it directly through the Swagger user interface, you can use the open-source technology that is provided by Swagger. This body of work aims to accomplish the provision of a method by which users can connect to operational APIs through the utilization of an interface that is based on HTML and is not only dynamic but also friendly to users. The user interface makes it possible to immediately make requests and investigate available options.

3.1.4 JavaDoc

Sun Microsystems developed JavaDoc, which is currently held by Oracle Corporation, in order to generate API documentation in HTML format from Java source code. A major feature of the HTML format is the ability to simply build hyperlinks between related documents. Javadoc's "doc comments" approach of documenting Java classes is widely recognized as the gold standard. Several IDEs, including IntelliJ IDEA, NetBeans, and Eclipse, can generate Javadoc HTML automatically. Numerous file editors use the Javadoc information as an internal reference to assist the user in creating Javadoc source [37].



```
        System.out.println("Result: " + format(distance(x, y, z)));

    }

    /**
     * Calculates the distance from the origin to the point (x,y,z).
     * @param x The distance of the point along the x-axis.
     * @param y The distance of the point along the y-axis.
     * @param z The distance of the point along the z-axis.
     * @return The distance from the origin to the point (x,y,z). The value will always be non-negative.
     */
    private static double distance(double x, double y, double z) {
        return Math.sqrt(x * x + y * y + z * z);
    }

    /**
     * Formats a double value so that it has a single digit after the decimal place.
     * If the digit after the decimal place is 0, only a String representation
     * can integer is returned.
     * @param input
     * @return
     */
    private static String format(double input) {
        DecimalFormat df = new DecimalFormat("0.#");
        return df.format(input);
```

Figure 3.6: JavaDoc open mode [38]

Users can examine the framework of a Java program with the help of Javadoc's API for making doclets and taglets. With this, JDiff can produce reports detailing the differences between two API versions.

In Java, Javadoc has no effect on performance because comments are stripped out during compilation. It is important to document code with comments and Javadoc so that it can be better understood and maintained.

3.1.5 AutoREST

The OpenAPI 2.0 and 3.0 specifications can be converted into client libraries with the aid of AutoRest, a tool that offers a structure for code creation for this purpose. Microsoft created it around the time the OpenAPI Initiative was established so Azure service teams could begin developing client libraries based on the newest Swagger and OpenAPI 2.0 standards [39].

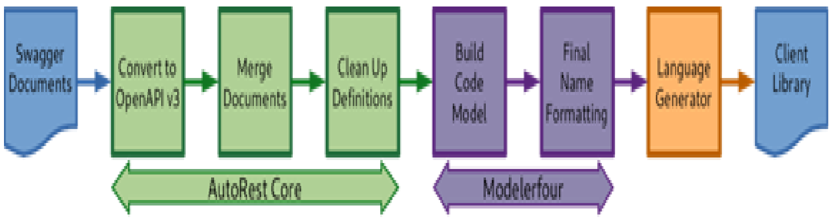
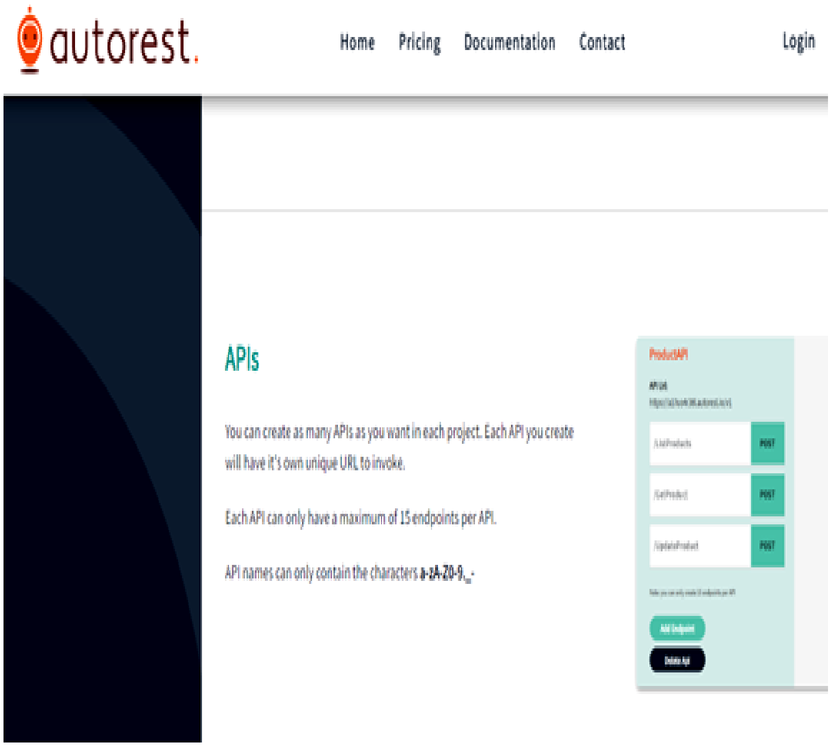


Figure 3.7: AutoREST pipeline [40]

Microsoft developed its own code generator despite the existence of alternative implementations, as limitations in Swagger 2.0 made it difficult, if not impossible, to express patterns for Azure services. AutoRest, for example, built and added implementations for x-ms-discriminator-value to distinguish between distinct schema types in requests and responses and x-ms-pageable to enable response collections to be paged via additional operation calls. The OpenAPI 3.0 specification finally incorporated constructs such as type discriminators and response operation linkages.



APIs

You can create as many APIs as you want in each project. Each API you create will have it's own unique URL to invoke.

Each API can only have a maximum of 15 endpoints per API.

API names can only contain the characters **a-zA-Z0-9_-**

Figure 3.8: AutoREST Webpage [41]

AutoRest is based on a configurable pipeline that integrates many OpenAPI input files into a single "code model" for usage with a language-specific code generator. These code generator add-ons will scan the code model and generate code that adheres to each language's particular style guidelines. Coding process for a language will utilize its related Azure Core implementation for HTTP request handling customization.

3.2 API Documentation and Comparison

This section provides a description of the design process as well as the planning procedures that were initially determined to be essential in order to begin the data gathering process that should compare the problem that was defined. In an effort to provide more clarity throughout the API documentation, chapter four has been subdivided. Documentation focuses on API Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST tools that are utilized, with a primary emphasis on how to obtain data that can be evaluated and processed.

3.2.1 Measurement and Comparison Process

Examining the documentation for an API allows one to see how different developers who are experts in the API's intended field approach typical programming challenges. It is possible to evaluate and identify the precise problems encountered by different users and the assumptions and methods that lead to these problems if these studies are performed in a usability lab, recorded, and the programmers are asked to think aloud while they work. Participants in a study on the usability of a file API, for instance, might be invited to build code that reads and writes files. The most frequent issues can be captured using screen capture. To give an example, certain types of programmers may have trouble picking and choosing the right items to use when creating a new file. They may wrongly assume the classes have a different name, or they may anticipate to utilize a single object instead of merging the several objects that the API tools require. These issues can be discovered by listening to recordings of programmers' spoken thoughts. Similar methods can be used to investigate the usability of an API design decision; however, rather than having developers work with a single API to complete tasks, they work with multiple APIs, each of which implements the decision in a slightly different way.

3.2.2 Compared Parameter

I made a study that compared API tools different ways to design object constructors so I could learn more about how to do it.

The first way was to only provide object constructors that needed all of the "syntax" properties of the object as parameters. For example, $"File(Stringpath)"$ is the constructor for a file object. MailMessage (MailAddress sender, MailAddress recipient, String subject, String body) is the constructor for an email message. By only giving out constructors that needed the right objects, these APIs could prevent certain errors without doing anything else. For example, a programmer could write compliable code that tried to read a file without saying which file to read or tried to send a message without saying who to send it to. The second way was to offer "default" constructors that did not need any parameters instead of or in addition to these constructors. $"File()"$ could be used to make a File object, and $"MailMessage()"$ could be used to make an email message. Programmers could set the properties of an object, such as $"file.path = "foo.txt""$ or $"mailMessage.recipient = "cj@msn.com,"$ to give the important information, such as the file-path or the email recipient. We called this method "create-set-call" because we started with an empty object and set its most important properties before calling other methods (such as read or send). This method seems to have a drawback in that it does not make the API's dependencies clear at compile time. For example, a programmer might not know that a mail message needs a sender or might forget to give one.

Some other things that i kept in my mind while comparing the tools, How to create the API documentation? Is it an automatic process or manual? How about the maintenance of the documentation? Creating a document is not the whole task, As the API will be there for a long time, the document also needs to be maintained properly. There could be some small modifications based on the need on the API, how easily that could be reflected in the document? What are the features and customizable options? And also how easy it is to understand the workflow of the API from the documentation?

Some major compared tagline of Postman, Redocly, SwaggerHub, JavaDoc, AutoREST below are noted:

- Tags or Syntax
- Advantage and disadvantage
- Developer and User Experience
- Support and Teamwork
- Tools Limitation
- Bugs and issues in the API's architecture

3.2.3 Analytical Comparison

Once enough information has been collected during the user experience phase, the analysis phase can commence. The acquired data will be analyzed thoroughly, providing some indication of the solution's effectiveness in meeting its stated goals. Here we will take a stab at guessing how accurate the API documentation is by manually comparing various sources. It is only natural that this subset could be influenced by the subjective biases of its participants. In an effort to eliminate bias, the manual comparison has not evaluate the significance of the discoveries but will instead establish the existence of the concepts inside the documentation pages.

3.3 API Documentation Analysis Stage

I provide a way to examine documentation evolution, which may be used to undertake a systematic and analytical examination of API documentation. We apply the classification schemes of Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST to the various revisions of API documentation for a given API library.

Stage 1: API documentation from the current and previous versions are compared head-to-head to identify changes. What we call a "document" in the context of API documentation is just a group of words that describe some part of an API. It is possible to add, remove, or change a portion of the API as it develops. Since our attention is on API documentation, we consider an API element to have been changed whenever either its declaration or its document is updated. When there are just minor changes between two sentences in a document, we say that there has been a revision. Any changes made are classified as either: a) a sentence that has been added; b) a sentence that has been removed; c) a sentence that has been altered. And last, from the discovered modifications, we extract several features, such as the occurrence of words and the position of changes.

Stage 2: Implementing a heuristic classification scheme for changes. Here, we start with a manual analysis of a sample size of a few hundred revisions, extracting features for different types. We develop heuristic rules specific to each category based on these criteria. We create a heuristic technique to categorize revisions based on their annotations, and our research shows that different sorts of annotations (such as @systax and @report) are linked to distinct keywords with specific typefaces. I use these heuristic API rules to categorize revisions into distinct groups. Differences between revisions that belong to different clusters are minimal.

Stage 3: Revising and evaluating sensitive information. Here, we use an iterative process to fine-tune our classifiers. Every time we run an iteration, we look at the misclassified findings to see if we can improve upon our current heuristics or come up with some whole new ones. We next repair any remaining incorrectly classified results and examine the entire process to draw conclusions about how API documentation has changed over time.

Chapter 4

4 Results

In this chapter, I examine the thesis's development process and the tools final standing in relation to tags or syntax, pros and cons, developer and user experience, tools' limitations, unblocks development from various perspectives, etc. Create an automatic overview mechanism that displays the location and relevance of resources based on keywords associated with the open API documentation development significant delay caused by exploring the open API documentation. Analysis of the project as a whole and its development as a topic of conversation.

4.1 Comparison on API Documentation Command or Syntax

4.1.1 Postman

Published Documentation: Each request and destination in the public collection is documented in full, and sample code is provided for a number of client languages. The available documentation is continuously updated to reflect the most recent changes to the collection. After adding edits to the documentation, it is not necessary to re-release it. Variables can also be used to temporarily store functions, making it more simpler to access and use them than the data they hold. Variables can also be used to temporarily store entire function definitions

```
var NameOfTheFunction = () => {
Function Definition
}
pm.environment.set(NameBywhichYouwantedtostoreavariable,
    NameOfTheFunction.toString());
```

The eval() method is used to call a function that has been stored in a system variable.

Postman is a powerful tool for performing API, sending HTTP queries, and viewing their answer due to its integration of variables and JavaScript functions. They offer a clean way to send requests through the Postman, whether through the body, a pre-request script, or a test run after the request has been executed. With sendRequest, one can initiate API calls from the post-execution test or pre-requirement script. This request's response is recorded in the function's return response, from which it can be retrieved and used in other requests or variables. Note the usage example under:

Function to send the request

```
pm.sendRequest({
 url: url,
 method:   POST  ,
 header: header,
 body: {
 mode:   raw  ,
 raw: JSON.stringify(body)
 }
 }, function (err, res) {}
```

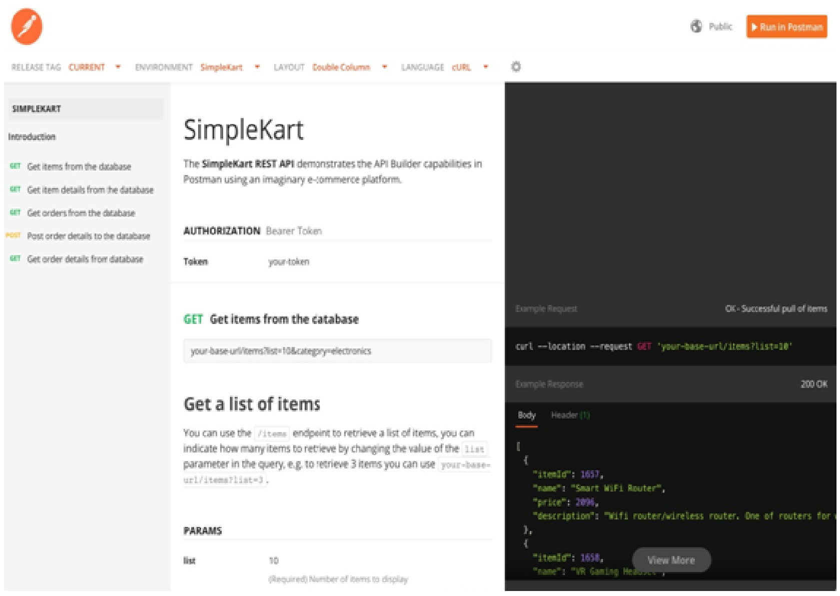
The response has become part of the collection and can be used by any further requests. This functionality also facilitates request chaining, which is useful for achieving the desired effect. To make a request stand on its own, you can produce the data it needs and make any calls to other requests that must be made in order to process it under the pre-requisite script section. When using a request chaining approach, it is important to ensure proper synchronization, or the waiting period between requests. Because JavaScript is asynchronous, pauses are required to instruct Postman to wait for the initial request to finish before proceeding. Here is an example of this in the code:

How to handle synchronization

```
setTimeout(function () {
// Operations to be performed on the completion of request if any
},Timeout)

Test Code
if (securedType!==     && securedType.length>0){
 postman.setNextRequest(Name of the same request);
API Body code
{
  SecuredType  :  {{currentSecuredType}}
}
```

Repeated requests are sent out until the 'securedType' variable has some information in it. In addition, Postman has a wide variety of built-in assertions that may be run on the API's output. However, utilizing different JavaScript libraries, you can make your own changes and assertions. Postman is compatible with a wide variety of libraries. In our application, we used the moment library to manipulate and verify dates and the arithmetic library to carry out a wide range of numerical computations.


Public
Run in Postman
RELEASE TAG CURRENT
ENVIRONMENT SimpleKart
LAYOUT Double Column
LANGUAGE cURL
SIMPLEKART
Introduction
GET Get items from the database
GET Get item details from the database
GET Get orders from the database
POST Post order details to the database
GET Get order details from database
Body
Header (1)
View More

SimpleKart

The **SimpleKart REST API** demonstrates the API Builder capabilities in Postman using an imaginary e-commerce platform.

AUTHORIZATION Bearer Token

Token your-token

GET **Get items from the catabase**

your-base-url/items?list=10&category=electronics

Get a list of items

You can use the `/items` endpoint to retrieve a list of items, you can indicate how many items to retrieve by changing the value of the `list` parameter in the query, e.g. to retrieve 3 items you can use `your-base-url/items?list=3`.

PARAMS

list 10

(Required) Number of items to display

Example Request OK - Successful pull of items

```
curl --location --request GET 'your-base-url/items?list=10'
```

Example Response 200 OK

```
[
  {
    "itemId": 1657,
    "name": "Smart WiFi Router",
    "price": 2096,
    "description": "Wifi router/wireless router. One of routers for
  },
  {
    "itemId": 1658,
    "name": "VR Gaming Headset",
```

Figure 4.1: Postman generated public API documentation [29]

4.1.2 Redocly

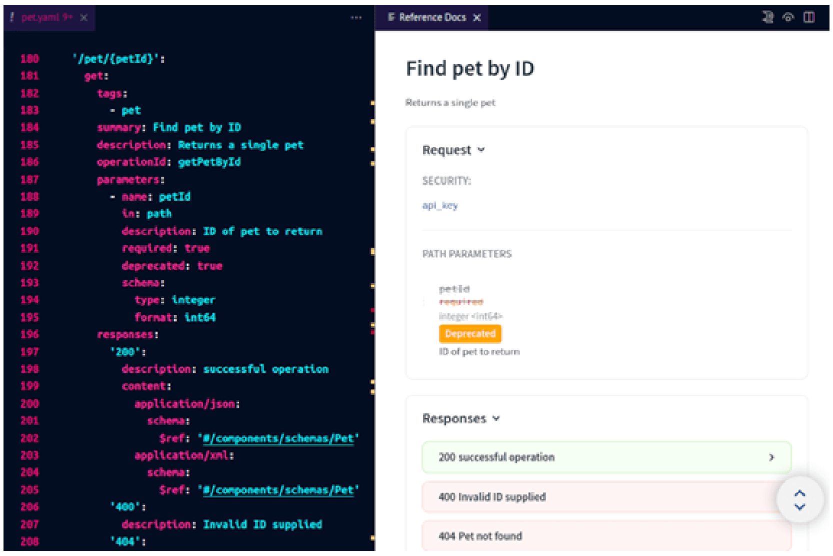
When multiple groups are working together, Redocy API definitions can quickly balloon out of control. Keeping the various reusable components in their own files and referencing them with ref in the main (root) API specification is recommended. The majority of OpenAPI tools, however, do not work with API definitions that span multiple files and instead necessitate a single unified document. You can use the Redocly CLI to merge multiple API definition files into a single document. The bundle command compiles all the necessary information from an API specification into a compact JSON or YAML file.

Table 3: Syntax and Option of Redocly

Options	Type	Description
`apis`	[string]	Names of API definition files or the identities you gave them in the apis section of your Redocly configuration file. All names specified in the apis section of your configuration file will be used as the default setting.
`-config`	string	Specify path to the config file.
`-dereferenced, -d`	boolean	Generate fully dereferenced bundle.
`-ext`	string	Identify the file type of the bundle. Options include json, yaml, and yml. Yaml is the default format.
`-extends`	[string]	Allows you to build on an existing setup using the –lint option. When a new instance of Redocly is created, its configuration file is used as the source for its default settings.
`-force, -f`	boolean	Produce output bundles notwithstanding incorrect conditions.
`-format`	string	Format for the output. Possible values are codeframe, stylish, json, or checkstyle. Default value is codeframe.
`-help`	boolean	Show help.
`-keep-url-references, -k`	boolean	Keep absolute url references.
`-lint`	boolean	Lint definition files. Default value is false.
`-max-problems`	integer	Truncate output to display the specified maximum number of problems. Default value is 100.
`-metafile`	string	Path for the bundle metadata file.
`-output, -o`	string	Name or folder for the bundle file. If you do not specify the file extension, .yaml is used by default. If the specified folder does not exist, it is created automatically. If the file specified as the bundler's output already exists, it is overwritten.
`-skip-rule`	[string]	Ignore some regulations. For information on how to bypass a preprocessor, rule, decorator, please refer to that section.
`-remove-unused -components`	boolean	Discard output bundle components that are not being used.
`-skip-preprocessor`	[string]	Leave out particular preprocessors. For information on how to bypass a preprocessor, rule, or decorator, please refer to that section.
`-skip-decorator`	[string]	A few interior designers should be disregarded. For information on how to bypass a preprocessor, rule, or decorator, please refer to that section.
`-version`	boolean	Show the version number.

Source: Redocly official documentation [32]

Live API Documentation: The Redocly OpenAPI VS Code extension relies on Redocly API docs to generate a preview of the API documentation based on the OpenAPI document you are editing. With this feature, you can view the OpenAPI source and the corresponding API documentation side-by-side. When you save a change to your OpenAPI definition, the documentation in the preview panel is automatically updated.



```yaml
  '/pet/{petId}':
    get:
      tags:
        - pet
      summary: Find pet by ID
      description: Returns a single pet
      operationId: getPetById
      parameters:
        - name: petId
          in: path
          description: ID of pet to return
          required: true
          deprecated: true
          schema:
            type: integer
            format: int64
      responses:
        '200':
          description: successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
            application/xml:
              schema:
                $ref: '#/components/schemas/Pet'
        '400':
          description: Invalid ID supplied
        '404':
```

Find pet by ID

Returns a single pet

Request

SECURITY:

api_key

PATH PARAMETERS

~~petId~~
~~required~~
integer <int64>
Deprecated
ID of pet to return

Responses

200 successful operation

400 Invalid ID supplied

404 Pet not found

Figure 4.2: Redocly generated public API documentation [32]

4.1.3 SwaggerHub

Interactive API Documentation: API specifications can be used with SwaggerHub to create dynamic API documentation. I can use it to try out API calls in your browser, investigate API endpoints, parameters, responses, and data models. The name convention for API docs pages is identical to that of API and domain pages, with the addition of a -docs suffix after the /apis (or /domains) portion of the URL:

https://app.swaggerhub.com/apis-docs/owner/name/version

https://app.swaggerhub.com/domains-docs/owner/name/version

If the version number is not specified, the following version will be displayed on the help page:

https://app.swaggerhub.com/apis-docs/swagger-hub/registry-api

Target server for requests -

When using SwaggerHub's "test it out" feature, your API definition must include the host (in OpenAPI 2.0) or servers (in OpenAPI 3.0) so that queries can be sent to the proper location:

```
swagger: '2.0'
host: myapi.com
schemes:
  - https
basePath: /v2

# or

openapi: 3.0.0
servers:
  - url: 'https://myapi.com/v2
```

Use SwaggerHub's dummy server in place of a real one if you do not have one yet or if you prefer not to expose your API to the public. The fake server will spit out responses according to the response schemas and examples you provide in your API description.

Permalinks: The API and domain documentation supports permalinks to individual operations, tags, models, and domain components. To get a permalink to a specific item, expand that item, and then copy the full link from the browser address bar: https://app.swaggerhub.com/apis-docs/swagger-hub/registry-api/1.0.47/APIs/searchApisAndDomains

Permalinks have the following syntax:

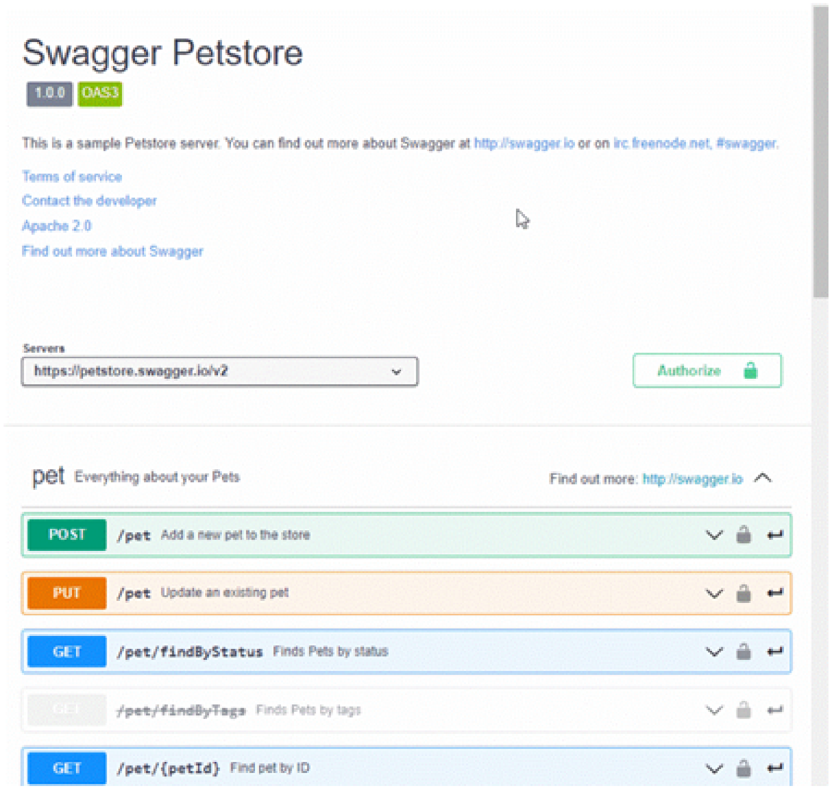
For tags and operations:

```
...#/TagName
...#/TagName/operationId
For other items:
...#/ItemType
...#/ItemType/ItemName
```

Table 4: Link Operation of SwaggerHub

Example	Description
.../pet	Link to the pet tag.
.../pet/updatePet	Link to the operation with ID updatePet inside the pet tag.
.../models/Order	Link to the Order model.
.../parameters/limitParam	Link to the limitParam parameter definition in an OpenAPI 2.0 domain.
.../components/parameters/limitParam	Link to the limitParam parameter definition in an OpenAPI 3.0 domain.

Source: Smart bear SwaggerHub support [46]



Swagger Petstore

1.0.0 OAS3

This is a sample Petstore server. You can find out more about Swagger at http://swagger.io or on irc.freenode.net, #swagger.

Terms of service

Contact the developer

Apache 2.0

Find out more about Swagger

Servers

https://petstore.swagger.io/v2

Authorize

pet Everything about your Pets — Find out more: http://swagger.io

POST /pet Add a new pet to the store

PUT /pet Update an existing pet

GET /pet/findByStatus Finds Pets by status

GET ~~/pet/findByTags~~ Finds Pets by tags

GET /pet/{petId} Find pet by ID

Figure 4.3: SwaggerHub API documentation [46]

4.1.4 JavaDoc

Documentation Comments: Whenever I make a declaration for a new class, interface, method, constructor, or field, I may add comments to the code that explain what that declaration is for (these are called "doc comments"). Each package can have its own doc comment, and the overview can have its own, with somewhat different syntax. An alternative name for doc comments is "Javadoc comments" (but this term violates its trademark usage). The text between the doc comment's opening /** and closing */ tags is considered a comment. Additional information about the use of leading asterisks on each line follows. A comment's text may span many lines if necessary.

```
/**
 * This is the typical format of a simple documentation comment
 * that spans two lines.
 */
```

To save space you can put a comment on one line:

```
/** This comment takes up only one line. */
```

Placement comments: For examples of classes, methods, and fields, respectively, see the examples of class, interface, constructor, and field declarations. Documentation comments within a method's actual body are disregarded. The Javadoc tool only accepts a single documentation remark per declaration statement.

An incorrect import statement is often placed between the class comment and the class declaration. Never do this, as the Javadoc tool will not take into account the class comment if I do.

```
/**
 * This is the class comment for the class Whatever.
 */
import com.sun; // MISTAKE - Important not to put import statement
    here

public class Whatever {
}
```

After the opening delimiter /**, the main description continues until the tag section. The initial @ character at the beginning of a line (ignoring preceding asterisks, white space, and separator /**) marks the beginning of the tag section. A comment can contain a tag section but no main description. Once the tags section begins, the main description must end. A tag's argument may be presented across multiple lines. There is no limit to the amount of tags that can be used, albeit some tag types can be reused while others cannot. This @see, for instance, initiates the tag section:

```
  /**
* This sentence would hold the main description for this doc comment.
* @see java.lang.Object
*/
```

Tags are keywords used in doc comments that are recognized by the Javadoc tool. Tags can be either block tags, written as @tag (or "standalone tags"), or in-line tags, written as @tag> (or "curly braces"). Any leading white space, asterisks, or separator characters will be disregarded when determining the meaning of a line that begins with a block tag. Thus, the @ symbol is not considered to be the beginning of a tag when used outside of the context of a tag. Use the HTML entity 064; to start a line with the @ symbol without having it be interpreted. The text after a block tag up to but not including the next tag or the end of the doc comment is called "related text," and each tag has its own. The accompanying text may be long enough to fill many lines. Anywhere normal text can go, an in-line tag can go as well, and vice versa. The following code snippet uses the inline tag @link and the block tag @deprecated.

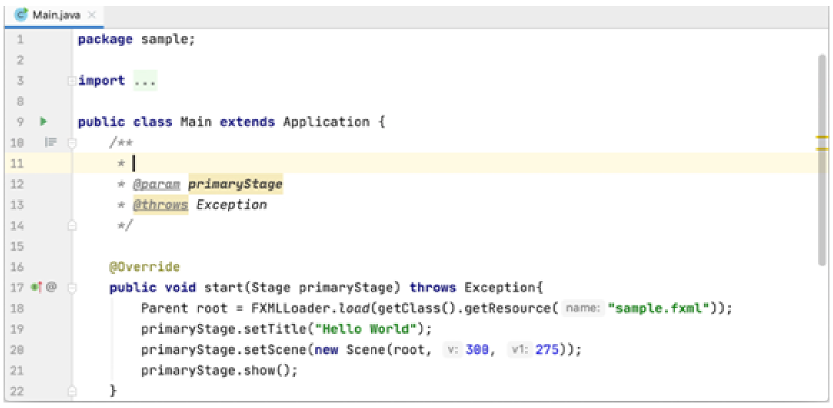
```
  /**
* @deprecated As of JDK 1.1, replaced by {@link
   #setBounds(int,int,int,int)}
*/
```

Using HTML entities and tags are permitted, and the text itself must be written in HTML. If your browser supports HTML 3.2, you can use that instead of the older versions; the standard doclet was created to output HTML 3.2-compliant code outside of the documentation comments, so you can utilize things like CSS and frames. Because of the frame sets, we label all created files as "HTML 4.0."

For instance, the less-than () and greater-than (>) symbols' respective entities should be written as and respectively. It is also correct to use for the ampersand. This example demonstrates how to use the b> tag for bold text in HTML.

Here is a doc comment:

```
  /**
* This is a <b>doc</b> comment.
* @see java.lang.Object
*/
```



```
package sample;

import ...

public class Main extends Application {
    /**
     * |
     * @param primaryStage
     * @throws Exception
     */

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource( name: "sample.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root,  v: 300,  v1: 275));
        primaryStage.show();
    }
```

Figure 4.4: JavaDoc print code

4.1.5 AutoREST

Command Line Interface Documentation: Simplifying the AutoRest command line was a priority, and many options that used to be available directly on the command line are now stored in a configuration file, which can be overridden by the user.

```
autorest [config-file.md | config-file.json | config-file.yaml]
    [additional options]
```

AutoRest will utilize a configuration file to govern the production of source code. AutoRest will search for a file named readme.md by default, or it can be specified on the command line. This strategy is favored over passing all the information on the command line. If you prefer a different name for your configuration file, you can specify it on the command line:

```
autorest my_config.md
```

By appending a double dash (–) before a value and setting it to equals (=), you can override a setting from the configuration file on the command line. example:

```
autorest --input-file=myfile.json --output-folder=./generated/code/
    --namespace=foo.bar
```

AutoREST allows for code to be generated straight from private GitHub folders. Choices include:

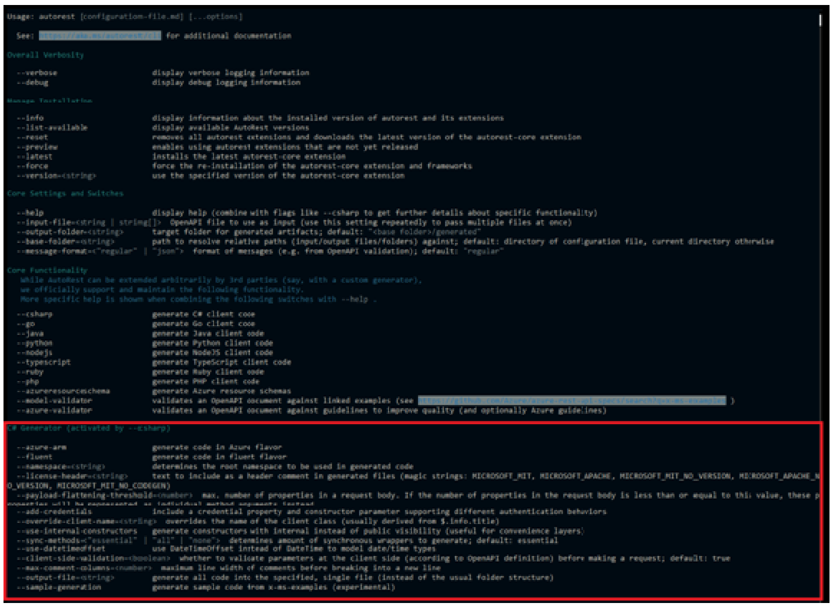
Token query parameter: When accessing a file in a private repository, after selecting "Raw," append the token query parameter to the end of the repository's URL, such as this: https://github.com/path-on-some-private-repo>/readme.md?token =token>. With this kind of URI in hand, AutoRest will remember the token and utilize it for any future requests (e.g. querying referenced OpenAPI definitions). If you occasionally need to run AutoRest on any sensitive data on your own, this is a simple and efficient way to do it.

Create your own OAuth tokens in GitHub's Settings > Personal access tokens. Make a version that can be used throughout the repository. Set the env var GITHUB AUTH TOKEN or use the –github-auth-token=token> option to provide it to AutoRest. All automation and scripts should be written in this fashion. Obviously, you should not just type this token into your scripts without first storing it securely somewhere like Azure KeyVault. An organization's repository may need special access permissions granted to the Github Token. (Near the token that reads "Enable SSO> " Select the necessary group and then click the Authorize button.

Table 5: Common flags of AutoREST

Option	Description
–input-file=FILENAME	Including the specified file in the generating process's input file list.
`-output-folder=DIRECTORY`	The folder in which the output files will be placed. The default location is./generated if not otherwise specified.
`-clear-output-folder`	Before writing your newly generated code to our output folder, please delete all of the existing files in that directory.
`-namespace=NAMESPACE`	adjusts the resulting code's namespace
`-add-credential`	A credential will be needed before the produced client may perform network calls, if that option was set during generation. For details on how to verify access to our created clients, please refer to our language documentation.
`-tag=VALUE`	the most favored method of setting up conditional arrangements. Simply put, the value of the tag flag VALUE determines the value that will be applied to the input-file in my configuration file. For more details, please review the part titled "Adding Tags When Generating."

Source: Generating Clients with AutoREST [47]



```
Usage: autorest [configuration-file.md] [...options]

  See: https://aka.ms/autorest/cli for additional documentation

Overall Verbosity

  --verbose                      display verbose logging information
  --debug                        display debug logging information

Manage Installation

  --info                         display information about the installed version of autorest and its extensions
  --list-available               display available AutoRest versions
  --reset                        removes all autorest extensions and downloads the latest version of the autorest-core extension
  --preview                      enables using autorest extensions that are not yet released
  --latest                       installs the latest autorest-core extension
  --force                        force the re-installation of the autorest-core extension and frameworks
  --version=<string>             use the specified version of the autorest-core extension

Core Settings and Switches

  --help                         display help (combine with flags like --csharp to get further details about specific functionality)
  --input-file=<string | string[]>  OpenAPI file to use as input (use this setting repeatedly to pass multiple files at once)
  --output-folder=<string>       target folder for generated artifacts; default: "<base folder>/generated"
  --base-folder=<string>         path to resolve relative paths (input/output files/folders) against; default: directory of configuration file, current directory otherwise
  --message-format=<"regular" | "json">  format of messages (e.g. from OpenAPI validation); default: "regular"

Core Functionality
   While AutoRest can be extended arbitrarily by 3rd parties (say, with a custom generator),
   we officially support and maintain the following functionality.
   More specific help is shown when combining the following switches with --help .

  --csharp                       generate C# client code
  --go                           generate Go client code
  --java                         generate Java client code
  --python                       generate Python client code
  --nodejs                       generate NodeJS client code
  --typescript                   generate TypeScript client code
  --ruby                         generate Ruby client code
  --php                          generate PHP client code
  --azureresourceschema          generate Azure resource schemas
  --model-validator              validates an OpenAPI document against linked examples (see https://github.com/Azure/azure-rest-api-specs/search?q=x-ms-examples )
  --azure-validator              validates an OpenAPI document against guidelines to improve quality (and optionally Azure guidelines)

C# Generator (activated by --csharp)

  --azure-arm                    generate code in Azure flavor
  --fluent                       generate code in fluent flavor
  --namespace=<string>           determines the root namespace to be used in generated code
  --license-header=<string>      text to include as a header comment in generated files (magic strings: MICROSOFT_MIT, MICROSOFT_APACHE, MICROSOFT_MIT_NO_VERSION, MICROSOFT_APACHE_N
O_VERSION, MICROSOFT_MIT_NO_CODEGEN)
  --payload-flattening-threshold=<number>  max. number of properties in a request body. If the number of properties in the request body is less than or equal to this value, these p
properties will be represented as individual method arguments instead
  --add-credentials              include a credential property and constructor parameter supporting different authentication behaviors
  --override-client-name=<string>  overrides the name of the client class (usually derived from $.info.title)
  --use-internal-constructors    generate constructors with internal instead of public visibility (useful for convenience layers)
  --sync-methods=<"essential" | "all" | "none">  determines amount of synchronous wrappers to generate; default: essential
  --use-datetimeoffset           use DateTimeOffset instead of DateTime to model date/time types
  --client-side-validation=<boolean>  whether to validate parameters at the client side (according to OpenAPI definition) before making a request; default: true
  --max-comment-columns=<number>  maximum line width of comments before breaking into a new line
  --output-file=<string>         generate all code into the specified, single file (instead of the usual folder structure)
  --sample-generation            generate sample code from x-ms-examples (experimental)
```

Figure 4.5: AutoREST documentation [47]

4.2 Pros and Cons

4.2.1 Postman

Advantage: The fact that Postman is so easy to work with is among the program's many strong points. Testers can quickly create test cases by populating pre-made forms using an intuitive interface. This saves them a lot of time. The code snippets provided by Postman make it easier to construct scripts by providing validation examples for response time, response code, and other aspects of responses

Disadvantage: There is currently no support for global variables in the Postman Monitor. When we set up a Monitor, we have to make a copy of all of the global variables and save them as local variables if we want to make use of the same values for a variety of purposes within a freshly fashioned Environment. Only by using the free edition of Postman Monitor can we get a sense of what capabilities it possesses.

4.2.2 Redocly

Advantage: The open-source API documentation tool Redoc serves as the foundation for the SaaS application known as Redocly. While Redoc generates straightforward and free API documentation, Redocly adds several elements to provide developer portals that may be tailored to individual needs. API providers are able to document, create example code and API references.

Disadvantage: There is no interactive playground in the free version, error messages that are not very helpful when the OpenAPI specification cannot be parsed, and I'm not a fan of the way that sets of request/response attributes were presented.

4.2.3 SwaggerHub

Advantages: Swagger provides a technique to automate the documentation, that indicates that Swagger will pick up the methods that include attributes such as GET, PUT, POST, and DELETE and will produce the documentation in of itself. In addition, the documentation for Swagger is updated automatically if any changes are applied to the system.

Disadvantages: SwaggerHub When it comes to integrations, SwaggerHub falls short. They have integrated APIs and certain connectors, but they do not have integration with many of the things that we use. On the other hand, they do have integration with some connectors. For example, we were required to use third-party scripts in order to establish a connection with SVN.

4.2.4 JavaDoc

Advantages: Good API documentation can be generated with the assistance of the Javadoc tools, although the vast majority of Java API documentation is lacking. The engineer is ultimately responsible for the API's documentation because the API itself is part of the source code. In this piece, Brian critiques the state that Java documentation Practice is in right now and offers some advice on how to produce Javadoc that is more helpful.

Disadvantages: It has the potential to make your code exceedingly muddled. Therefore, you should ensure that the comments are succinct while still providing a lot of detail. Should this not be done correctly, it has the potential to confuse everyone and lead to errors in the code created by other developers. It is time-consuming to write, and it is necessary to manage it when the functionality of the code evolves.

4.2.5 AutoREST

Advantages: AutoREST's documentation is simpler, provides more human-friendly results, and supports a wide variety of file formats. AutoREST's stateless calls are ideal for cloud-based apps since they require no state maintenance between REST executions and are simple to reload and scale. If you plan on using CSG Forte products and functions like Digital Terminal, reporting, or Forte.js, then you should definitely go with AutoREST instead than SOAP. AutoREST is ideal for ISVs with several merchant or third-party app developers who wish to receive and leverage our lead to specific because it requires a developer to consume our API requests.

Disadvantages: AutoREST APIs' potential drawback is that it is possible to lose the capability of maintaining state via REST, such when within sessions. Also, there can be more of a learning curve for novice programmers. Before beginning API development, it is crucial to have a firm grasp on what constitutes an AutoREST API and why these limitations are in place.

4.3 User Experience: Tools

Throughout the lifecycle of a product's development, designers, developers, and users rely on API documentation tools to study, design, and test prototypes of their work in progress. Skillsets in both front-end development and user research are required to map into the front user's experience and optimize it for maximum efficiency.

Table 6: User Experience on API tools

`Tool Name`	Platform	Regarding tool	Best for	Pricing.
`Postman`	Windows, Mac, Linux, and Chrome browser-plugin	It is an API development environment with good documentation support.	API testing	Basic free package, Professional plan costs USD 29 per month, Business plan is USD 99 per month.
`Redocly`	Windows, Mac, Linux.	Different type APIs may be tested for their functionality, security, and load issues.	APIs and documentation for web service security and functionality were improved.	Free basic package, Standard package costs USD 69 per month.
`SwaggerHub`	Chrome, Firefox, Safari, Edge	It is a one-stop shop for all your API development and management needs	The tool is best for API designing.	Free individual use, For teams USD 90 per month.
`JavaDoc`	Windows, macOS, Linux	API, online, desktop, and mobile testing tool for beginners and pros.	Automated testing	There is a free tier of service, but any additional help will cost money.
`AutoREST`	Cloud-based continuous testing	Integrates seamlessly with UI documentation generation and automates API tests without the need for code	Fully automated API testing with codeless automation logic, full document management, and API error monitoring and tracking.	There is a free trial that you can try. Pricing Budget at least USD 150 per month for comprehensive automation of your API, UI, database, and mainframe.

4.4 Support and Teamwork

Appropriate design, development, and documentation are cornerstones of distributing high-quality APIs. To create the ideal application programming interface (API), members of different software development groups often work together and share knowledge. The API development process will be iterated and deliberated upon in teams using SwaggerHub, and Postman and AutoREST will walk through a guide on forming and managing organizations.

The creation of an API is a collaborative effort involving many parties. In order to build an API, you must first identify its needs. Is this an internal API used only by the corporation to improve its own software, or is it an external API available to the public in the hopes of boosting sales and brand recognition?

The API would determine if public or private APIs are necessary based on the specifications provided by the product owners and level expert. In the SwaggerHub Editor, API owners can specify exactly who can access their API.

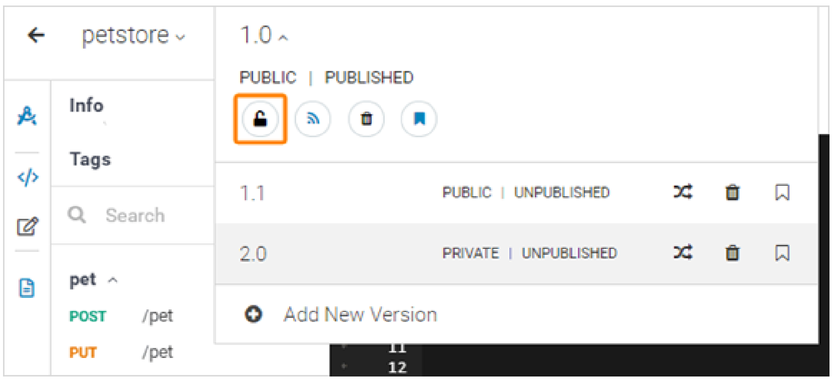


Figure 4.6: SwaggerHub Editor [46]

The idea for a private API typically originates with technological leads and development managers who see the value in having an internal API to streamline the creation of new apps within the company. Discussions on public APIs, which are available to anyone to use, are typically led by product owners, marketing managers, and technical leads. Only until the API design and development team has thoroughly investigated and settled upon the necessary functional technologies as documented in JavaDoc can the API be made a reality.

Developers should prioritize the needs of their organizations when designing and developing software. Infrastructure, deployments, firewalls, domains, licensing, procurement, networking, and similar tasks offer no value to the business and should not be delegated to developers. When a developer has a pleasant time with the tools we provide, they can devote their time to improving our products and adding new features that will make our customers' lives easier.

4.5 API documentation Tools Limitation

API documentation are generally the first thing developers look at when they decide whether or not to use a specific product. It provides weight to the cliché that an API is only ever as good as its documentation. The API might never be released if it does not come with solid documentation. As a result, API designers must now devote more time and effort to this stage of the project's creation. Hosted API documentation from API specialists such Postman, SwaggerHub is one approach. Designers have an advantage when it comes to releasing high-quality, fully-functional documentation when they utilize open-source solutions like API documentation tool packages.

- **Advertisements Using the API:** Unfortunately, marketing brochures can only ever provide a high-level overview of the API. While you are showing the documentation to someone with greater experience with API development, they may want more in-depth information. Then, one should go with a more comprehensive API documentation style.
- **API Reference Documents:** The goal of API reference guides, a specific sort of documentation, is to provide a comprehensive overview of the API. Information such as the API design team's goals, commonplace themes, the API's endpoints and parameters, a history of changes, and a collection of commonly asked questions can be helpful. The reference book may look like a certain way to document the API, but it may also be seen as overly "vanilla" or unimpressive. Users of the final product would prefer more engaging documentation, especially in light of the inventive approaches taken by competing API design teams.
- **Topical API References for Greater Depth:** If business API documentation consists solely of in-depth articles on certain topics, readers may become overwhelmed. The guidelines' comprehensiveness may be for naught if readers find them too difficult to follow
- **How-to Guides for Pasting in API Code:** One potential drawback of recipe-based API testing and documentation is that it may cause developers to over-specialize in a few key areas. We must guide them to the "larger picture" of the API, not just the functional pieces. And the API documentation as code will only be useful if the examples are applicable to the users' needs. The key is adapting the system to the specifics of actual use cases.

4.6 Bugs and Issues in the API's architecture

Neglecting to Offer Instruction: The lack of a clear path to mastery is a typical flaw in developer-written API documentation. Libraries often provide extensive reference documentation, but they do not always make it easy to get started with the fundamentals, and they do not always provide guidance for moving beyond the

"I went through the tutorial" stage and into the "I am a wizard" stage. If business do not give their users a way to get up to speed on the API, it will have a much harder time being adopted and used. To begin, let us address the elephant in the room: confusion over how to get started. It will not be easy to get rid of an API that's hard to learn and even harder to get started with. It is going to be tough to get anyone to embrace an API if it takes them days to get any value out of it.

Putting Emphasis on Procedures While Ignoring Ideas: API documentation relies heavily on tutorials. They are the easiest method to begin using a new piece of technology, and their purpose extends far beyond that of mere instruction. Tutorials are like test drives for knowledge. It gives potential customers a chance to experiment with their technology, learn what works and what does not, and judge the quality of its documentation and development tools, among other things. We can see results right away, which makes people more comfortable with given technology and energized about what they can create with it. Even worse, tutorials that are short on details promote copy-and-pasting code, which is a major contributor to the proliferation of defects and poor practices. It also has the unintended consequence of stifling innovation, engagement, and development within the community. Only so many things can be made by only cutting and pasting, and none of them are especially impressive.

Absence of Sample Code: One of the most common issues, and the most aggravating for developers, is the focus on providing reference information without supplying any code at all. It is also the most effective, typically.

It is especially problematic for developers who are just starting out with your API (possible adopters and fans) since there is no example code available.

Typically, we picture a book or a manual when we think of API documentation. When we are more experienced with an API, we often look to the documentation to learn the meaning of a parameter or the value we should supply to trigger a specific effect. Questions like "what errno values may read(2) set?" and "how does parseInt pick what radix to use?" come to mind. However, that is just the most basic documentation of records. Good documentation does not merely catalogue facts; it fosters comprehension. One of the most effective methods of constructing it is through studying existing examples of code. Many seemingly simple classes, such as TextureBrush, actually include a lot of sophisticated features. However, take note of how the abundant usage of sample code makes the whole thing manageable.

The Inability to Construct Anything Valuable: It is smart to count on users' established familiarity with the system. Having prior expertise is helpful because it prevents you from having to waste time teaching concepts that are already understood. Its worth lies in the fact that it contains more than just "raw" knowledge; for example, it may contain information on what to expect from a system, what to do in unusual situations, and so on. Even if you wouldd rather not, there are occasions when you have to start from scratch. You need to take the time to document concepts that are essential to your users' ability to write any code at all but will never directly result in them writing anything.

Not Thinking Outside the Box: Compiling complex libraries can be more involved than simply executing make or pressing F9 (this is just one example). Does your organization have a blog? Why not tell the world about the one that involves some non-trivial setup but ultimately provides a superior development experience? An excellent illustration can be found in JavaDoc. This stuff can be quite useful, especially for newbies. They also help to ease transitions; working with a novel API in one's preferred setting makes the process feel less daunting.

Chapter 5

5 Discussion

In this section, I take a look back at the work that went into this thesis and evaluate how the final product stacks up against what I covered in the first section. That section focused on API documentation and the importance of key phrases related to open APIs in order to solve the problem of a time-consuming delay in development caused by the need to explore open API tools. In this section, we will investigate the evolution of the thesis and how it relates to the remainder of API documentation.

5.1 API Documentation Design

The design of the API for rebranding is built around the use case description and the requirements of the stakeholders. In addition, the design takes into account quality characteristics that are regarded as being relevant based on previous research in the field. Usability and power are the two aspects that Henning [42] see as being important. As a result, the design ought to incorporate components that facilitate learnability, uniformity, flexibility, performance, and error prevention. Table 7 contains a rundown of the aspects of the design that contribute to the quality traits being promoted. The design is primarily based on the bolt-on method that was outlined by Meng [43]. It is anticipated that there is a backend system that the APIs will be built on top of when employing this technique. The backend has been left mostly unaltered and is kept entirely separate from the APIs and the mobile application. In addition to this, the design makes an effort to accommodate the requirements of the various stakeholders, which are detailed in table 8.

Table 7: Relationship between final product quality and design decisions

Quality attribute	Design Solutions
Learnability	Documentation that can be read by both machines and humans, The API's function names should be consistent ,uniformity in style
Consistency	The API's function names should be consistent, uniformity in style
Extensibility	Taking API architecture into account: Which questions/fields are required?, Can additional fields be added to the API at a later date?
Performance	Reducing the number of times slow back-end services are called, Caching
Error prevention	Error checking and recovery, Implement proper error codes

Table 8: Primary players of APIs

Stakeholder	Main Needs
API developer and designer	Simplicity in design and implementation
Storage worker	Program that actually does its job, free of errors
Application developer	APIs that are easy to learn and find, Effective and rapid application creation
Customer IT personnel	Lower the expense of development as much as possible.

The GET method is used by every resource to retrieve a copy of itself. Unit also facilitates resource updates and additions using the PUT and POST methods. This is essential for transferring troops across commands. It is recommended to follow the guidelines provided by the framework when naming endpoints and URIs. Correct use of verb tense, articles, and singular/plural nouns is essential. Lowercase letters are used exclusively, hyphens are used in place of spaces, and there is no room for creativity in the naming scheme.

The return sets from the rebranding API will be in JavaScript Object Notation (JSON). Content negotiation and other markup languages may be added in the future if they prove useful, but they are not currently supported. Table 9 describes the HTTP status codes and their meanings.

Table 9: HTTP Status Codes for the API

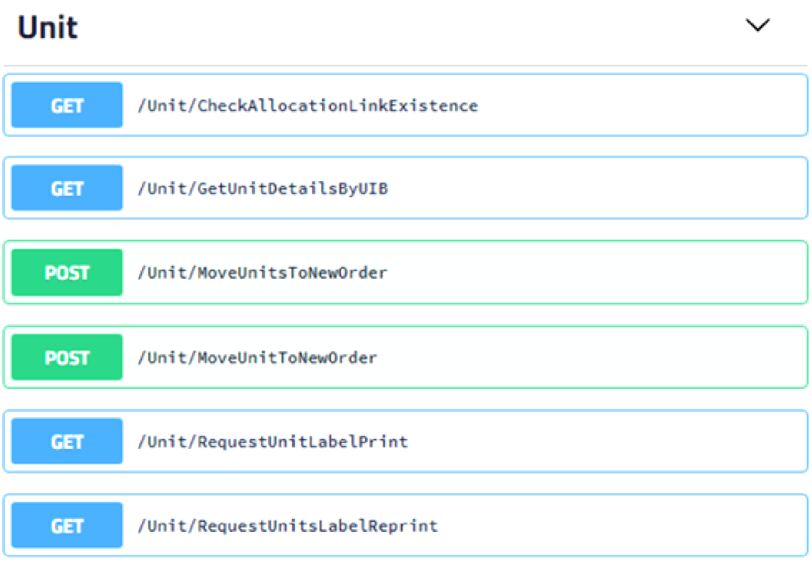
Status Code	Explanation
`200`	Used to indicate success
`201`	Resource was created successfully
`204`	No content to return
`400`	General return code for unspecified error
`401`	Problems in authentication
`403`	Access is denied
`404`	Requested resource does nott exist
`405`	Used method was not supported
`406`	Requested media type is not supported
`500`	Server-side error, i.e. a malfunction in the API

The ERP system's unit table has 130 columns, which means dozens of updates to the API may be necessary if a new column value must be returned of Postman, SwaggerHub tools. In the event that an API that allows 10 fields is originally developed, this may occur. Two additional fields are discovered to be necessary and the API is updated to reflect this. Obviously, this is not what is wanted; rather, it is preferable that all of the columns be supported right off the get. An optional parameter should be supplied in the implementation of the unit details API in order to define which fields are to be returned.

5.2 API Documentation Management

To ensure the viability of the mobile app, Tieto created a number of APIs that are now hosted on Microsoft's Azure cloud platform. See Figure 3 for an example of an API that deals with information about a single item, as well as APIs for managing individual users and their orders. Currently, the APIs can be used for both simple and sophisticated activities, such as retrieving information about users, orders, and units, and transferring units between orders. The existing APIs are not included in this thesis, but they are used as case studies in designing the administration. Swagger is used for documenting APIs; this lets users try out the APIs and offers information such as the request endpoint, parameters, and result and response codes.

Users may do things like monitor traffic, configure security policies, and discover and deploy new APIs with the help of the various parts of the system, including as the runtime, gateway, management, developer portal, and developer tools.


Unit

GET /Unit/CheckAllocationLinkExistence

GET /Unit/GetUnitDetailsByUIB

POST /Unit/MoveUnitsToNewOrder

POST /Unit/MoveUnitToNewOrder

GET /Unit/RequestUnitLabelPrint

GET /Unit/RequestUnitsLabelReprint

Figure 5.1: Simple Unit APIs [35]

API strategy definition is the first step in Tieto's API management methodology. Next, we will define API management policies and security-related challenges, and finally, we will define the setup and platform for API administration. According to Khan et al. [45], figuring out what the API does for customers is the first step. In a similar vein, it must be specified which client requirements API management meets [45]. It is an API management platform that facilitates API discovery, management, and monitoring for its user base. Customers may experience issues with common API tasks like producing use data or establishing security policies if API management is not in place.

5.3 Documentation Problems

In Table 10 reveal instances of poor documentation that Nybom et al. [15] found. We found that the well-documented examples provided nothing in the way of documentation [15]. These were all rationalized on the grounds that they satisfied the developer's informational requirements. The quality of the documentation was therefore the primary focus of our documentation.

Table 10: Categorizes the ten distinct issues into their respective forms.

Problem	Description
Content Incompleteness	An API topic or element description was not located where it should have been.
`Content Ambiguity`	The description of an API component, while detailed, was nevertheless rather vague.
`Unexplained examples`	There was insufficient explanation given for a code example.
`Content Obsoleteness`	An older version of the API was used in the topic's documentation
`Inconsistency`	Not all integration documentation was on the same page.
Incorrectness	There were several inaccuracies.
`Presentation bloat`	A subject or API element was described in too much detail.
`Fragmentation`	Too many different pages or sections had different pieces of information on the same thing.
`Excess structural information`	Repetitive information about an element's syntax or organization that can be easily acquired with modern IDEs was included in the object's definition.
`Tangled information`	Unneeded details were mixed up in the explanation of an API issue or element.

Contents: Documenting an API with the best of intentions does not guarantee that you will be able to foresee every possible use case. Understanding how to utilize an API element when it is part of a complex interaction with other API components may require more than just an explanation of its functioning. The JavaDoc API, for instance, includes features for configuring distributed caching over many nodes. The Java RMI (remote method invocation) API is required for coordinating the activities of several nodes in a distributed environment. User played around with the Postman API in order to establish a distributed caching system in Java.

Confusion was the second issue with the material. The documentation here seems to cover a topic of interest, but it actually glosses over key points, leaving room for many interpretations. Ambiguity might be seen of as a sort of incompleteness, but with crucially important details missing. The respondents' reactions varied depending on the nature of the issue. They cited a lack of knowledge as an example of incompleteness, and they cited ambiguity as an example of difficulty in understanding.

As for the third issue, missing context in the form of examples was a concern. The majority of developers do value code samples, but some respondents found the lack of context frustrating. Create your own Ajax programs with the help of the Dojo JavaScript APIs. One responder appreciated the code examples provided for numerous API functions but became upset when he or she could not figure out how to configure a parameter in the example. Because it is a JavaScript API, Dojo requires documentation that describes the code examples in terms of how they may be used in a browser.

Presentation: A bloated appearance was the first type of issue that arose during the presentation. However, when the title or header is too broad, it might be difficult for readers to identify whether or not the content they are reading actually contains the information they need. Fragmentation was the second issue with the presentation. Respondents found it inconvenient that the descriptions of the functionality and use of an API element were split out on such a granular level, requiring them to go through numerous pages of an API document. For instance, the Redocly APIs follow a two-layer description structure of each function, which is not standard in other APIs.

An excessive amount of structural detail was a third issue with the presentation. Object-oriented APIs typically have a type's description include interrelations with other types. For instance, every class in the Java SE (Standard Edition) APIs derives from java.lang.Object.

Confusing data was the last issue with the presentation. When asked whether it was beneficial to learn about APIs, respondents found that explanations that included particular examples of their use were more enlightening than those that included examples of their use in a jumbled fashion.

5.4 Problem Study

An in-depth quantitative analysis of the development of API documentation is urgently required. Due to the documentation and size of API updates, it is difficult to do such a study.

- What Sections of the API Reference Guide See the Most Updates?

  We show all possible ratios and give some instances of each. Over 2,130 versions of the java.util package and the other four J2SE libraries were compared. In Figure 2, we classify the changes made to API documentation into three broad categories and then break those categories down into more specific revision kinds. The three main types of edits are displayed vertically, while the percentage of the total 2,131 edits assigned to each kind is displayed horizontally.

Finding: Despite their similarities, tags and annotations are treated differently in the official Java documentation6. Tags allow library developers to add structure and substance to the documentation (e.g., @since), while annotations are used to alter how API elements are handled by tools and libraries (e.g., @Entity). In this study, for the sake of brevity, we do not differentiate between the two definitions and instead utilize annotation to express both.

Versions are identified by the @version or @since annotation, and they consist of a number and a date that show when an API element was generated or modified. Whenever you make a modification to the API code, any annotation will notify you and update the version of the affected API element. Due to the fact that both annotations trigger consequential automated changes to documents, they account for a disproportionate share of edits. It is still an open topic on whether such version numbers are meaningful, given we see that library developers systematically deleted the two annotations in the documentation of lucene 3.0.0.

Java's @throws annotation and @exception annotation are used to indicate that a method of an API throws an exception. Changes to the exceptions usually signify variations in behavior.

Since related API items are often referenced in the documentation for one API element. The @see or @link annotations can be used to indicate the reference relations. We find that at least two motivations push library developers to alter reference relations. One factor is that the names of referred API items are updated. It is also not easy for anyone, not even library developers, to figure out which API pieces are related to one another by reference. In addition to referencing other API components, a single document may also make use of URL referencing to make reference to other online documents. Developers of libraries can also change URLs between releases. For example, in the java.util. Locale.getISO3Language() method of J2SE 1.5, the URL of ISO 639-2 language codes is modified from one to another.

- To What Degree Do Such Revisions Indicate Behavioral Differences?

When comparing two versions of an API library, the input/output values, capabilities, and call sequences of API elements are referred to as behavioral differences. Changes to the API documentation can reflect certain alterations in behavior.

Findings: Developers of a library can modify existing API methods to throw new exceptions, and they can also create new exception documentation for these methods. To introduce a new exception (NullException) and accompanying documentation, J2SE 1.3 requires library developers to re-implement the java.util.ResourceBundle. getObject (String) method. Overall, we discover that NullException, Class CastException, and IllegalArgumentException are the most often introduced exceptions.

Library documents have the freedom to alter API method throw exceptions and update their documentation accordingly. The Vector. addAll(Collection) method, for instance, throws ArrayIndexOutOfBoundsException in J2SE 1.3. In J2SE 1.4, the thrown exception has been renamed to NullPointerException, and its documentation has been updated to reflect this.

When an API element is deprecated, developers of related libraries may substitute another API element for it. Differences in conduct are probably reflected in revised alternatives. The org.apache.lucene.analysis.StopAnalyzer documentation is one such example. In lucene 2.9.0, the StopAnalyzer(Set) constructor has been renamed to "Use StopAnalyzer(Set,boolean)instead." Use StopAnalyzer (Version,Set) instead, the lucene 2.9.1 library developers wrote. As the change notes suggest, another API method should be used in place of the deprecated one.

Changing API call sequences probably reflect observable variances in user behavior. A line like "Initially invalid, until DocIdSetIterator.next() or DocIdSet Iterator.skipTo(int) is used for the first time" may be seen in the documentation for the org.apache. lucene.search.Scorer.score() method in lucene 2.9.2. Until the first time DocIdSetIterator.nextDoc() or DocIdSet Iterator.advance(int) is used, the following phrase is considered to be "initial invalid" in lucene 3.0.0. In the updated version, programmers should use a new set of API methods.

Some API methods in documents may have their input ranges modified by library developers, who may also update their documentation. For instance, in J2SE 1.3.1, "The ASCII letters, tab, newline, and carriage return are written as, t, n, and r, respectively," can be found in the documentation for the java.util.Properties.store (OutputStream,String) method. The creators of the J2SE 1.4.2 libraries have included a new ASCII translation for "form feed," changing the text to read "The ASCII symbols, tab, form feed, newline, and carriage return are written as, t, f n and r, respectively." The change means that additional characters (like form feed) can be inserted into the newer version.

- How Frequently Are API Elements and Their Documentation Changed?

We have looked into the extensive API documentation changes and API documentation changes that have been made. Here, we summarize the number of times APIs have changed across all library versions.

Findings: The application programming interface (API) libraries of two versions that are very close together are often very different from one another. The differences in the Application Programming Interface (API) between two versions of a library are largely proportionate to the differences in the version numbers of the two versions, as well as the majority of the proportions per evolution type are accounted for by additions and alterations.

6 Chapter 6

6.1 Conclusion

Although it would be beneficial to programmers and package developers, doing an evolution study of API documentation utilizing Postman, Redocly, SwaggerHub, JavaDoc, or AutoREST tools is challenging for several reasons. As part of this initiative, we propose a system for studying API documentation in order to draw conclusions regarding its evolution. Using quantitative analysis, we investigate the evolution of API documentation for five major tools. According to the findings, API documentation is an ever-present resource. Understanding these results assists library developers in providing up-to-date documentation and provides programmers with a better understanding of the dynamic nature of API documentation.

Using the literature and the proposed framework, the results indicate that a preliminary design for API documentation can be produced. The findings can subsequently be included into API documentation tools such as Postman, Redocly, SwaggerHub, JavaDoc, and AutoREST. Not only does it outline general ideas, but it also provides particular advice for putting those principles into reality. The study's purpose of collecting data for use in comparing API documentation was met with success. Respondents placed the resolution of five content-related issues higher than the resolution of problems with Postman, Redocly, SwaggerHub, JavaDoc, or AutoREST documentation because to their frequency and prevalence. It is not surprising that the most challenging API documentation issues required the most technical expertise to overcome. It is vital to have in-depth, authoritative knowledge of the API's implementation in order to finish, clarify, and revise the documentation. This makes it difficult for non-developers and new project contributors to perform these tasks.

6.2 Future Work

Introduction and methodology sections of the paper could have provided more information. Both are broad areas that may benefit from greater study. These studies are placed aside for further research.

I have basically considered REST APIs here in this work, but there is other web APIs available as well in the market now like SOAP, GraphQL or RPC (remote procedure call). In the future, those type of APIs could be considered besides REST.

And if some kind of web or mobile application could be developed where some one could just come and put their requirements and it will suggest the available options with some sort of best possible one as well, that would be great from a user's perspective. There also could be options for putting weights based on user's need to identify the best possible option for them.

References

[1] Shi, L., Zhong, H., Xie, T. & Li, M. An empirical study on evolution of API documentation. *International Conference On Fundamental Approaches To Software Engineering.* pp. 416-431 (2011)

[2] Zhong, H. & Su, Z. Detecting API documentation errors. *Proceedings Of The 2013 ACM SIGPLAN International Conference On Object Oriented Programming Systems Languages Applications.* pp. 803-816 (2013)

[3] Stylos, J., Faulring, A., Yang, Z. & Myers, B. Improving API documentation using API usage information. *2009 IEEE Symposium On Visual Languages And Human-Centric Computing (VL/HCC).* pp. 119-126 (2009)

[4] Kramer, D. Api documentation from source code comments: a case study of javadoc. *Proceedings Of The 17th Annual International Conference On Computer Documentation.* pp. 147-153 (1999)

[5] Robillard, M. & Chhetri, Y. Recommending reference API documentation. *Empirical Software Engineering.* **20**, 1558-1586 (2015)

[6] Treude, C. & Robillard, M. Augmenting API documentation with insights from stack overflow. *2016 IEEE/ACM 38th International Conference On Software Engineering (ICSE).* pp. 392-403 (2016)

[7] Endrikat, S., Hanenberg, S., Robbes, R. & Stefik, A. How do API documentation and static typing affect API usability?. *Proceedings Of The 36th International Conference On Software Engineering.* pp. 632-642 (2014)

[8] Maalej, W. & Robillard, M. Patterns of knowledge in API reference documentation. *IEEE Transactions On Software Engineering.* **39**, 1264-1282 (2013)

[9] Inzunza, S., Juárez-Ramırez, R. & Jiménez, S. API documentation. *World Conference On Information Systems And Technologies.* pp. 229-239 (2018)

[10] Watson, R., Stamnes, M., Jeannot-Schroeder, J. & Spyridakis, J. API documentation and software community values: a survey of open-source API documentation. *Proceedings Of The 31st ACM International Conference On Design Of Communication.* pp. 165-174 (2013)

[11] Fucci, D., Mollaalizadehbahnemiri, A. & Maalej, W. On using machine learning to identify knowledge in API reference documentation. *Proceedings Of The 2019 27th ACM Joint Meeting On European Software Engineering Conference And Symposium On The Foundations Of Software Engineering.* pp. 109-119 (2019)

[12] Petrosyan, G., Robillard, M. & De Mori, R. Discovering information explaining API types using text classification. *2015 IEEE/ACM 37th IEEE International Conference On Software Engineering.* **1** pp. 869-879 (2015)

[13] Grau, J., Keilwagen, J., Gohr, A., Haldemann, B., Posch, S. & Grosse, I. Jstacs: a Java framework for statistical analysis and classification of biological sequences. *The Journal Of Machine Learning Research.* **13** pp. 1967-1971 (2012)

[14] Stylos, J., Myers, B. & Yang, Z. Jadeite: improving API documentation using usage information. *CHI'09 Extended Abstracts On Human Factors In Computing Systems.* pp. 4429-4434 (2009)

[15] Nybom, K., Ashraf, A. & Porres, I. A systematic mapping study on API documentation generation approaches. *2018 44th Euromicro Conference On Software Engineering And Advanced Applications (SEAA).* pp. 462-469 (2018)

[16] Stylos, J. & Myers, B. Mapping the space of API design decisions. *IEEE Symposium On Visual Languages And Human-Centric Computing (VL/HCC 2007).* pp. 50-60 (2007)

[17] Jansen, S. How quality attributes of software platform architectures influence software ecosystems. *Proceedings Of The 2013 International Workshop On Ecosystem Architectures.* pp. 6-10 (2013)

[18] Alvaro, A., Almeida, E. & Meira, S. Quality attributes for a component quality model. *10th WCOP/19th ECCOP, Glasgow, Scotland.* pp. 31-37 (2005)

[19] Kumar, N. & Devanbu, P. OntoCat: Automatically categorizing knowledge in API Documentation. *ArXiv Preprint ArXiv:1607.07602.* (2016)

[20] Tao, Y., Jiang, J., Liu, Y., Xu, Z. & Qin, S. Understanding performance concerns in the API documentation of data science libraries. *2020 35th IEEE/ACM International Conference On Automated Software Engineering (ASE).* pp. 895-906 (2020)

[21] Parnin, C., Treude, C., Grammel, L. & Storey, M. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute Of Technology, Tech. Rep.* **11** (2012)

[22] Cummaudo, A., Vasa, R., Grundy, J. & Abdelrazek, M. Requirements of API documentation: a case study into computer vision services. *IEEE Transactions On Software Engineering.* (2020)

[23] Leslie, D. Using Javadoc and XML to produce API reference documentation. *Proceedings Of The 20th Annual International Conference On Computer Documentation.* pp. 104-109 (2002)

[24] Postman Documenting your API. *Postman Learning Center.* (2022,7), https://learning.postman.com/docs/publishing-your-api/documenting-your-api/

[25] Nafi, K., Kar, T., Roy, B., Roy, C. & Schneider, K. Clcdsa: cross language code clone detection using syntactical features and api documentation. *2019 34th IEEE/ACM International Conference On Automated Software Engineering (ASE).* pp. 1026-1037 (2019)

[26] Parnin, C. & Treude, C. Measuring API documentation on the web. *Proceedings Of The 2nd International Workshop On Web 2.0 For Software Engineering.* pp. 25-30 (2011)

[27] Khamis, N., Witte, R. & Rilling, J. Automatic quality assessment of source code comments: the JavadocMiner. *International Conference On Application Of Natural Language To Information Systems.* pp. 68-79 (2010)

[28] Rama, G. & Kak, A. Some structural measures of API usability. *Software: Practice And Experience.* **45**, 75-110 (2015)

[29] Introduction. *Postman Learning Center.* (2022,7), https://learning.postman.com/docs/getting-started/introduction/

[30] Exploring the Public API Network. *Exploring the Public API Network.* (2022,7), https://learning.postman.com/docs/getting-started/exploring-public-api-network/

[31] Postman - Introduction *Postman - Introduction.* (2022,7), https://www.tutorialspoint.com/postman/postman_introduction.htm

[32] Stylish docs straight from API definitions *Stylish docs straight from API definitions.* (2022,7), https://redocly.com/

[33] Getting started with Redocly *Getting started with Redocly.* (2022,7), https://redocly.com/docs//

[34] Redocly tutorial authoring and publishing API docs with Redocly's command-line tools *Redocly tutorial authoring and publishing API docs with Redocly's command-line tools.* (2022,7), https://idratherbewriting.com/learnapidoc/pubapis_redocly.HTML

[35] SwaggerHub Integrations *SwaggerHub Integrations.* (2022,7), https://swagger.io/tools/swaggerhub/integrations/

[36] MY hub *MY hub.* (2022,7), https://app.swaggerhub.com/home

[37] The Java API Documentation Generator *The Java API Documentation Generator.* (2022,7), https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html

[38] Create Hosted API Documentation Online With Our API Docs Generator Tool *Create Hosted API Documentation Online With Our API Docs Generator Tool.* (2022,7), https://stoplight.io/documentation

[39] Autorest Overview *Autorest Overview.* (2022,7), https://www.autorest.io/documentation

[40] Azure Core HTTP client library for JavaScript - version 1.9.1 *Azure Core HTTP client library for JavaScript - version 1.9.1.* (2022,7), https://docs.microsoft.com/en-us/javascript/api/overview/azure/core-rest-pipeline-readme?view=azure-node-latest

[41] AutoREST API *AutoREST API.* (2022,7), https://autorest.io/documentation

[42] Henning, M. API design matters. *Communications Of The ACM.* **52**, 46-56 (2009)

[43] Meng, M., Steinhardt, S. & Schubert, A. Optimizing API Documentation: Some Guidelines and Effects. *Proceedings Of The 38th ACM International Conference On Design Of Communication.* pp. 1-11 (2020)

[44] Nasehi, S. & Maurer, F. Unit tests as API usage examples. *2010 IEEE International Conference On Software Maintenance.* pp. 1-10 (2010)

[45] Khan, J., Khondaker, M., Uddin, G. & Iqbal, A. Automatic detection of five api documentation smells: Practitioners' perspectives. *2021 IEEE International Conference On Software Analysis, Evolution And Reengineering (SANER).* pp. 318-329 (2021)

[46] Smart Bear Support *Smart Bear Support.* (2022,7), https://support.smartbear.com/swaggerhub/docs/

[47] Generating Clients with AutoREST *Generating Clients with AutoREST.* (2022,7), https://github.com/Azure/autorest/blob/main/docs/generate/readme.md